OUR SPECIAL APPRECIATION

After four years of studying at Ho Chi Minh University of Technology, especially during the semester devoted to our thesis, we have gained a wealth of academic knowledge and essential skills necessary for becoming engineers. We believe that these experiences have made us stronger and more confident in facing the challenges of lifelong learning and future professional work.

First and foremost, we would like to express our deepest gratitude to our supervisor, Associate Professor Huynh Thai Hoang, who has been a tremendous mentor throughout the completion of our graduate thesis. Without his enthusiasm, encouragement, support, and continuous optimism, this achievement would not have been possible. He has always provided us with heartfelt guidance, while also giving us the freedom to pursue our chosen topic and ensuring that we stayed focused on the core objectives of the thesis.

We also extend our sincere appreciation to all the lecturers of the Faculty of Electrical and Electronics Engineering for their inspiring lectures and valuable presentations, which have greatly supported us in our journey toward the thesis defense. We are also grateful to our friends, especially our talented classmates, whose frequent help and collaboration have significantly contributed to our learning and performance.

Finally, we owe our heartfelt gratitude to our families for their unwavering love and support. Without their nurturing, encouragement, and constant care, we would not have become who we are today.

STUDENTS

Hoang Trong Dai & Ly Kim Tien

ABSTRACT

The main objective of this thesis is to apply image processing and a fuzzy logic controller to track and follow a specific object. We designed a smart suitcase embedded with state-of-the-art object detection algorithms and modern control techniques to enable autonomous movement, including obstacle avoidance. In addition, we developed a user-friendly mobile application to help prevent the suitcase from being lost in crowded environments.

The thesis report is organized into the following chapters:

- **Chapter 1:** Introduction
- **Chapter 2:** Basic Concepts
- **Chapter 3:** Hardware Design
- **Chapter 4:** Software Design
- **Chapter 5:** Results and Conclusion

Chapter 1 introduces the motivation behind this research and reviews existing products with similar functionality, highlighting the differences from our proposed approach. Chapter 2 discusses the theoretical background and evaluates the feasibility of our techniques in real-life applications. Chapters 3 and 4 detail the hardware and software design, including the integration of Intel's latest product for deep learning model inference. Finally, Chapter 5 presents the results, summarizes the main contributions, and outlines potential directions for future development.

Contents

List of Figures

Chapter 1

INTRODUCTION

1.1 Subject Introduction

Image processing and computer vision are academic fields that enable machines to receive, analyze, and interpret visual data autonomously, and then make appropriate decisions based on that information. These technologies have been widely applied across many disciplines in modern society. However, due to their reliance on advanced mathematical computations and sophisticated algorithms, they remain challenging areas of study. Machine vision and embedded systems are expected to play a pivotal role in the next Industrial Revolution, with the potential to generate enormous economic value in the near future. Consequently, many leading corporations and research laboratories are heavily investing in this field, driven by the ambition to explore its vast potential and contribute to national development in various aspects.

In practical applications, computer vision and embedded hardware are fundamental components of Internet of Things (IoT) projects and beyond. Among their numerous tasks, tracking and detecting specific objects remain highly complex mathematical problems, yet they are crucial for a wide range of real-world applications. Examples include auto-tracking robots or consumer products such as intelligent suitcases. Inspired by this, we propose the design of a smart suitcase capable of autonomously following its user, thereby giving them greater freedom to perform other tasks while moving through crowded environments such as airport terminals.

This project represents an opportunity for us to apply the knowledge we acquired

at university, while also leveraging state-of-the-art models introduced in reputable scientific publications. We therefore name our project “Smart Suitcase”, a title that clearly reflects both our motivation and our objectives.

1.2 Objective

Traveling can be either enjoyable or stressful, and luggage often plays a decisive role in shaping that experience. While many manufacturers have introduced advancements in suitcase materials and design, the core functionality of suitcases has not changed significantly over the years. With the rapid progress of technology, however, suitcases are now becoming “smart,” offering users additional convenience during their trips. Modern smart luggage can autonomously follow its owner, allowing them to remain hands-free and focus on other activities while walking in airports or other crowded environments.

Most existing smart suitcases on the market rely on Bluetooth and GPS as their primary solutions for tracking. While effective, these technologies have limitations: such products are often expensive, and GPS-based location tracking suffers from significant errors in short-range scenarios.

Some examples of currently available smart suitcases appearing in the marketplace will be presented in the following figures.

Figure 1.1: *Arlo Skye Aluminum Carry-on ($550)*

Figure 1.2: *Barracuda Suitcase ($350)*

Figure 1.3: *G-RO Tough Terrain Expandable Carry-On ($449)*

Figure 1.4: *Samsara Smart Luggage ($435)*

The motivation of this project is to design a smart suitcase that is user-friendly, cost-effective, and capable of automatically following its owner. This device demonstrates its usefulness by maneuvering through crowded airports while its owner is engaged in other tasks, such as texting or taking care of a child.

1.3 Thesis Summary

The objective of this thesis is to develop a smart suitcase that can autonomously track and follow its owner. The intelligent suitcase relies primarily on deep learning and computer vision. Specifically, we employ the MobileNet-SSD convolutional neural network, a state-of-the-art object detection algorithm known for its efficiency and accuracy. With the support of Intel Corporation's latest high-performance Vision Processing Unit (VPU), our machine vision application is able to achieve real-time detection on a low-cost mainboard such as the Raspberry Pi. This is particularly significant, since object detection on Raspberry Pi is typically limited by its constrained memory and computational capacity.

Throughout this project, we have aimed to apply the knowledge acquired during our university studies to build a practical and valuable product. By integrating a widely used development kit such as the STM32F407-Discovery board to control the motors and handle data exchange with other boards through classical communication protocols, we were able to design a device that is both functional and efficient.

Another important consideration is the issue of luggage misplacement in crowded environments. To address this, our suitcase is equipped with a notification system that alerts the owner through a buzzer and mobile vibration signal via an Android application. Furthermore, the suitcase can provide the last known position by activating its camera to capture an image and transferring it to a server.

Chapter 2

BASIC CONCEPTS

2.1 Machine learning and object detection

2.1.1 Convolutional Neural network (CNN/ ConvNets) [9]

Convolutional Neural Networks (CNNs) consist of neurons with learnable weights and biases. Each neuron receives an input, performs a dot product operation, and, in most cases, applies a non-linear activation function. The entire network can therefore be viewed as a single differentiable function that maps raw image pixels at the input to class scores at the output. Similar to standard neural networks, CNNs employ a loss function (e.g., Softmax or SVM) at the final fully connected layer, and many of the optimization techniques developed for traditional neural networks are directly applicable.

Unlike general neural networks, ConvNet architectures explicitly assume that the input data consists of images. This prior knowledge enables the encoding of specific properties into the architecture, making the forward function more computationally efficient and significantly reducing the number of trainable parameters.

A basic ConvNet is structured as a sequence of layers, where each layer transforms one volume of activations into another through a differentiable function. The three fundamental building blocks of CNNs are the *Convolutional Layer, Pooling Layer*, and *Fully-Connected Layer*.

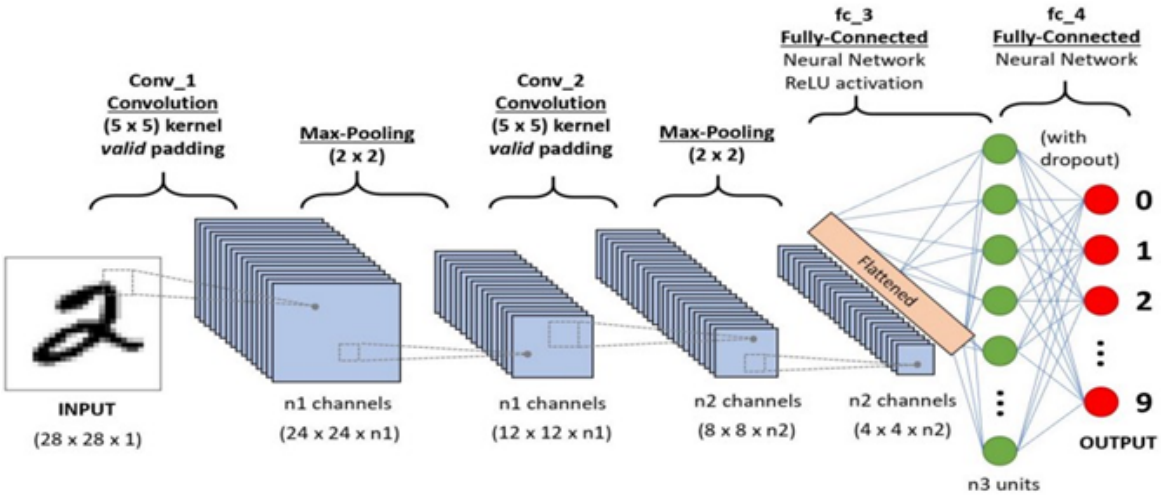


Figure 2.1: *A CNN sequence to classify handwritten digits*

2.1.1.1 Convolutional Layer

For example, input Images with dimensions Height x Width x Channel = 5x5x1. The element involved in carrying out the convolution operation in the first part of a Convolutional Layer is called the Kernel/Filter K (K = 3x3x1 in this case), represented in the color yellow.

The filter moves to the right with a certain Stride value till it parses the complete width. Moving on, it hops down to the beginning (left) of the image with the same Stride value and repeats the process until the entire image is traversed.

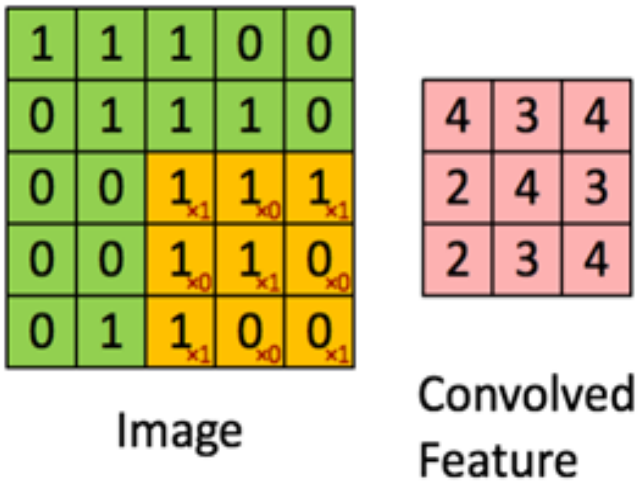


Figure 2.2: *Convoluting a 5x5x1 image with a 3x3x1 kernel to get a 3x3x1 convolved feature*

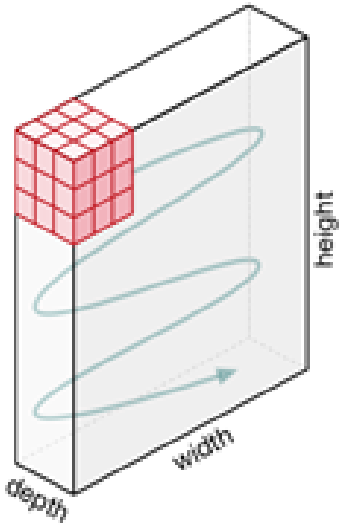


Figure 2.3: *Movement of the Kernels*

In the case of images with multiple channels (e.g. RGB), the Kernel has the same depth as that of the input image. Matrix Multiplication is performed between Kn and In stack ([K1, I1]; [K2, I2]; [K3, I3]) and all the results are summed with the bias to give us a squashed one-depth channel Convoluted Feature Output.

The objectives of the Convolution Operation is to extract the high-level features such as edges, from the input image. ConvNets may contain many Convolution Layers. Conventionally, the first Convolution Layer is responsible for capturing the Low-level features such as edges, color, gradient orientation, etc. The latter Convolution Layers will help the whole architecture to adapt to the High-level features, give it the wholesome understanding of images in the dataset.

The output of this layer will have reduction in dimensionality comparing to the input size. This depends on which kind of Padding we apply. There are two kind of padding: Valid and Same Padding. Same Padding is the method we add pixels 0 at the boundary of the input image and then apply the kernel over it. This is how output dimension can be the same as the input's one. On the other hand, Valid Padding will decrease the size of output.

The correct formula for calculating size of output dimension is:

$$\begin{cases} H' = \frac{H-F+2P}{S} + 1 \\ W' = \frac{W-F+2P}{S} + 1 \\ D' = K \end{cases} \tag{2.1}$$

Where:
(H, W, D) and (H', W', D') the dimension of input and output image respectively.
F the size of filters.
P number of padding rows or columns of zero pixels.
S stride length.
K number of applied filters.

2.1.1.2 Pooling Layer

The Pooling Layer is responsible for *reducing the spatial size of the Convolved Feature*. This is to decrease the computational power required to process data through dimensional reduction. In addition to this, this layer is useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training of the model.

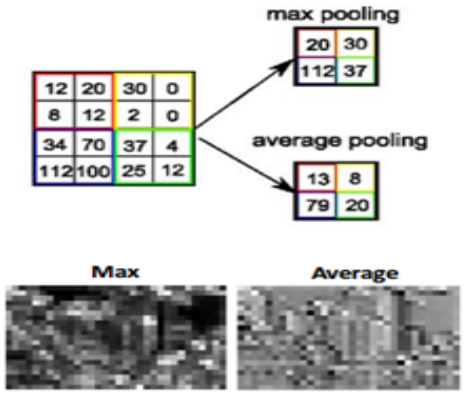


Figure 2.4: *Two types of Pooling*

There are two types of Pooling which are Max Pooling and Average Pooling. The Max Pooling returns the maximum pixel on the region of image covered by the kernel while the Average Pooling returns the average value of all pixels in that region. This means that Max Pooling provides the most important features of objects and discards

noisy elements altogether whereas Average Pooling extracts features smoothly. In image processing, Max Pooling performs a lot better than Average Pooling.

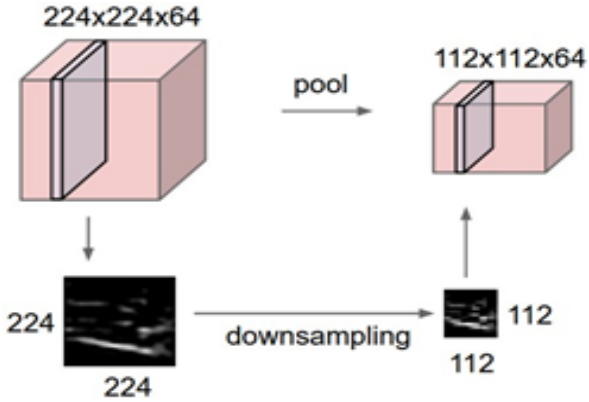


Figure 2.5: *The input volume of size [224x224x64] is pooled with filter size 2, stride 2 into output volume of size [112x112x64]*

The correct formula for calculating size of output dimension is:

$$\begin{cases} H' = \frac{H-F}{S} + 1 \\ W' = \frac{W-F}{S} + 1 \\ D' = K \end{cases} \tag{2.2}$$

Where:
(H, W, D) and (H', W', D') the dimension of input and output image respectively.
F the size of filters.
S stride length.

2.1.1.3 Fully-Connected Layer

Adding a Fully-Connected layer is a (usually) cheap way of learning non-linear combinations of the high-level features as represented by the output of the convolutional layer. The Fully-Connected layer is learning a possibly non-linear function in that space.

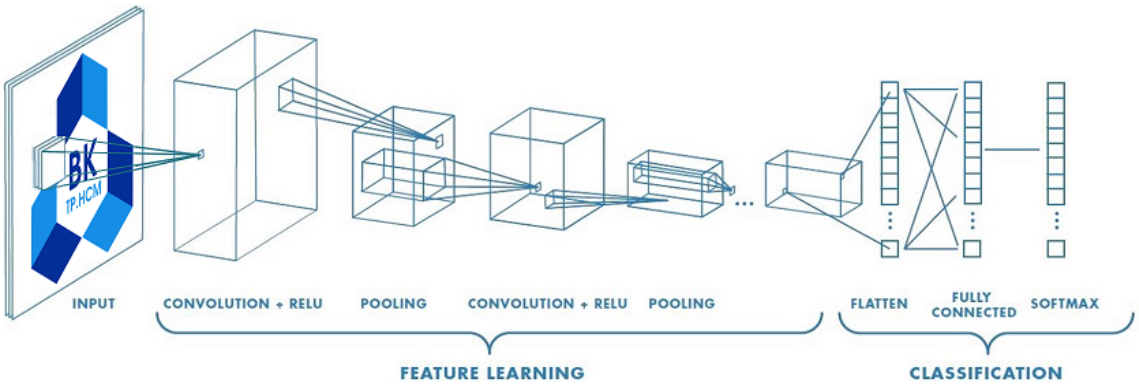


Figure 2.6: *Typical model of Convolution Neural Network*

After the Feature Extraction phase, the latest image should be flattened into column vector before being fed to a feed-forward neural network and backpropagation applied to every iteration of training. Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset. Over a series of epochs, the model is able to distinguish between dominating and certain low-level features in images and classify them using the *Softmax Classification technique*.

2.1.2 MobileNet [2]

2.1.2.1 Introduction to MobileNet Convolutional Neural Network

Convolutional Neural Networks (CNNs) have become ubiquitous in computer vision ever since AlexNet was released and won the ImageNet Challenge in 2012. The general trend in CNN development has been to design deeper and more complex networks in order to achieve higher accuracy. However, improvements in accuracy do not necessarily translate into efficiency in terms of model size and computational speed. In many real-world applications, recognition tasks must be performed in real time on platforms with limited computational resources.

In 2017, Google introduced a family of efficient models known as *MobileNets*, specifically designed for mobile and embedded vision applications. MobileNets are built on a streamlined architecture that employs *Depthwise Separable Convolutions*, enabling lightweight neural networks that achieve an effective trade-off between la-

tency and accuracy.

Figure 2.7: *MobileNets are compatile with mobile applications*

2.1.2.2 Depthwise Separable Convolution

The MobileNet model is based on *Depthwise Separable Convolutions*, which are a form of factorized convolutions. Instead of performing a standard convolution directly, the operation is decomposed into two separate steps: a *Depthwise Convolution* followed by a 1×1 convolution, also called a *Pointwise Convolution*.

In this architecture, the *Depthwise Convolution* applies a single filter to each input channel independently, thereby performing channel-wise spatial filtering. The *Pointwise Convolution*, on the other hand, uses 1×1 kernels to linearly combine the outputs from the depthwise stage across all channels.

Thus, a *Depthwise Separable Convolution* consists of two layers with distinct roles:

- A filtering stage (*Depthwise Convolution*)
- A combination stage (*Pointwise Convolution*)

This factorization greatly reduces the computational cost and the number of parameters, while maintaining competitive accuracy compared to standard convolutions.

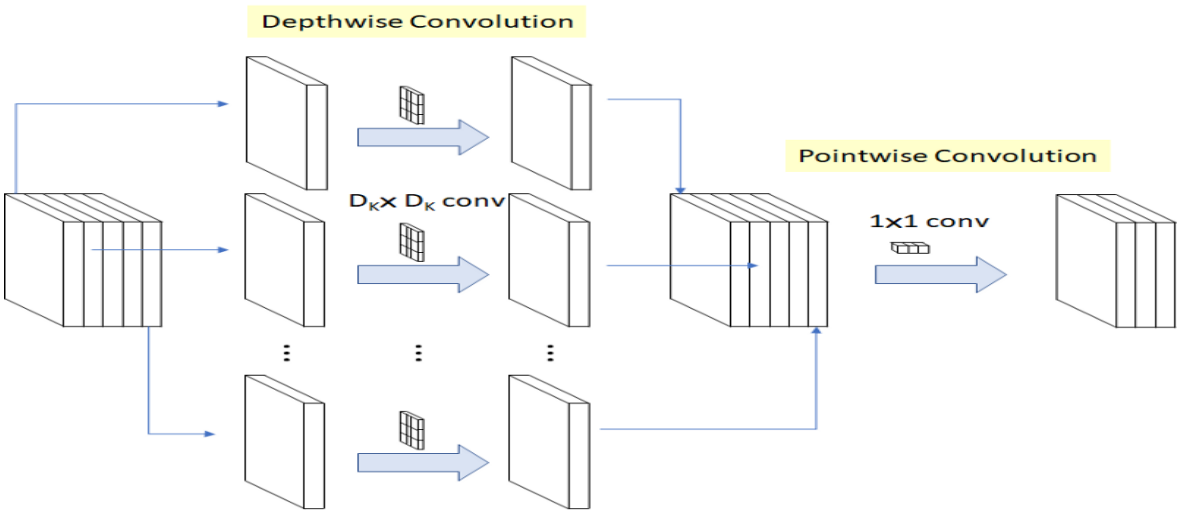


Figure 2.8: *Depthwise Separable Convolution includes two stages of convolutions*

2.1.2.3 Why MobileNet is an efficient convolutional neural networks?

To have solution to this big question, we have to consider the cost function of Depthwise Separable Convolution and a Standard Convolution.

Cost Function of Depthwise Separable Convolution:

- Depthwise Convolution: Filtering Stage
  - Consider the input tensor with the shape (D_F, D_F, M) and the output tensor has the shape (G_G, G_G, M). The kernel in this stage has the shape (D_K, D_K, 1) and there are M kernels like this applied along the depth of input tensor.

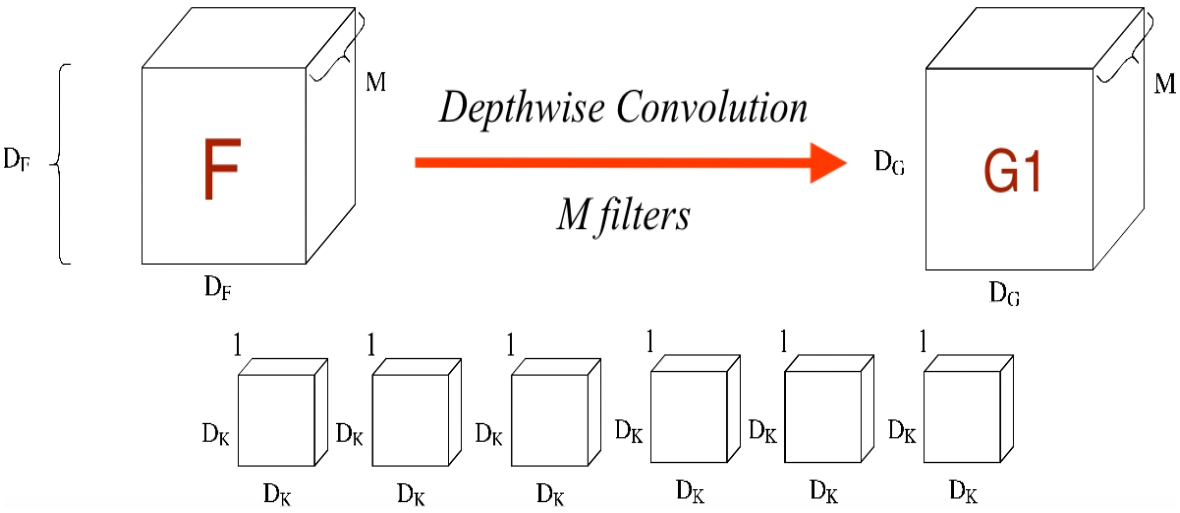


Figure 2.9: *Depthwise Convolutions*

  - Number of multiplications in this stage can be computed as the following:
    * Number of multiplications on one convolution operation is D_K^2.
    * Number of multiplications on one channel is $D_G^2.D_K^2$.
    * Number of multiplications of Depthwise Convolution is $D_G^2.D_K^2.M$.

- Pointwise Convolution: Combination Stage
  - Consider the input tensor with the shape (D_G, D_G, M) and the output tensor has the shape (D_G, D_G, N). The kernel in this stage has the shape (1, 1, M) and there are N kernels like this applied in convolution operations with the input tensor. The output tensors of this stage eventually have height and width of input tensors as well as the depth of kernels..

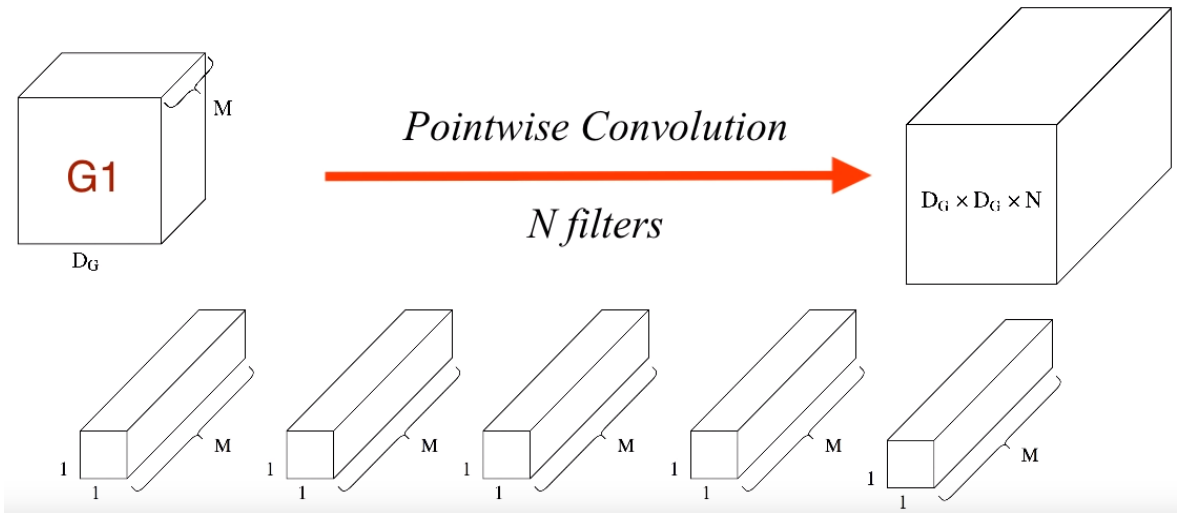


Figure 2.10: *Depthwise Convolutions*

- Number of multiplications in this stage can be computed as the following:
  - Number of multiplications on one slide of convolution operation is M.
  - Number of multiplications of one kernel is $D_G^2.M$.
  - Number of multiplications of Pointwise Convolution is $D_G^2.M.N$.

$\rightarrow$ Thus, total multiplication of Depthwise Separable Convolution is: #DSC_Multiplications = #DC_Multiplications + #PC_Multiplications = $D_G^2.D_K^2.M$ + $D_G^2.M.N = D_G^2.M.(D_K^2 + N)$ (1)

Cost Function of Standard Convolution: Number of multiplications can be computed as the following:

- Number of multiplications of a convolution operation is $D_K^2.M$.
- Number of multiplications of a convolution operation is $D_G^2.D_K^2.M$.
- Number of multiplications of a convolution operation is #SC_Multiplications = $D_G^2.D_K^2.M.N$ (2).

$\rightarrow$ By expressing convolutions as a dual step process of filtering and combination, we will get the reduction in computation of:

$$\frac{\#DSC\_Multiplications}{\#SC\_Multiplications} = \frac{(1)}{(2)} = \frac{D_G^2.M.(D_K^2+N)}{D_G^2.D_K^2.M.N} = \frac{1}{N} + \frac{1}{D_K^2} \tag{2.3}$$

By taking an example of a convolutional operation that has output feature volume is 1024 ($N = 1024$) and kernel size is 3 pixels ($D_K = 3$) then

$$\frac{\#DSC\_Multiplications}{\#SC\_Multiplications} = \frac{1}{N} + \frac{1}{D_K^2} = \frac{1}{1024} + \frac{1}{3^2} = 0.112 \tag{2.4}$$

This means that we have decreased the amount of multiplications 9 times lighter by applying dual stages of Depthwise Separable Convolution. Therefore, MobileNets has become an efficient convolutional neural network for vision applications on mobile devices.

2.1.2.4 Network structure

The MobileNet structure is built on Depthwise Separable Convolutions as mentioned in the previous section except for the first layer which is a full convolution.

The MobileNet architecture is defined in the following table. All layers are followed by a Batch Normalization and ReLU nonlinearity with the exception of the final fully connected layer which has no nonlinearity and feeds into a Softmax Layer for classification.

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Figure 2.11: *MobileNet body architecture*

The contrast between Septhwise Separable Convolutions can be visualized with the following figure.

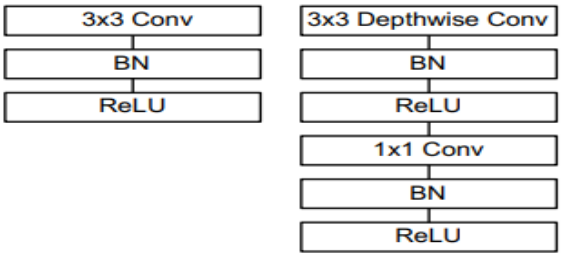


Figure 2.12: *Difference between standard (left) and Depthwise Separable (right) Convolution*

2.1.3 SSD: Single Shot Detector [3]

2.1.3.1 Introduction of SSD

SSD or Single Shot Detector is a method for detecting objects in images using a single deep neural network. By using SSD, user only need to take one single shot to detect multiple objects within the image, while the regional proposal network (RPN) based approaches such as R-CNN need two shots: one for generating region proposals, one for detecting the object of each proposal. Therefore, SSD is much faster than the other two-shot approaches.

A scientific paper which is published on December 2016 provided a reliable results that compare the performances of different approaches on Pascal VOC 2007 test. As can be seen from the table below that SSD300 is the only real-time detection method that can achieve 70% mean Average Precision (mAP). By using a larger input image, SSD512 outperform all methods on accuracy while maintaining a close to real-time speed.

Method	mAP	FPS	batch size	# Boxes	Input resolution
Faster R-CNN (VGG16)	73.2	7	1	~ 6000	$\sim 1000 \times 600$
Fast YOLO	52.7	155	1	98	448×448
YOLO (VGG16)	66.4	21	1	98	448×448
SSD300	74.3	46	1	8732	300×300
SSD512	76.8	19	1	24564	512×512
SSD300	74.3	59	8	8732	300×300
SSD512	76.8	22	8	24564	512×512

Figure 2.13: *Results on Pascal VOC 2017 of different methods [3]*

2.1.3.2 MultiBox detector

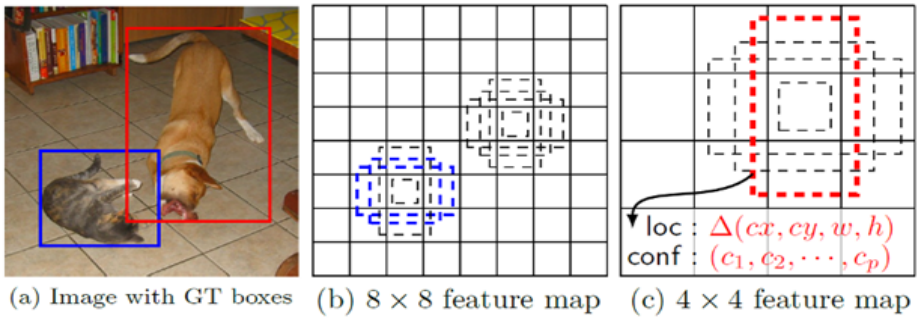


Figure 2.14: *Multiple Bounding Boxes for Localization (loc) and Confidence (conf) [3]*

After going through a certain of convolutions for feature extraction, we obtain a feature layer of size m×n (number of locations) with p channels, such as 8×8 or 4×4 above. And a 3×3 conv is applied on this m×n×p feature layer.

For each location, we got k bounding boxes. These k bounding boxes have different sizes and aspect ratios. It is the concept that is maybe a vertical rectangle is me fit for human while the horizontal one is for the car. For each of the bounding box, we will compute c class scores and 4 offsets relative to the original default bounding box shape.

→ Therefore there are (c + 4)kmn outputs.

2.1.3.3 SSD Architecture

SSD is a single shot detector using VGG16 network as a feature extractor and some custom convolution layers and filters to make prediction. The following figure is an example of simple detector architecture.

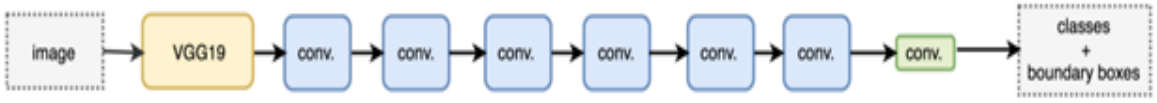


Figure 2.15: *Simple CNN architecture for prediction and localization [11]*

The problem with the above-mentioned network is using many convolutional layers that reduce spatial dimension and resolution. To be more precise, the model above

can only detect large objects only. To fix that, SSD model makes independent object detections from multiple feature maps.

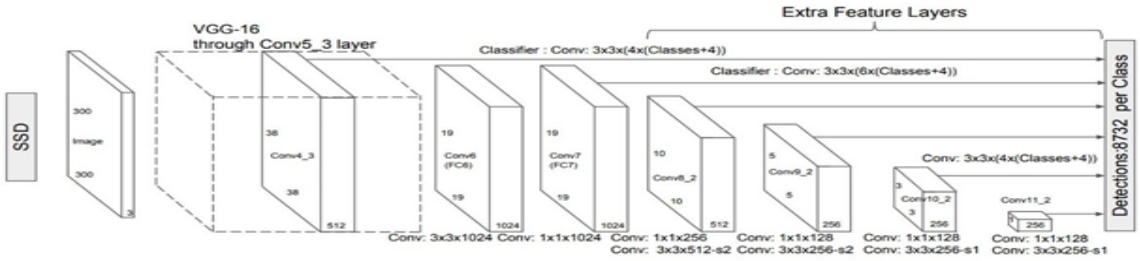


Figure 2.16: *SSD architecture [3]*

SSD uses layers already deep down into the convolutional network to detect objects. But the above obstacle still appear. The first layer of object detection (conv4_3) has spatial dimension of 38x38 which is a really large reduction from the source image. Therefore, SSD usually performs badly for small object detection when comparing with other method. The other solution for this problem is to use input with high resolution.

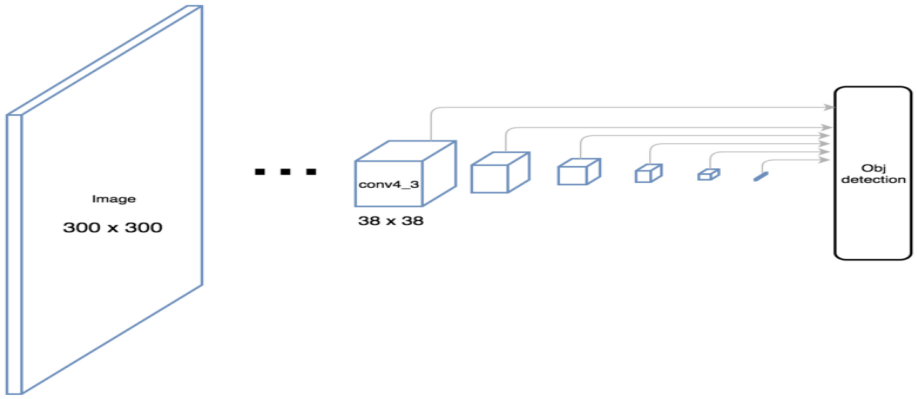


Figure 2.17: *Reduction of spatial dimension make SSD hard for small object detection [11]*

2.1.4 Lightning Memory-Mapped Database (LMDB)

Lightning Memory-Mapped Database (LMDB) is a software library that provides a high-performance embedded transactional database in the form of a key-value store and it uses B+ tree to manage data.

LMDB bases on Single-Level Storage (SLS) of MULTICS (MULTIplexed Information and Computing Service) Operation System. SLS stores data in bytes (without

Type and Structure of them). All data files of Lightning on Disk will be mapped to Memory by mmap() and Read/Write actions will be executed into Mapped-Files in Virtual Address. The advantage of SLS and Mmap is the ability to skip Cache, Buffer and Memcpy and hence Lightning can use Memory with the highest efficiency and Read/Write Performance is also well improved.

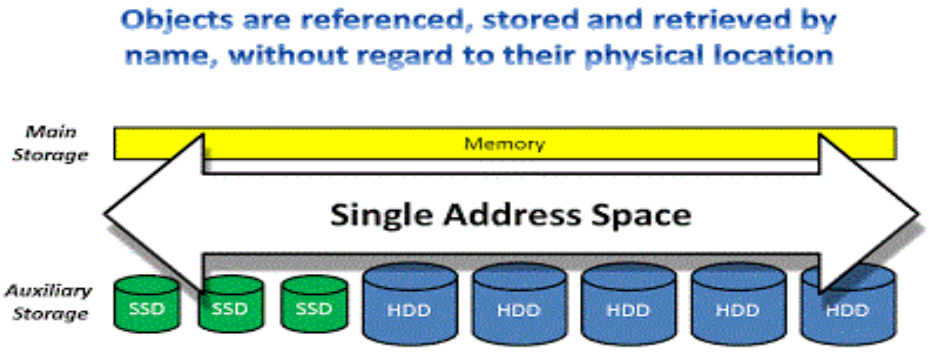


Figure 2.18: *Single-Level Storage [12]*

Another feature of LMDB is to implement Multiversion Concurrency Control (MVCC). MVCC concurrency model is enforced using a single writer (serialized) and the copy-on-write semantics to handle writes to the database. Readers do not block writers and writers do not block readers, and since there is single writer to the database, there will not be any deadlocks. MVCC makes sure that each accessing thread to database always gets the latest data without waiting for other modification thanks to copy-on-write semantics (or snapshot isolation). To be more precise, when a modification occurs, a copied version of data will be created, and this leads to the fact that data in real-time cannot be overwritten.

There are three plus points of LMDB: Compact, Reliability and Performance. Compact characteristic of LMDB is its light size on disk (nearly 40KB) and can be applied on different Operation System including Linux, Windows, MacOS, Solaris, etc. In addition to this, LMDB had passed many test case with no failure. In fact, copy-on-write semantics mentioned above is the key point for this success. The final point is about excellent performance of Read/Write speed which is more outperformed than other kinds of database.

2.1.5 Google Colaboratory

Google Colaboratory is a Google research project created to help disseminate machine learning education and research. It's a Jupyter notebook environment that requires no setup to use and runs entirely in the cloud. It also gives users a total of 12 GB of ram and up to 12 hours of continuous use on Tesla K80 GPU.

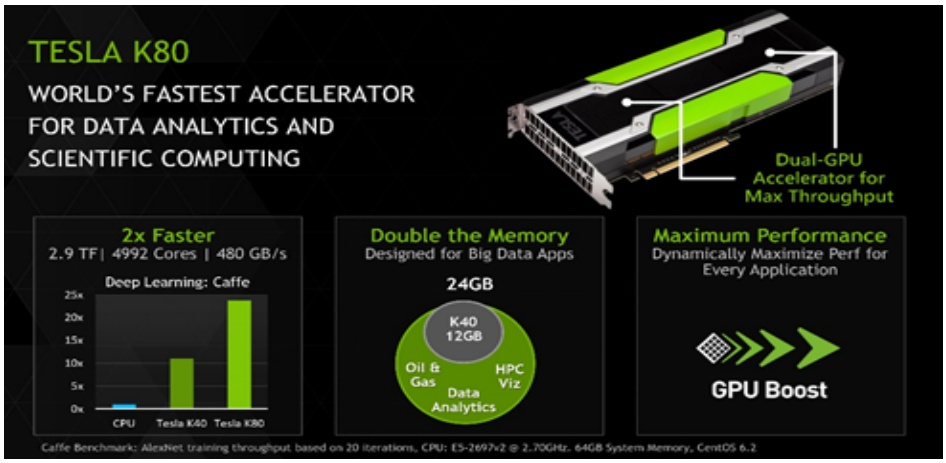


Figure 2.19: *GPU embedded inside Google Colaboratory*

CPU vs. GPU in Deep Learning, why choose GPU? CPUs and GPUs process tasks in different ways. Regarding interrelations, they are often compared with brain and brawn. A CPU (the brain) can work on a variety of different calculations, while a GPU (the brawn) is best at focusing all the computing abilities on a specific task. That is because a CPU consists of a few cores (up to 24) optimized for sequential serial processing. It is designed to maximize the performance of a single task within a job; however, the range of tasks is wide. On the other hand, a GPU uses thousands of smaller and more efficient cores for a massively parallel architecture aimed at handling multiple functions at the same time.

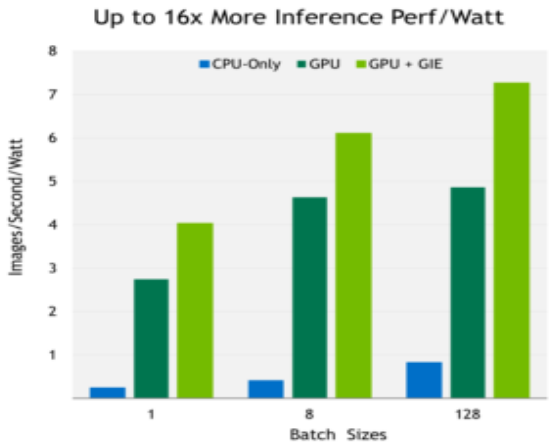


Figure 2.20: *NVIDIA GPU Inference Engine (GIE) provides even higher efficiency and performance for neural network inference [13]*

Modern GPUs provide superior processing power, memory bandwidth and efficiency over their CPU counterparts. They are 50–100 times faster in tasks that require multiple parallel processes, such as machine learning and big data analysis.

2.2 Fuzzy controller

Fuzzy logic is applied with great success in various control application. Almost all the consumer products have fuzzy control. Some of the examples include controlling room temperature with the help of air-conditioner, anti-braking system used in vehicles, control on traffic lights, washing machines, large economic systems, etc.

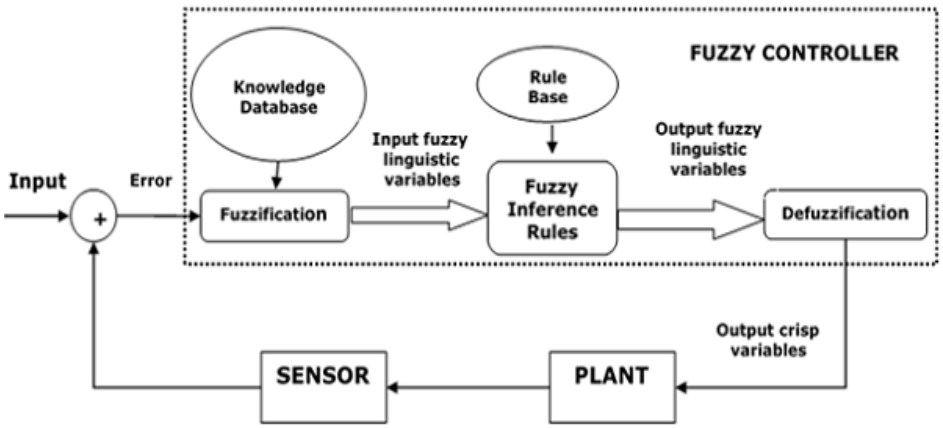


Figure 2.21: *Fuzzy Logic Controller architecture*

There are two types of techniques for constructing fuzzy models: Mamdani and Sugeno. The Sugeno method works well with linear techniques, as it is computationally efficient. It is suitable to apply optimisation and adaptive techniques. Furthermore, it guarantees continuity of the output surface and it is well suited to mathematical analysis. An advantage of the Mamdani method is that is it intuitive and has widespread of acceptance. It can be well suited to human input.

Defuzzification task is to find one single crisp value that summarises the fuzzy set. There are several mathematical techniques available: centroid, bisector, mean, maximum, maximum and weighted average. The following figure demonstrate illustration of how values for each method are chosen.

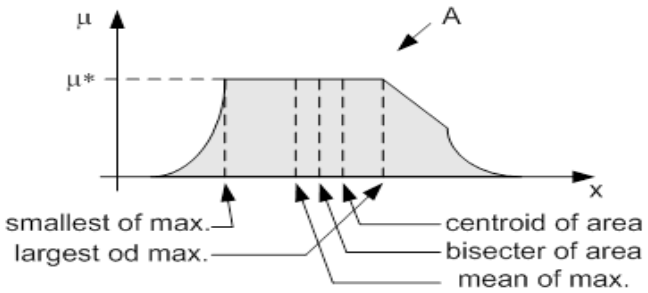


Figure 2.22: *Graphical demonstration of defuzzification methods*

2.3 Obstacle avoidance algorithm

2.3.1 Overview

Obstacle avoidance plays a key role in most mobile robots. With given knowledge or sensorial information of the surrounding environment, obstacle avoidance helps robots move towards without any collisions. Generally, for moving safely to the goal, the global path planning needs to be associated with a local obstacle handing that involves obstacle detection and obstacle avoidance. Without having any given map, there are many techniques to build an obstacle-avoiding robot based on the environment information collected by a camera, an ultrasonic sensor, a laser sensor or an infrared sensor. Let's have a brief overview of popular existing obstacle avoidance algorithms:

- **The Bug Algorithms** [4]: This is a simple approach by following the boundary

of the obstacle. The Bug 1 is a full contour around the object to evaluate the point with shortest distance to the goal, while the Bug 2 is a definition of the leaving point where the boundary intersects the line segment that connects the start point to the target.These algorithms do not take robot kinematics into account and sensor noise also impacts the robot performance.

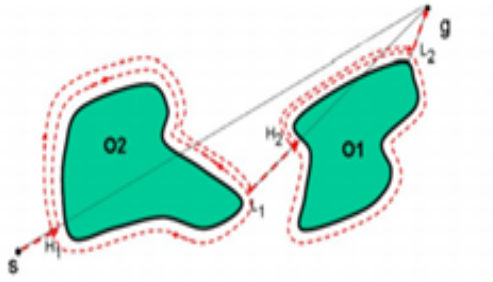


Figure 2.23: *Obstacle avoidance with Bug 1 algorithm [5]*

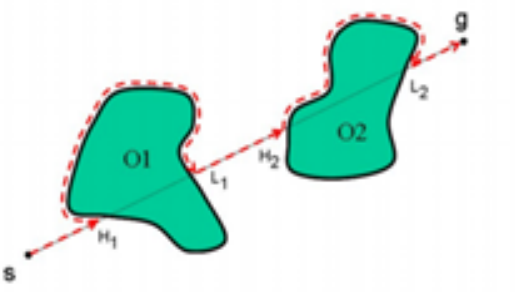


Figure 2.24: *Obstacle avoidance with Bug 2 algorithm [5]*

- **The Potential Field Algorithm** [6]: The robot immersed in a potential field is subject to the action of a force that drives it to the goal (the attractive potential) while keeping away from the obstacle (the repulsive potential).

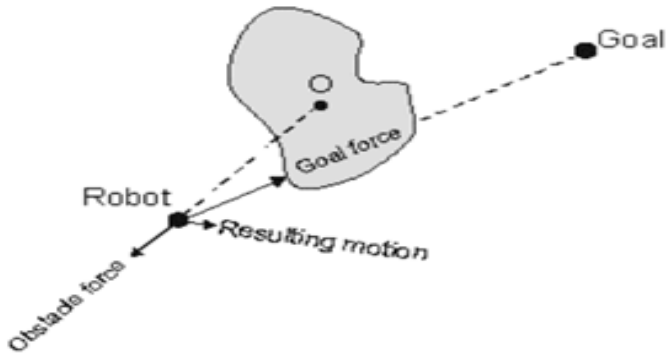


Figure 2.25: *Obstacle avoidance with potential field algorithm [7]*

Though the potential field algorithm is simply implemented, it also has difficulty in dealing with a concave obstacle and using in real time applications.

- **The Vector Field Histogram Algorithm (VFH)** [8]: The VFH method maps a two-dimensional Cartesian histogram grid of obstacle representation onto a 1D structure known as a polar histogram. The x-axis of the polar histogram represents the angular sector and the y-axis shows the probability that there is an

obstacle in that direction. Based on the polar histogram, the VFH evaluates the steering angle by setting a threshold for "valleys" and minimizing the cost function.

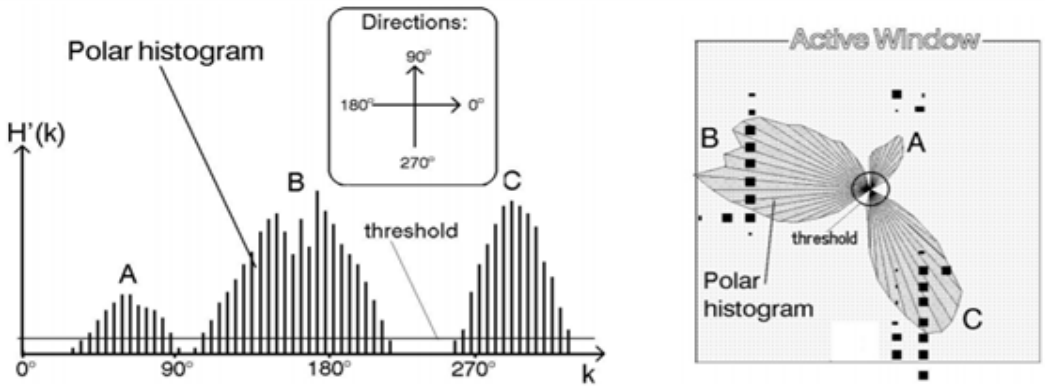


Figure 2.26: *Polar histogram of the VFH for obstacle avoidance [7]*

This optimized algorithm takes into account the kinematics of the robot, however, there is limitation with a large computation load.

2.3.2 The simple, real-time obstacle avoidance

2.3.2.1 Ultrasonic sensor

Ultrasonic sensor is widely used for distance measurement. An ultrasonic sensor uses transducers to send and receive the ultrasonic sound, which vibrates at a frequency above the range of human hearing. It determines the distance to a target by measuring time lapses between the sending and receiving of the ultrasonic pulse.

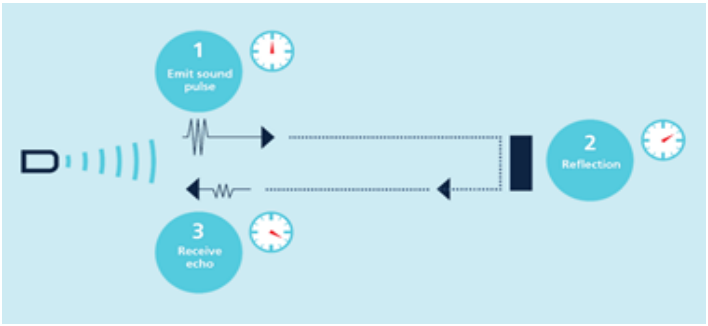


Figure 2.27: *Ultrasonic principle*

Ultrasonic sensor is suitable for target distances from 20 mm to 10m with high accuracy. It is also capable of detecting all materials which reflect sound and not affected by atmospheric dust, rain or snow. However, it has difficulties in working with soft, curved and thin objects. There are also some cases that the sensor cannot detect objects.

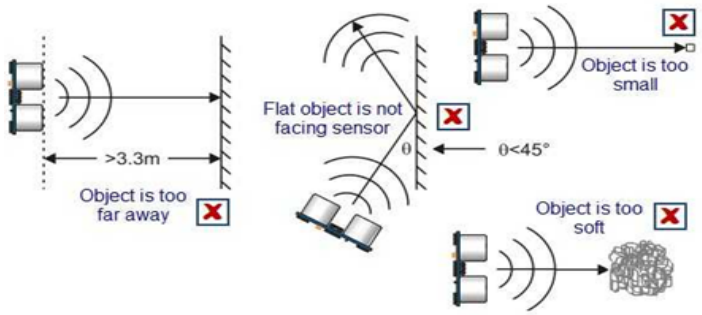


Figure 2.28: *Examples of situations where an ultrasonic sensor fails to detect objects*

Steps to read an ultrasonic sensor:

- Send a 10us pulse to the sensor on the Trigger Pin, the sensor will automatically send out an ultrasonic wave.
- Begin monitoring the Echo Pin.
- When the Echo Pin goes high, begin a timer.
- When the Echo Pin goes low, get the elapsed time and measure distance using the formula: $distance(cm) = \frac{elapsed\_time(\mu s) * 0.0001 * 340}{2}$

2.3.2.2 The proposed algorithm

- The required robot for this algorithm has a ring of ultrasonic sensors covering an angle of 180 degrees.
- The algorithm defines a sensitivity bubble over the range of sonar sensors, which is used for obstacle detection.
- The readings of the sensors represent the distance between the position of the sensors and the obstacles and are shown in a polar diagram.

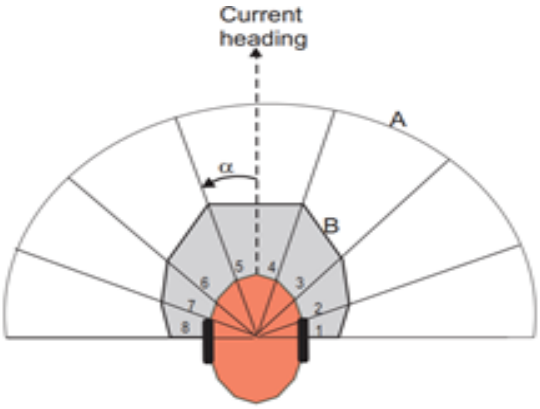


Figure 2.29: *An example of a robot with ultrasonic sensors and defined sensitivity bubble [7]*

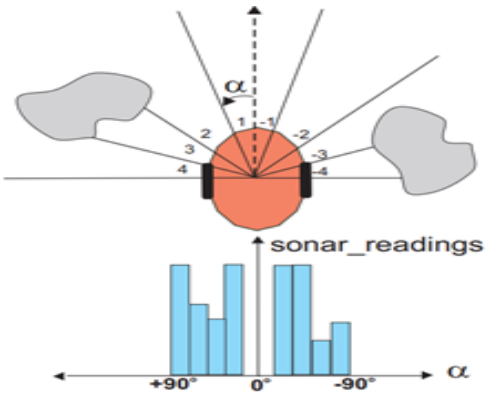


Figure 2.30: *Polar diagram of the sonar readings [7]*

- At first, the robot moves towards the goal. If an obstacle is detected within the sensitivity bubble, it will head to the area having the lowest density of obstacles according to the polar diagram. Then it will follow that direction until the goal becomes visible or until a new obstacle is found.

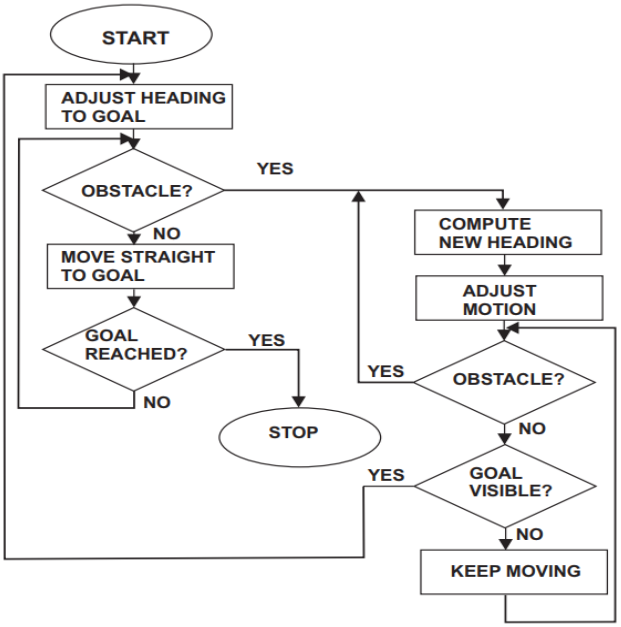


Figure 2.31: *Flow chart for the bubble rebound algorithm [7]*

- Consider N is the total number of the ultrasonic sensors, which are uniformly distributed at an angular pace: $\alpha_0 = \frac{\pi}{N}$. And the solar index, i, contains angular information: $\alpha_i = i * \alpha_0$, $i \in \left[\frac{-N}{2}, \frac{N}{2}\right]$. The rebound angle is measured by

$$\alpha_R = \frac{\sum_{i=\frac{-N}{2}}^{\frac{N}{2}} \alpha_i * D_i}{\sum_{i=\frac{-N}{2}}^{\frac{N}{2}} D_i} \text{ where } D_i \text{ is the distance found by the sensor } i.$$

- **Advantages and disadvantages:**
  - **Advantages:**
    - It demands low computational load.
    - It can avoid both static and moving obstacles.
    - It can be implemented for many kinds of sensors.
  - **Disadvantages:**
    - The motion is not smooth.
    - Motion is done even if there is no path to goal.
    - It requires a high level path planner to perform well.

2.4 Data communication

2.4.1 SPI communication protocol

- Serial Peripheral Interface (SPI) is a protocol for communication between electronic devices. Communication via SPI involves a master-slave relationship.

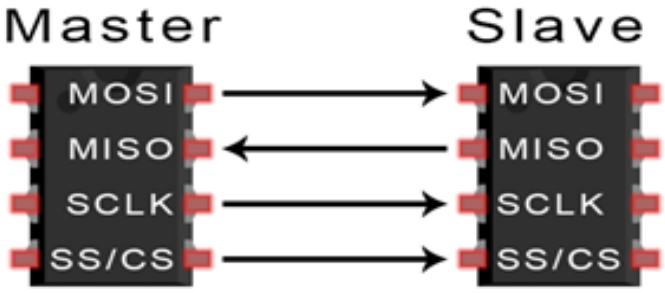


Figure 2.32: *Simple configuration of SPI*

  - **MOSI**: Line for data from master to slave.
  - **MISO**: Line for data from slave to master.

  - **SCLK**: Line for clock signal, which determines the speed of data transfer through sending one bit of data per clock cycle.
  - **SS/CS**: Line for master to select which slave to send data to. The master chooses its slave by setting the slave's CS/SS to a low voltage level.
- Steps of SPI data transfer:
  - The master outputs the clock signal.
  - The master activates the slave by switching the SS/CS pin to a low voltage state.
  - The master sends one bit at a time to the slave along the MOSI line
  - The slave receives and responses to the master through the MISO line if needed.
- **Advantages and disadvantages:**
  - **Advantages:**
    * Data can be transferred continuously without interruption.
    * Data can be sent and received at the same time.
  - **Disadvantages:**
    * Uses more wires that other communication protocols like I2C, UART.
    * Can not perform data checking.

2.4.2 Client-server communication

Client-server is a relationship where one program (client) requests a service or resource from another program (server). A client-server communication typically consists of one or multiple clients in communication with a single server via the Internet.

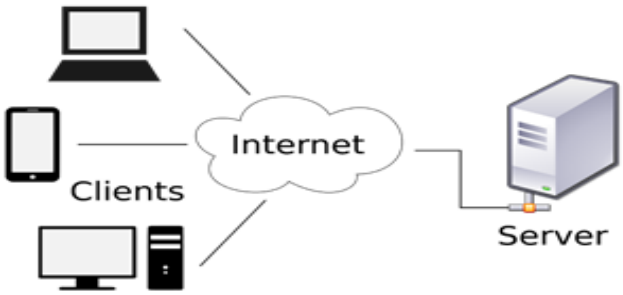


Figure 2.33: *A network diagram of clients and a server*

In order to exchange messages over the Internet, we need a communication protocol to define a common language so that the server and client(s) can understand each other. There are two common IP communication protocols including TCP (Transmission Control Protocol) and UDP (User Datagram Protocol). In this thesis, we will implement TCP, the most used protocol.

TCP is a connection-oriented protocol working with an IP, which is the reason for the term "TCP/IP". In TCP, the receiver sends back the acknowledgement of the received packet so that the sender can resend if the response does not come or any packet error is generated. Especially, TCP packet involves error checking to avoid losing data if there is any connection lost. All of which makes TCP a more reliable and secure protocol than UDP.

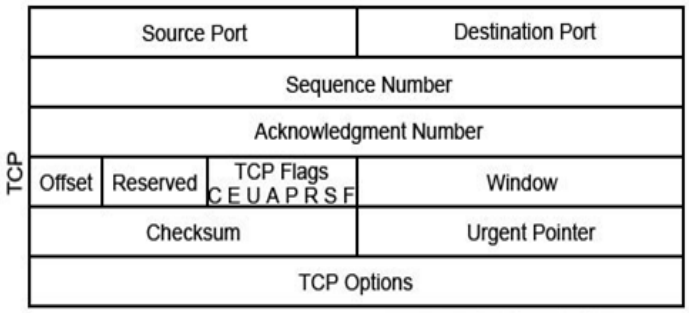


Figure 2.34: *TCP Packet*

The socket is used in TCP in order to find the exact end-to-end connection. It is an endpoint of a connection between two processes. Each side of a socket connection uses a specific port number which does not change during the life of that connection. The port number together with the IP address uniquely identify an endpoint.

2.5 Parallel Programing

Our application is to detect desired object at real-time and the data is increasing exponentially with time. The problem we encountered is the processor's computing power cannot deal with data collecting and image processing at the same time. There should be the way to process data efficiently and effectively. The only solution is able to solve it is parallel programing. Fortunately, Raspberry Pi's CPU is Quad-Core ARM, this means that we can run 4 processes at a time.

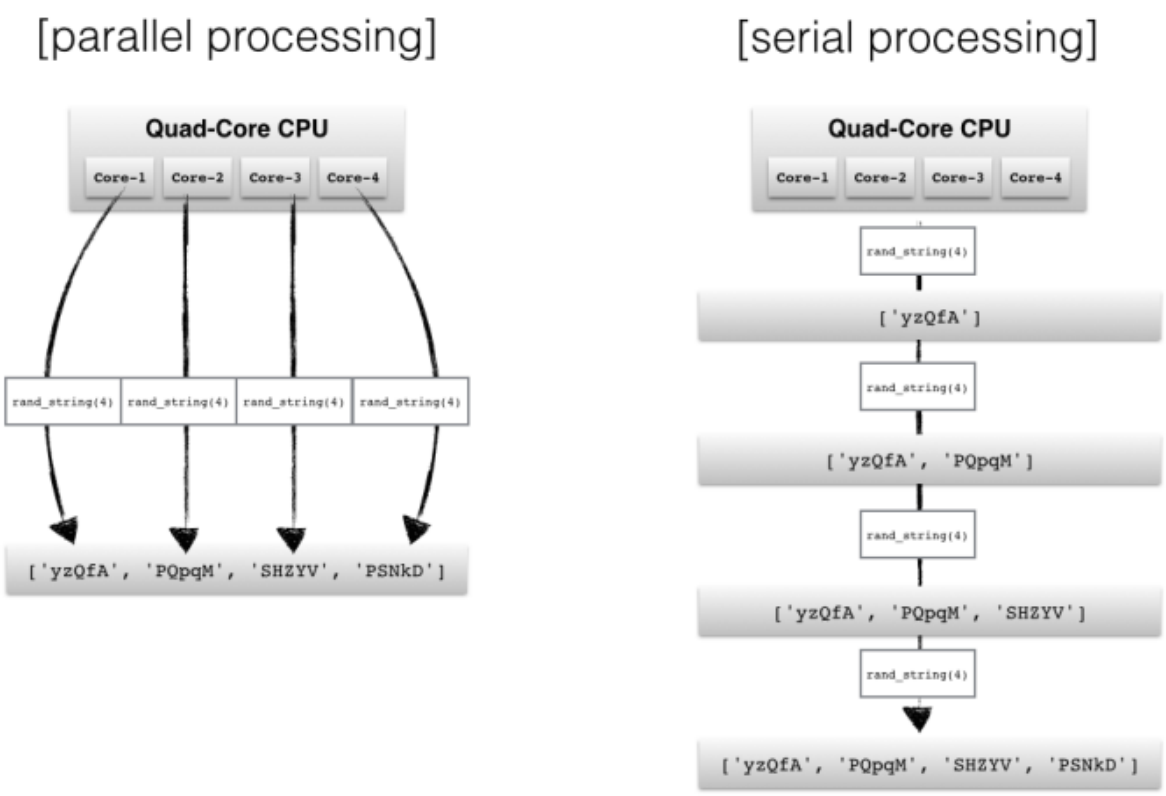


Figure 2.35: *An example of Parallel and Serial Processing*

2.5.1 Multi-Threading Vs. Multi-Processing

In this range of application, these two common approaches of parallel programming are either used. But there are some different points of these methods:

- Multi-Processing is sharing of computer resources among the number of processes or execution of different processes at the same time whereas Multi-Threading is the execution of the number of different tasks within the same process.
- If many tasks are submitted to different threads, these tasks can be seen as subtasks of a single process and those threads will usually have access to the same memory area. To be more precise, different threads use the shared memory and this can lead to the conflicts in case of improper synchronization. A safer approach would be to submit multiple processes to complete separate memory locations. Each process will run independently from each other.

In object detection application, multiple processes are used for different tasks such as: data collecting, image processing, data communication. Threads only used for

computations in object detection in real-time video streaming and depend on number of Movidius Neural Compute Sticks applied in this application. Specifically, if only single NCS is used, only one thread will run.

2.5.2 Problem of passing data between processes

As mentioned above, multiple processes use different memory location so that two same named variables do not have same value. A very good method for sharing data between processes Queue which is supported by multiprocessing library in Python.

Queue is an abstract data structure, somewhat similar to Stacks, but unlike Stacks, a Queue is open at both its ends. One end is used to insert data (enqueue) while the other one is used to remove data (dequeue). Queue follows FIFO (First-In-First-Out) methodology – item stored first will be accessed first. Queue objects are always used for thread and process safe which make them perfect for passing data without potentially corrupting data.

Figure 2.36: *Queue Representation*

Chapter 3

HARDWARE DESIGN

3.1 Overview

Figure 3.1: *Suitcase model*

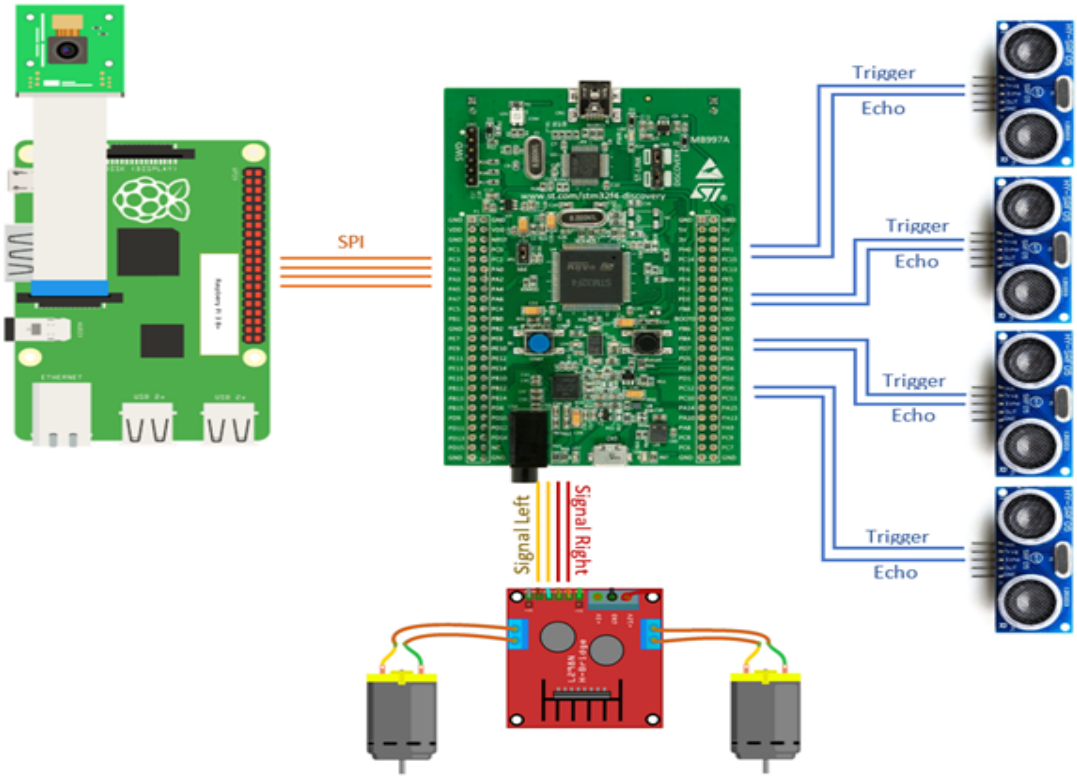


Figure 3.2: *Hardware overview*

In our model, Raspberry Pi is used for object detection based on computer vision. It is equipped with a camera for recording real-time scene and an Intel Movidius Compute Stick for boosting the speed of image processing. Additionally, Raspberry Pi sends data to ARM using SPI communication protocol, which consists of 4 wires: MOSI, MISO, SCLK and SS/CS.

STM32F4's duty is to read four ultrasonic sensors and control the motion of the suitcase. Regarding sensors readings, STM32F4 uses two wires in connection with every sensor: one for sending trigger pulse and one for receiving echo pulse. In order to control the two motors, STM32F4 transmit signals through four wires, a PWM and a compared pulse for each motor.

For powering devices, we use a portable power bank which supplies 5V for both Raspberry Pi and ARM STM32F4. A lipo battery provides 12V energy source for the driver and also, through the power converter to power four ultrasonic sensors with 5V.

3.2 Components details

3.2.1 Raspberry Pi 3 B+

Figure 3.3: *Raspberry Pi 3 B+[19]*

Raspberry Pi is a low cost, credit-card computer that acts as a desktop computer with basic applications like web browser, text editor, video player, games. It runs mainly on Raspbian, which is a Debian-based computer operating system. Raspberry Pi aims to educate people in computing and can also be used for electronics devices ore home automation projects.

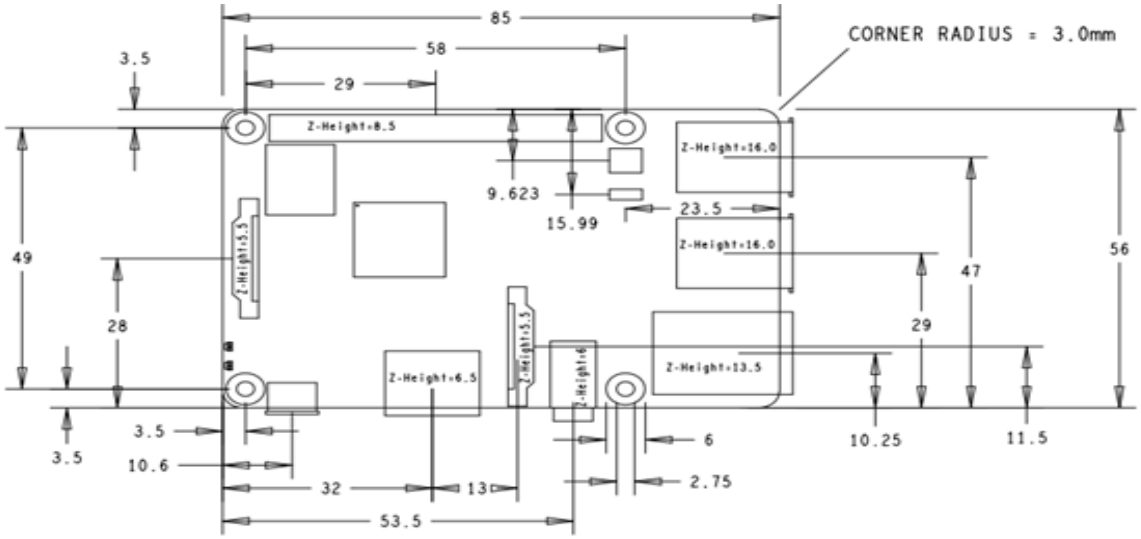


Figure 3.4: *Mechanical drawings of Raspberry Pi 3 Model B+ [19]*

Raspberry Pi 3 Model B+ is the latest version of Raspberry Pi with:

- 1.4GHz 64-bit quad-core processor.
- 1GB SDRAM.
- 40-pin GPIO header.
- 4 USB 2.0 ports.
- 2.4 GHz and 5 GHz dual-band wireless LAN.
- Bluetooth 4.2.
- Ethernet.
- Full-size HDMI.
- Camera port.
- Micro SD for loading operating system and storing data.

Raspberry Pi 3 Model B+ uses Broadcom chip which has dramatic advancement of higher rated frequency than the other models. The ARM cores embedded inside this model are capable of running at up to 1.4GHz, making it about 17% faster than the original Raspberry Pi 3. In addition, The BCM2837B0 chip is packaged with a heat spreader for better thermals. This allows higher clock frequencies or running at lower voltages to reduce power consumption.

3.2.2 Intel Movidius Neural Compute Stick 2

3.2.2.1 What is Movidius NCS?

Low-power consumption is indispensable for autonomous applications and IoT devices and appliances. In order to develop deep learning inference applications at the edge, Intel Corporation has introduced an energy-efficient and low-cost tool with the name Movidius Neural Compute Stick.

Figure 3.5: *Movidius Neural Compute Stick 2 [21]*

At the core of this strategy is the Myriad Vision Processing Unit (VPU) which is an AI optimized chip for accelerating computational power of machine vision based on convolutional neural network (CNN). Myriad VPUs have dedicated architecture for high-quality image processing, computer vision and deep neural networks, making them suitable to drive the demands of vision-centric tasks in modern smart device. Intel has extended at the same time Myriad System-on-Chip (SoC) board and Movidius Neural Compute Stick (Movidius NCS is a USB-thumb-drive-sized deep learning machine and can be easily attached to the edge devices such as Raspberry Pi.

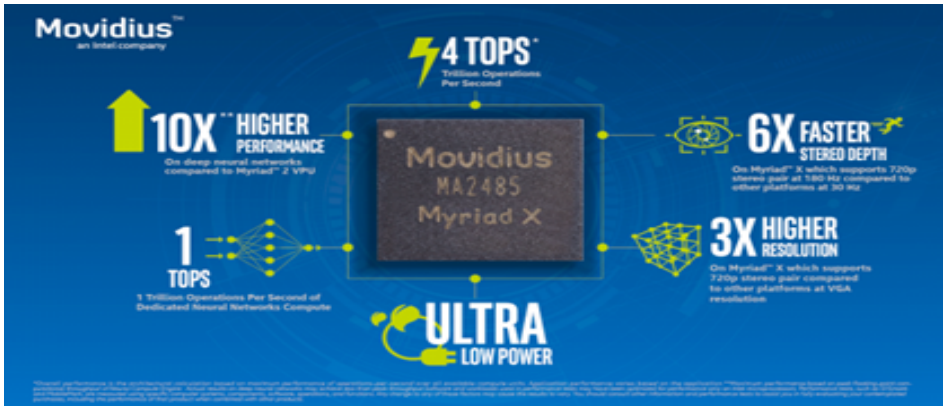


Figure 3.6: *Hardware Based Acceleration for Deep Neural Networks [22]*

An important note about this product is that Movidius NCS is not a GPU because it can only be used for prediction or inference, not training. NCS would be classified as a coprocessor and get one purpose: running (forward-pass) neural network calculations. In this range of our thesis, NCS is being used for object detection. In addition, this device is meant to be used on single board computers like Raspberry Pi, so that the power draw has to be minimal, making it inappropriate for actually training a network.

3.2.2.2 Why choose Intel NCS version 2?

Currently, there are two versions of NCS devices available in the market. Movidius NCS 2 is the latest version was released in late 2018. Its purpose is to improve computer vision and AI inference applications, make it faster and easier to develop products for the network edge. There are several features that make NCS 2 become great choice for inference accelerator.

Most essential point should be emphasized is Myriad X is inside NCS2. The Myriad X VPU has 16 128-bit Streaming Hybrid Architecture Vector Engine SHAVE) processor cores (up from 12 cores inside Myriad 2 of NCS version 1). At the same time, Myriad X is the first VPU to feature Intel's Neural Compute Engine, a dedicated hardware accelerator with native FP16 support to reduce the time and energy spent on computing at little or no cost in accuracy.

The second characteristic about this device is USB 3.0/3.1 standard has been supported. This is an improvement over the original NCS's USB 2.0. Thus, it is explicable to claim that Intel's NCS 2 is 8 times faster than its predecessor.

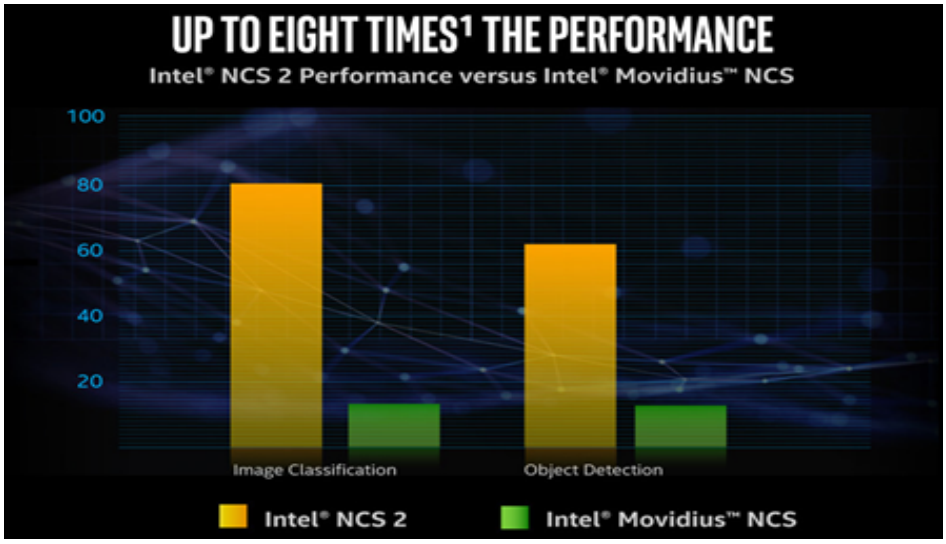


Figure 3.7: *Intel's NCS 2 is 8 times faster than its predecessor [21]*

Last but not least, Intel has added NCS 2 support in OpenVINO Toolkit, a software platform for optimizing deep learning models. This new toolkit is compatible for other Intel's hardware platforms and provides support for many kinds of deep learning frameworks.

NCSDK Supported Frameworks	OpenVINO™ toolkit Supported Frameworks
Caffe	Caffe
TensorFlow	TensorFlow
N/A	MxNet
N/A	Kaldi
N/A	ONNX

Figure 3.8: *Supported deep learning frameworks of old (left) and new (right) toolkit [21]*

3.2.3 STM32F407-DISCOVERY Board

Figure 3.9: *STM32F4 Discovery kit [20]*

The STM32F4 Discovery kit allows users to easily develop applications with the STM32F407VG high-performance microcontroller. It includes:

- TM32F407VG microcontroller featuring 32-bit ARM Cortex-M4 with FPU core, 1 Mbyte Flash memory, 192 Kbyte RAM.
- On-board ST-LINK/V2.
- SB ST-LINK with three different interfaces: debug port, virtual COM port, mass storage.
- Board power supply.

- 2 push buttons: user and reset.
- SB OTG with micro-AB connector.
- Eight LEDS: Power on, USB Communication, 4 user LEDs, 2 USB OTG LEDs.

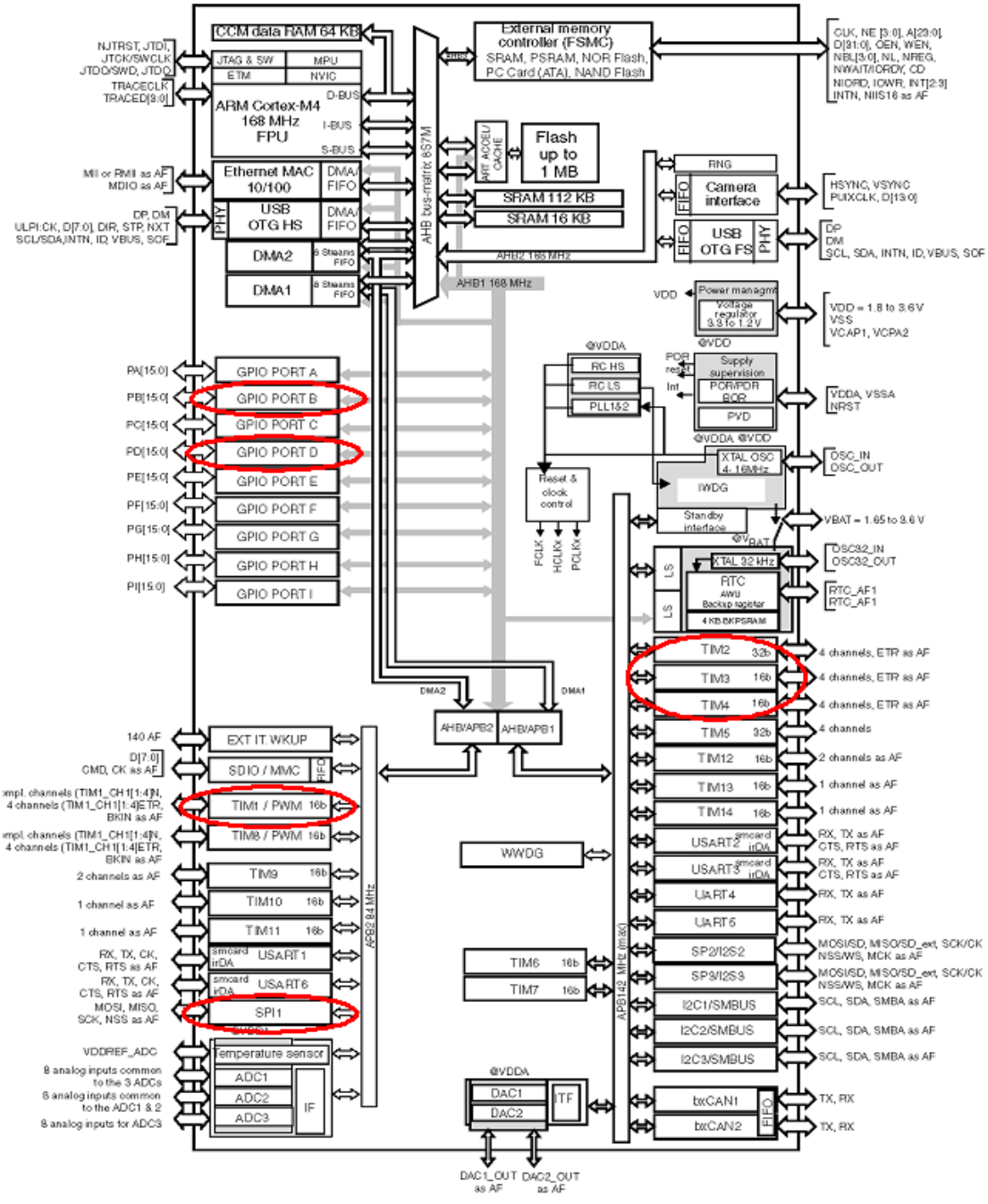


Figure 3.10: *STM32F4 block diagram [?]*

Purpose of applied function block in this project (The circled regions):

- *TIM1/PWM:* provides pwm generation. We use channel 1 and channel 3 to control the right and left motor respectively.
- *SPI1:* provides SPI communication interface. We use this 4-wire function to transfer data to Rasperry Pi.
- *TIM2:* 10us timer for trigger pulse period of ultrasonic sensors.
- *TIM3:* 100ms timer for cycle of SPI communication, motors controller and ultrasonic sensor reading.
- *TIM4:* 1us timer for echo pulse counter from ultrasonic sensors.
- *GPIO PORT D:* we set PD1, PD3, PD4, PD5 for output trigger pulse transmission to ultrasonic sensors.
- *GPIO PORT B:* we set PB1, PB3, PB4, PB5 for external interupt to receive echo pulse from ultrasonic sensors.

3.2.4 Raspberry Pi Camera

Figure 3.11: *Raspberry Pi Camera Module V1 5MP [19]*

The Raspberry Pi Camera Board plugs directly into the CSI connector on the Raspberry Pi. It's able to deliver a crystal clear 5MP resolution image, or 1080p HD video recording at 30fps.

3.2.5 Driver L298N

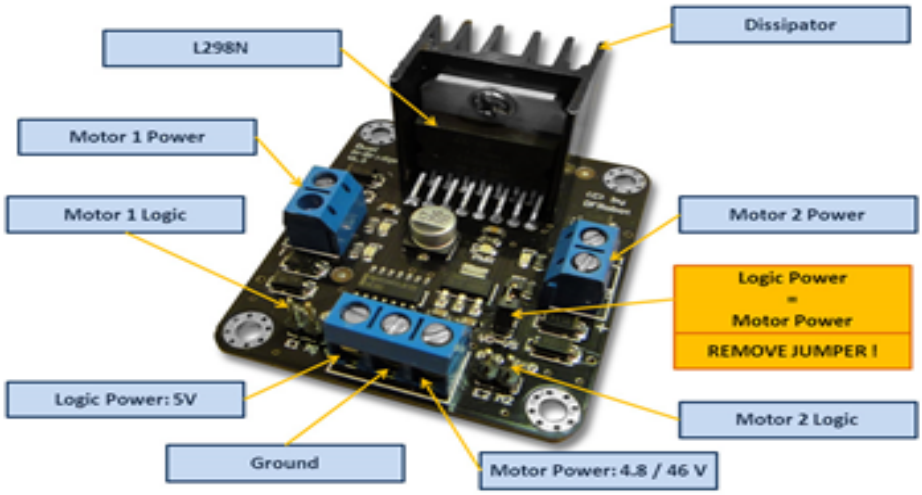


Figure 3.12: *Driver L298N*

L298N is a dual H-Bridge motor driver which controls speed and direction of 2 motors at the same time simply by changing the input voltage. The most common way of doing this is by using PWM signal.

L298N uses 5-35V with a peak current up to 2A. If the motor voltage is greater than 12V, we must disconnect the jumper or else it will damage the onboard 5V regulator. However, there is a voltage drop of approximately 2V between the supply power and motor voltage.

3.2.6 Power converter LM2596

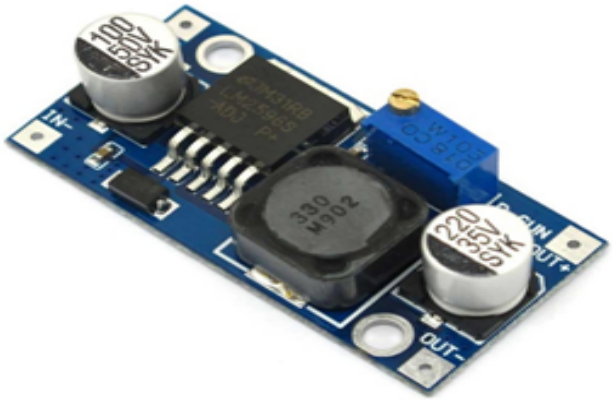


Figure 3.13: *LM2596 DC-DC adjustable step-down buck power converter*

- Input voltage: 3-30V
- Output voltage: 1.5-30V (adjustable)
- Max current: 3A
- Efficiency: 92%
- Power: 15W
- Size: 45*20*14mm

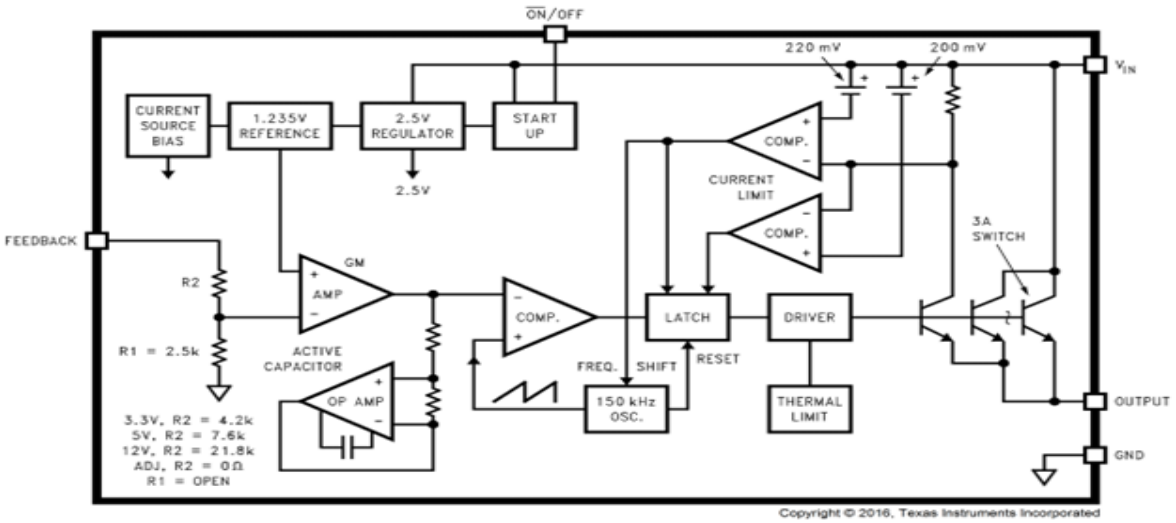


Figure 3.14: *LM2596 functional block diagram [15]*

3.2.7 Ultrasonic sensors HC-SR05

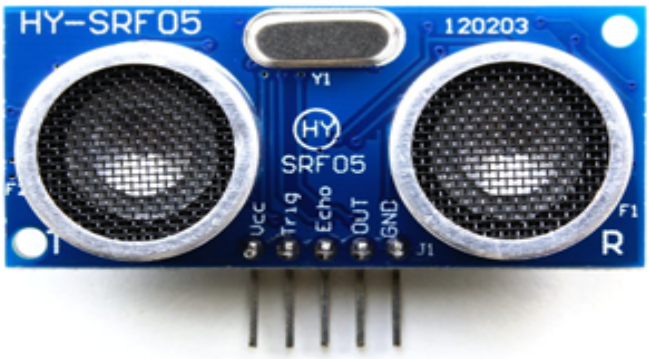


Figure 3.15: *Ultrasonic sensors HC-SR05*

The HC-SR05 ultrasonic distance sensor provides an easy way to measure distances up to 4.5m.

- Supply voltage: 4.5-5.5VDC.
- Supply current: 10-40mA.
- Sound frequency: 40KHz.
- Trigger pin: 10us digital pulse.
- Measurement range: 2-450cm.
- Measurement resolution: 0.3cm.
- Measurement angle: 15° .

The HC-SR05 ultrasonic sensor are being used in many autonomous and robotic applications. The strength of this device is its cheap price, light weight, low power consumption and not so complicated computation. It is also used under water which is low-visible condition. The three well-known applications of this sensor are:

- **Obstacle avoidance:** The first received reflective wave help to calculate the distance to the nearest object. The robot embedded this sensor will know the best direction for continuous movement. This is also one of our essential part in this project.

- **Sonar mapping:** Multibeam measures the depth of the sea floor. It measures the length of time it takes for the sound to travel from the boat to the sea floor and back as an echo. Scientists use the data of time measurements to figure out how deep the water is.

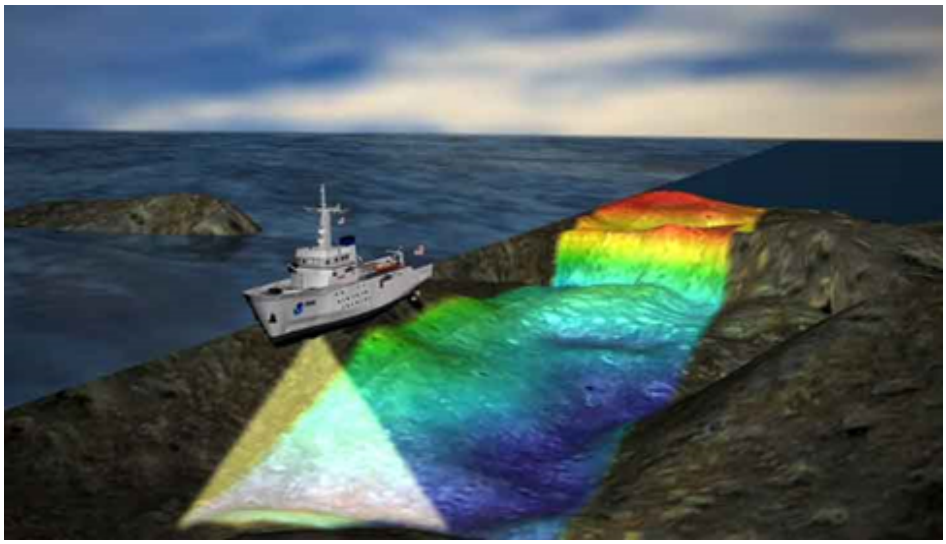


Figure 3.16: *Sonar mapping application [14]*

- **Object recognition:** Multibeam of reflection will be processed for physical structure of different objects. If the classification of these objects succeeds, the results will be useful in robotic chasing function.

3.2.8 Motors

Figure 3.17: *Chihai Motor DC Magnetic Holzer Encoder Gear Motor*

DC gear motor with:

- Voltage: 12V.
- No-load speed: 140rpm.
- No-load current: ¡100mA.
- Torque: up to 8.5kg.cm.
- Motor size: 25*25*73mm.
- Encoder: 495ppr with 3-5V supply voltage.

In our project:

- We uses a 65mm diameter wheel, so the maximum motors force is $\frac{8.5*10^{-2}}{\frac{65}{2}*10^{-3}} * 2 \approx 5.2N$.
- Consider the typical rolling resistance coefficient (RRC) about 0.01, the maximum model weight accepted is $\frac{5.2}{0.01*9.81} \approx 53kg$

3.2.9 Portable power bank

Figure 3.18: *Portable power bank*

Portable Power Bank is a common battery charger for mobile phones, Bluetooth headphones, portable speakers or GPS devices. Lithium-Ion and Lithium-Polymer

batteries are the most used cell types in power bank. This is a stable and simple way to charge devices on the go.

We use a 10000mAh portable power bank for powering Raspberry Pi 3 B+ and STM32F4 simultaneously through 5V-2A USB chargers.

3.2.10 LiPo battery 3S

Figure 3.19: *XPower 11.1V 4200mah 30C Battery*

LiPo batteries, short for **Lithium Polymer** battery, are a type of rechargeable battery which are widely used for planes, helicopters, and multi-rotor/drone. The advantages of this kind of battery are that they have more than four times the energy of density of nickel cadmium or nickel metal hydride batteries. They are also lightweight, pliable and can be made to almost any size or shape. Furthermore, LiPo battery cells are more resistant to physical trauma than the others that make them suitable for flying objects or autonomous applications.

We use a LiPo battery to power the two motors and four ultrasonic sensors (after passing through the power converter). Its parameters are:

- Voltage: 11.1V.
- Capacity: 4200mAh.
- Discharge Rate: 30C.
- Plug: XT60.

3.2.11 LiPo Battery Voltage Tester Low Voltage Buzzer Alarm

Figure 3.20: *LiPo battery checker*

This additional module is used for two functions:

- **Protecting LiPo battery cells:** LiPo cells must not be discharged below 3.3V or it will permanently be damaged. Many chargers do not even accept to charge LiPo battery below 3V per cell. Fortunately, this electrical board will help to announce when any cell's voltage is lower than the threshold voltage (usually 3.3V) by buzzing with loud noise.
- **Preventing voltage drop:** Since we are using a LiPo battery for controlling the two motors and four ultrasonic sensors, we need a low voltage buzzer alarm to ensure the battery is enough for maintaining suitcase performance. To specify, this device will buzzer with the LED light whenever the voltage is below the set value, which is 3.3V.

Chapter 4

SOFTWARE DESIGN

4.1 Overall diagram

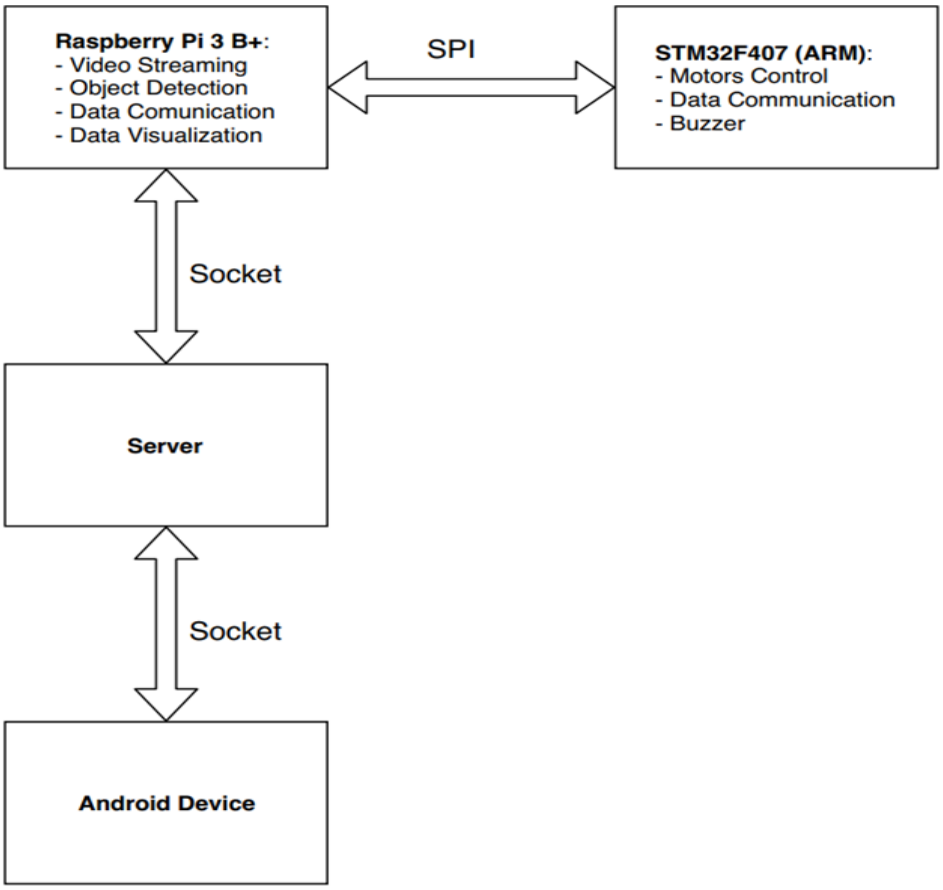


Figure 4.1: *Overall diagram for software design.*

4.2 Raspberry Pi

4.2.1 Training deep learning neural network model for object detection

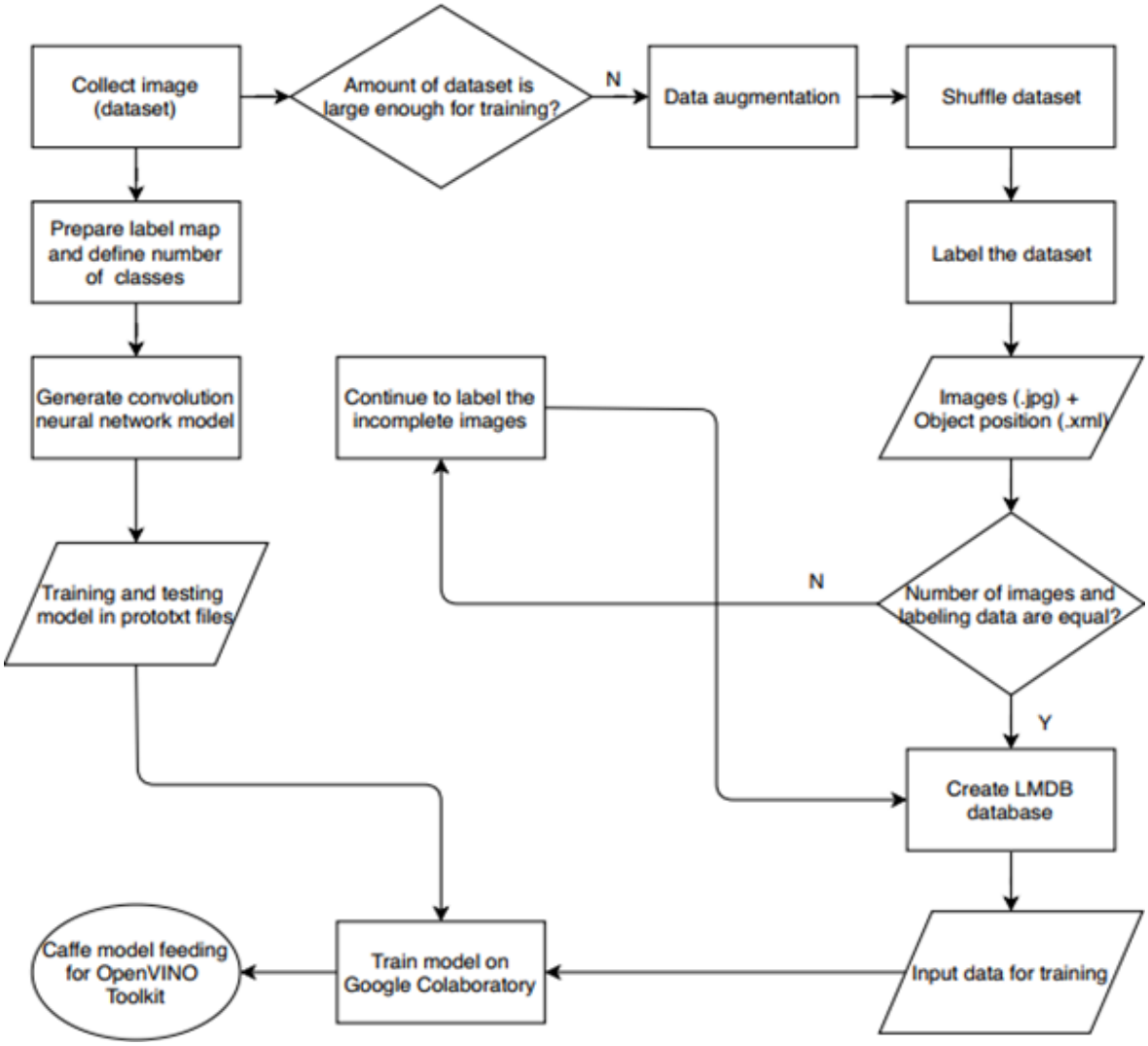


Figure 4.2: *Training CNN model.*

4.2.1.1 Data augmentation

A real obstacle in almost conditions that people have to deal with in training a deep learning model is the shortage of dataset. Especially for the case of customized dataset (such as school logo, personal objects, etc.) that is unique and hard to find the supplier. A very good method helps us to overcome this problem is image data augmentation.

Data augmentation is a way of creating new data based on the available source with different orientations. Working with limited data has its own challenges but using this technique can have positive results including enlarging dataset and preventing current model from being overfitting.

Data augmentation involves creating transformed versions of source images. There are several augmentation techniques applied to transform a single image such as flip, blur adding, rotation, image's characteristic adjustment, etc. These following figures are good examples of this ideas.

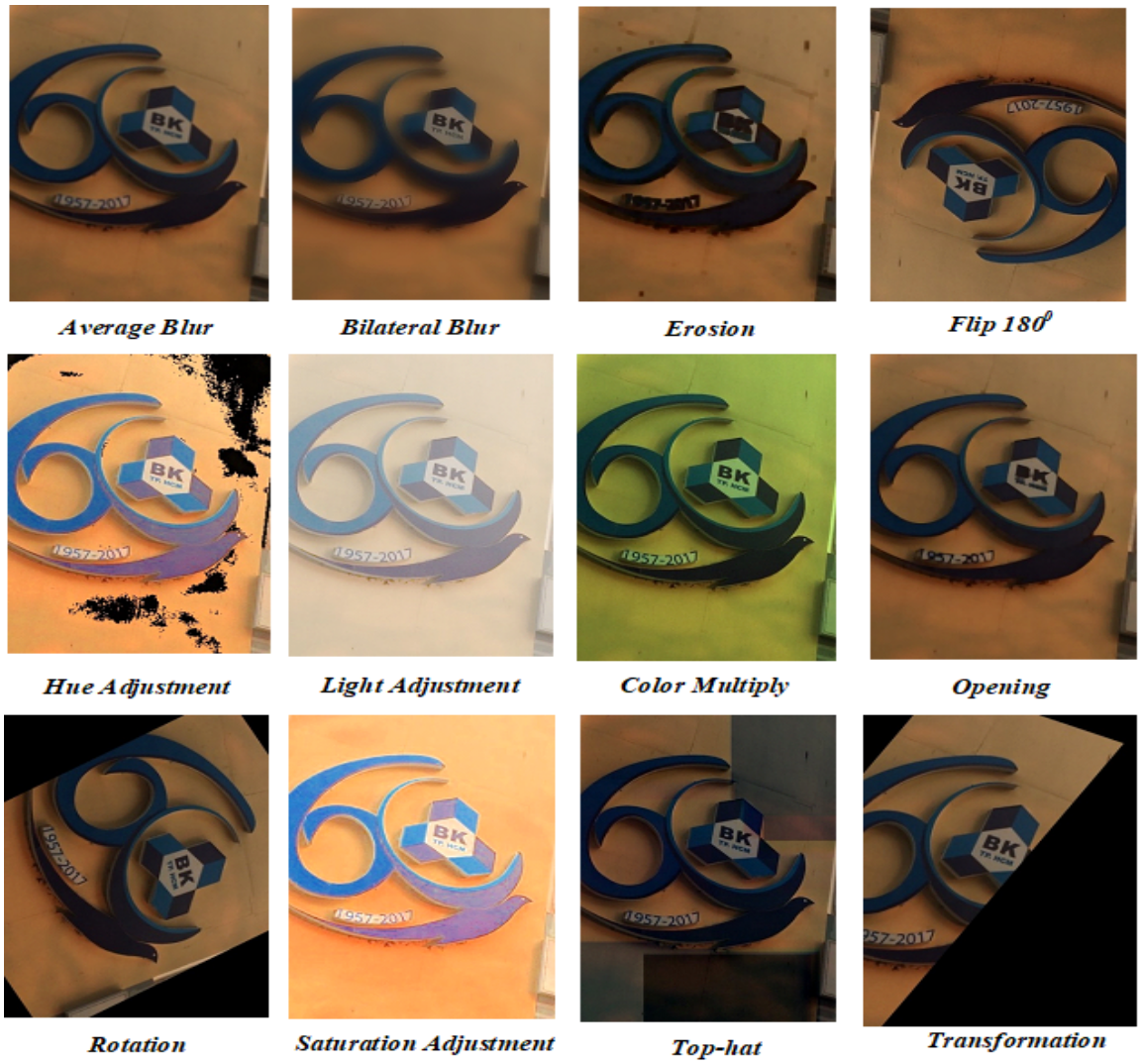


Figure 4.3: *Some examples of image data augmentation.*

In this step of training, these effects are easy to be created with the support of OpenCV2 library that is well-knowed as a very helpful tool for conventional image

processing.

4.2.1.2 Label the dataset

In this step, we will detect the desired objects and save the positions of them (a rectangle bounding box) before training. These annotations are saved as XML files in default PASCAL VOC format (this format used by ImageNet).

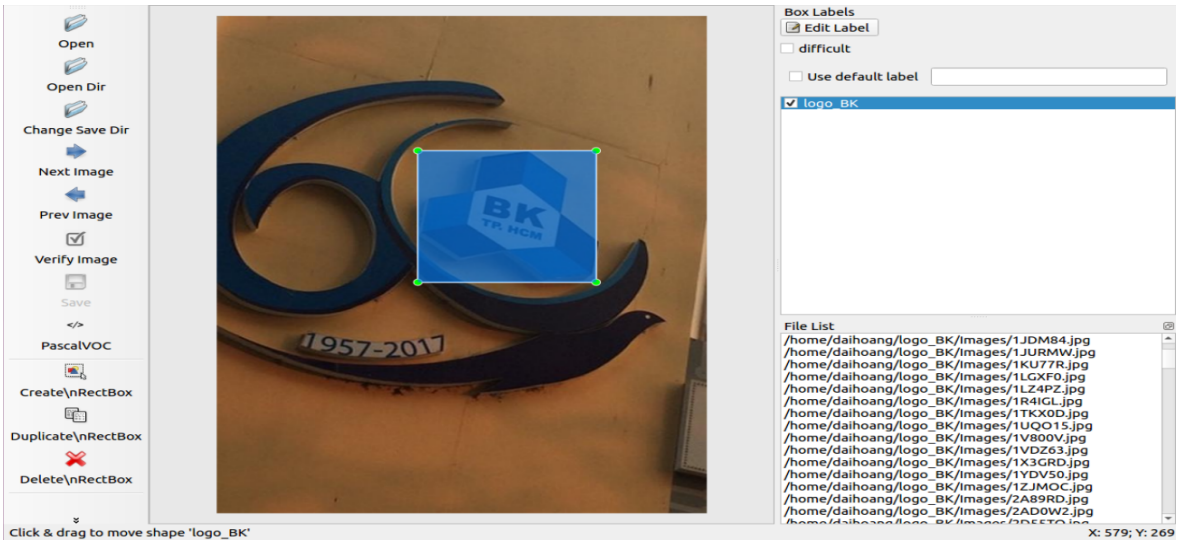


Figure 4.4: *Labeling dataset.*

Figure 4.5: *Rectangle bounding box detected (left) and its information contained in an XML file (right).*

4.2.1.3 Prepare label map and define number of classes

Label mapping file helps the model to know how many desired classes for detection. In Caffe framework, this step has to be done manually before training.

For instance, in this range of thesis, there are only one target for tracking – "logo_BK", so the total number of classes must be assigned here is two because "background" is also an important class in object detection. An example of valid label mapping declaration is shown below.

```
1  item {
2    name: "none_of_the_above"
3    label: 0
4    display_name: "background"
5  }
6  item {
7    name: "logo_BK"
8    label: 1
9    display_name: "logo_BK"
10 }
```

Figure 4.6: *Classes of Object detection are declared in label mapping file.*

4.2.1.4 Create LMDB database

After finishing labeling and getting annotation of all labeled images, we will divide our dataset into training and validation set. Normally about 80% of entire dataset used for training and the remaining for validation. This task can be automatically executed with a simple code like the following. In line number 8, we can change the amount of training set (maybe 70% or 90%) but it has to be greater than 50%.

```python
def createTrainvalTxt(baseDirDataSet):
    bufferTrain = ''
    bufferTest = ''
    baseDir = baseDirDataSet+'/Images'
    images = os.listdir(baseDir)
    numOfTrainingSet = int(len(images)*0.8)     #80% of images will be used for training and the remaining images used for testing
    loop = 0
    for filename in images:
        loop += 1
        filenameOnly, file_extension = os.path.splitext(filename)
        # print (file_extension)
        s = 'Images/'+filenameOnly+'.jpg'+' '+'Labels/'+filenameOnly+'.xml\n'
        print (repr(s))
        img_file, anno = s.strip("\n").split(" ")
        print(repr(img_file), repr(anno))
        if loop <= numOfTrainingSet:
            bufferTrain += s
        else:
            bufferTest += s
    with open(baseDirDataSet+'/Structure/trainval.txt', 'w') as file:
        file.write(bufferTrain)
    with open(baseDirDataSet+'/Structure/test.txt', 'w') as file:
        file.write(bufferTest)

    print('Done')
```

Figure 4.7: *A sample code for splitting training dataset.*

Moving forward onto the business of how creating lmdb database, Caffe framework gives the user a python executed file (create_annoset.py) required some input parameters. This exactly makes everything become easier in this stage. The subsequent figure shows that the process of creating lmdb database has succeeded.

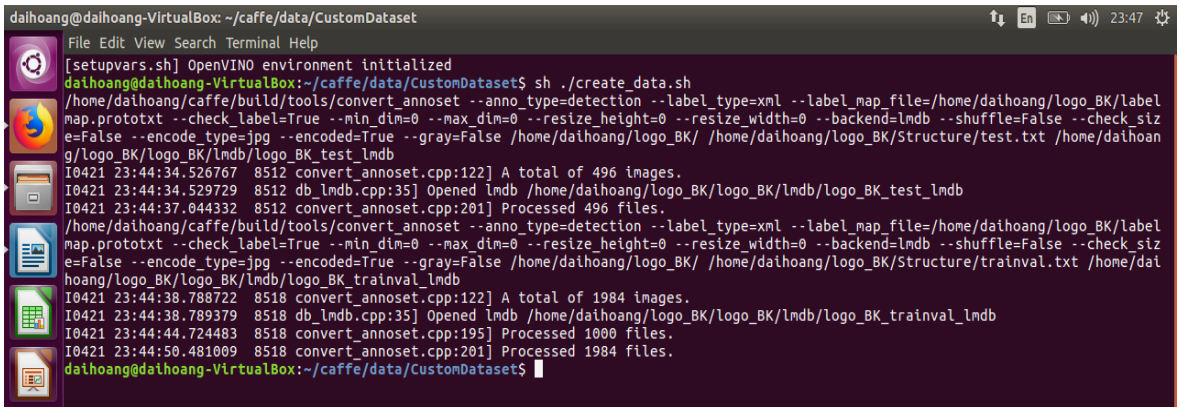


Figure 4.8: *Success on creating lmdb database.*

4.2.1.5 Train model on Google Colaboratory

- **Solver definition:** Another important file to define the entire training model is solver definition. This file is the way we can build my own model, training the parameters on training data. The solver we used in this model will be shown below.

```
train_net: "example/MobileNetSSD_train.prototxt"    # path to the network definition
test_net: "example/MobileNetSSD_test.prototxt"
test_iter: 673  # how many mini-batches to test in each validation phase
test_interval: 10000    # how often do we call the test phase
base_lr: 0.0005 # base learning rate
display: 10 # how often do we print training loss
max_iter: 120000
lr_policy: "multistep"  # step means to decrease lr after a number of iterations
gamma: 0.5  # ratio of decrement in each step
weight_decay: 0.00005   # regularization
snapshot: 1000  # taking snapshot is like saving your progress in a game
snapshot_prefix: "snapshot/mobilenet"
solver_mode: GPU
debug_info: false
snapshot_after_train: true
test_initialization: false
average_loss: 10
stepvalue: 20000
stepvalue: 40000
iter_size: 1
type: "RMSProp"
eval_type: "detection"
ap_version: "11point"
```

Figure 4.9: *An example of solver definition.*

- **Train model on Google Colaboratoy**
  - After completely defining training model and solver as well as dataset stored in LMDB database, the training phase will be took place on Google Colaboratory to make use of computational power of GPU Tesla K40 supported by Google Corporation.
  - This training session may take a few hours to get the desired loss. An example of result is shown below.

```
I0602 13:08:53.179177 42613 solver.cpp:243] Iteration 0, loss = 11.7218
I0602 13:08:53.179327 42613 solver.cpp:259]     Train net output #0: mbox_loss = 11.7218 (* 1 = 11.7218 loss)
I0602 13:08:53.179392 42613 sgd_solver.cpp:138] Iteration 0, lr = 0.0005
I0602 13:09:29.314649 42613 solver.cpp:243] Iteration 10, loss = 5.27145
I0602 13:09:29.314882 42613 solver.cpp:259]     Train net output #0: mbox_loss = 3.93367 (* 1 = 3.93367 loss)
I0602 13:09:29.314898 42613 sgd_solver.cpp:138] Iteration 10, lr = 0.0005
I0602 13:10:05.361060 42613 solver.cpp:243] Iteration 20, loss = 3.5302
I0602 13:10:05.361208 42613 solver.cpp:259]     Train net output #0: mbox_loss = 3.50857 (* 1 = 3.50857 loss)
I0602 13:10:05.361227 42613 sgd_solver.cpp:138] Iteration 20, lr = 0.0005
I0602 13:10:41.051959 42613 solver.cpp:243] Iteration 30, loss = 3.16511
I0602 13:10:41.052213 42613 solver.cpp:259]     Train net output #0: mbox_loss = 2.92797 (* 1 = 2.92797 loss)
I0602 13:10:41.052243 42613 sgd_solver.cpp:138] Iteration 30, lr = 0.0005
I0602 13:11:17.240660 42613 solver.cpp:243] Iteration 40, loss = 2.82758
I0602 13:11:17.240907 42613 solver.cpp:259]     Train net output #0: mbox_loss = 2.68941 (* 1 = 2.68941 loss)
I0602 13:11:17.240947 42613 sgd_solver.cpp:138] Iteration 40, lr = 0.0005
I0602 13:11:53.484347 42613 solver.cpp:243] Iteration 50, loss = 2.55773
I0602 13:11:53.484608 42613 solver.cpp:259]     Train net output #0: mbox_loss = 2.29136 (* 1 = 2.29136 loss)
I0602 13:11:53.484647 42613 sgd_solver.cpp:138] Iteration 50, lr = 0.0005
I0602 13:12:29.192392 42613 solver.cpp:243] Iteration 60, loss = 2.35423
I0602 13:12:29.192544 42613 solver.cpp:259]     Train net output #0: mbox_loss = 2.22172 (* 1 = 2.22172 loss)
I0602 13:12:29.192559 42613 sgd_solver.cpp:138] Iteration 60, lr = 0.0005
I0602 13:13:04.842308 42613 solver.cpp:243] Iteration 70, loss = 2.35669
I0602 13:13:04.842530 42613 solver.cpp:259]     Train net output #0: mbox_loss = 2.23136 (* 1 = 2.23136 loss)
I0602 13:13:04.842547 42613 sgd_solver.cpp:138] Iteration 70, lr = 0.0005
I0602 13:13:40.590780 42613 solver.cpp:243] Iteration 80, loss = 2.17317
I0602 13:13:40.590957 42613 solver.cpp:259]     Train net output #0: mbox_loss = 2.05571 (* 1 = 2.05571 loss)
I0602 13:13:40.590975 42613 sgd_solver.cpp:138] Iteration 80, lr = 0.0005
I0602 13:14:16.732743 42613 solver.cpp:243] Iteration 90, loss = 2.02298
I0602 13:14:16.732847 42613 solver.cpp:259]     Train net output #0: mbox_loss = 1.66058 (* 1 = 1.66058 loss)
I0602 13:14:16.732861 42613 sgd_solver.cpp:138] Iteration 90, lr = 0.0005
I0602 13:14:52.307106 42613 solver.cpp:243] Iteration 100, loss = 1.90731
I0602 13:14:52.307225 42613 solver.cpp:259]     Train net output #0: mbox_loss = 1.78205 (* 1 = 1.78205 loss)
I0602 13:14:52.307240 42613 sgd_solver.cpp:138] Iteration 100, lr = 0.0005
I0602 13:15:28.228690 42613 solver.cpp:243] Iteration 110, loss = 1.74692
I0602 13:15:28.228863 42613 solver.cpp:259]     Train net output #0: mbox_loss = 2.15239 (* 1 = 2.15239 loss)
I0602 13:15:28.228888 42613 sgd_solver.cpp:138] Iteration 110, lr = 0.0005
I0602 13:16:03.852288 42613 solver.cpp:243] Iteration 120, loss = 1.77511
I0602 13:16:03.852409 42613 solver.cpp:259]     Train net output #0: mbox_loss = 1.62131 (* 1 = 1.62131 loss)
I0602 13:16:03.852422 42613 sgd_solver.cpp:138] Iteration 120, lr = 0.0005
I0602 13:16:40.339390 42613 solver.cpp:243] Iteration 130, loss = 1.52384
I0602 13:16:40.339552 42613 solver.cpp:259]     Train net output #0: mbox_loss = 1.65469 (* 1 = 1.65469 loss)
I0602 13:16:40.339570 42613 sgd_solver.cpp:138] Iteration 130, lr = 0.0005
I0602 13:17:16.274834 42613 solver.cpp:243] Iteration 140, loss = 1.60283
I0602 13:17:16.274996 42613 solver.cpp:259]     Train net output #0: mbox_loss = 1.81225 (* 1 = 1.81225 loss)
```

Figure 4.10: *Training session on Google Colaboratory.*

  - This training session will provide files .caffemodel. This file with model definition will be used as inputs of model optimizer of OpenVINO to create .xml and .bin before inference engine.

- **Model Optimizer**
  - Model optimizer is a cross-platform command-line tool that facilitates the transition between the training and deployment environment, performs static model analysis, and adjusts deep learning models for optimal execution on end-point target devices.

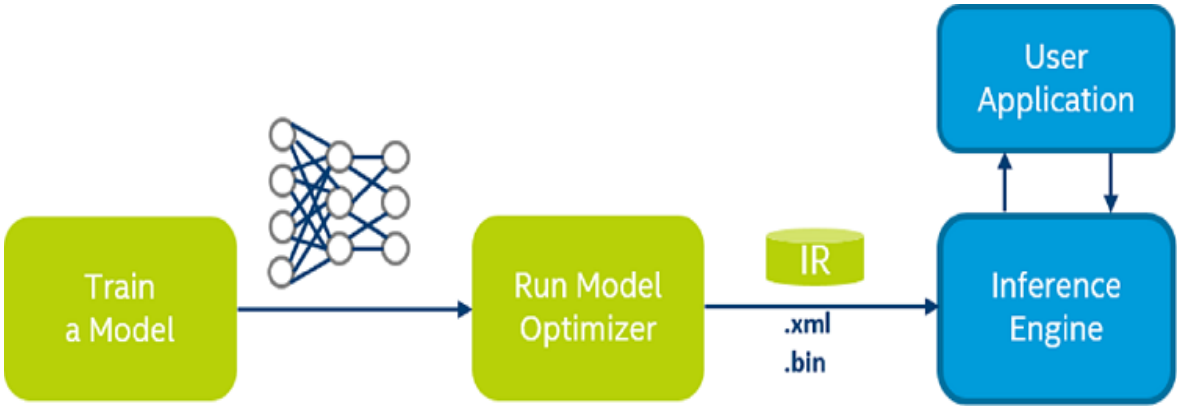


Figure 4.11: *Typical workflow for deploying a trained deep learning model [23].*

- Model Optimizer produces an Intermediate Representation (IR) of the network, which can be read, loaded, and inferred with the Inference Engine. The IR is the pair of files that describing the model:
  - .xml: Describe the network topology.
  - .bin: Contains the weights and biases binary data.

4.2.2 Object detection on Raspberry Pi and Intel Movidius NCS2

The entire process of this important task is summarized by the below diagram:

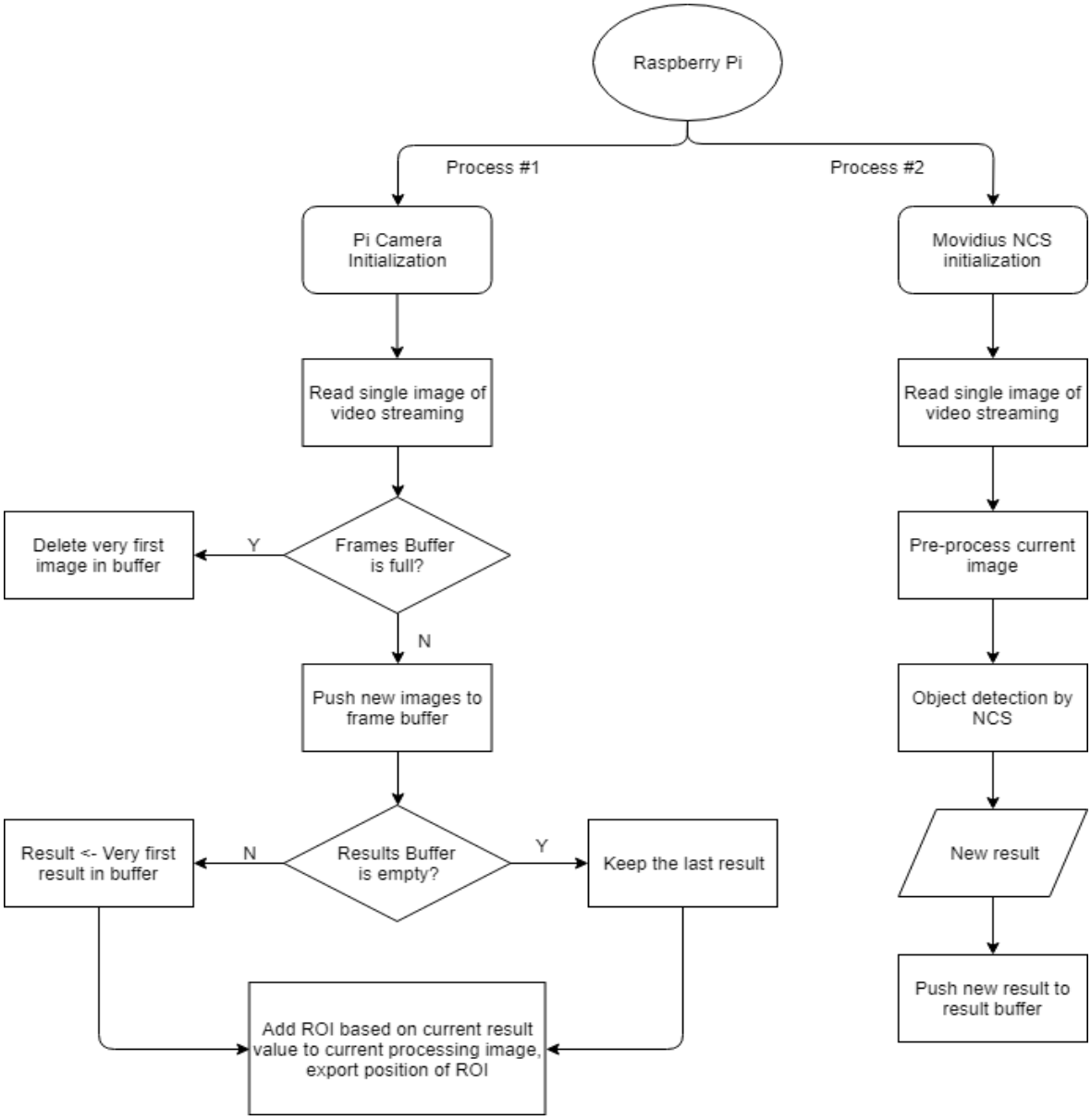


Figure 4.12: *Object detection system.*

As mentioned above, to prevent data from being corrupted and enhance the speed of data processing. We apply multiprocessing or parallel programming to make use of computational power of Quad-core CPU inside Raspberry Pi. Data sharing technique is also responsible for image processing. In this circumstance, frameBuffer and result are multiprocessing Queue used for containing and passing images between two parallel operations.

To reach the requirement of improving FPS to approach real-time tracking, Intel Movidius NCS has been used in this project. Most important task is to feed this

computing stick the IR files at the initialization and the image data for object detection at real-time loop.

The following code will show how to initialize Movidius before using. Its content are declaring directory of IR files (.xml and .bin), container of detection output, device used for inferring (MYRIAD of NCS2) and number of deploying devices as well.

```
class NcsWorker(object):
    def __init__(self, devid, frameBuffer, results, camera_width, camera_height, number_of_ncs):
        self.devid = devid
        self.frameBuffer = frameBuffer
        self.model_xml = "./mo_caffe/no_bn.xml"
        self.model_bin = "./mo_caffe/no_bn.bin"
        self.camera_width = camera_width
        self.camera_height = camera_height
        self.num_requests = 4
        self.inferred_request = [0] * self.num_requests
        self.heap_request = []
        self.inferred_cnt = 0
        self.plugin = IEPlugin(device = "MYRIAD")
        self.net = IENetwork(model = self.model_xml, weights = self.model_bin)
        self.input_blob = next(iter(self.net.inputs))
        self.exec_net = self.plugin.load(network = self.net, num_requests = self.num_requests)
        self.results = results
        self.number_of_ncs = number_of_ncs
```

Figure 4.13: *Initialization code of Movidius NCS2.*

4.2.3 Tracking result transfer (Third Process on Raspberry)

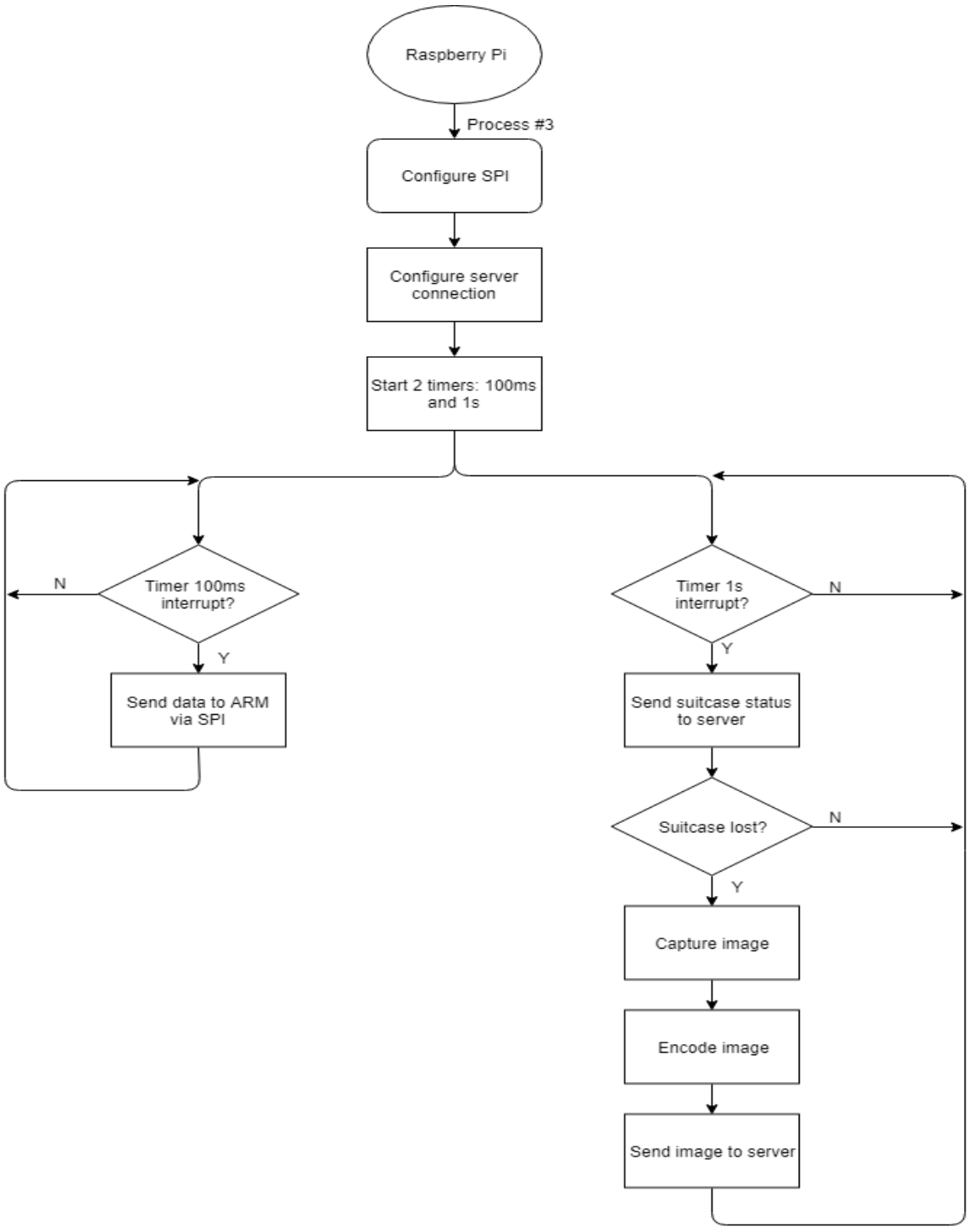


Figure 4.14: *Initialization of Movidius NCS2.*

4.2.3.1 Tracking result transfer to ARM

- Raspberry Pi is the master-side of the SPI connection. It sends the owner-tracking result to ARM via SPI every 100ms.
- Since Raspberry Pi plays the role of a master, it needs to define SPI clock signal. This is the SPI configuration in Raspberry:

```
#SPI config
spi = spidev.SpiDev()
spi.open(0,0)          #Port 0, device 0 (cs0)
spi.max_speed_hz = 16000000
```

Figure 4.15: *SPI configuration.*

- Raspberry buffer to ARM:

isTracking	error_Position [0]	error_Position [1]	error_Distance [0]	error_Distance [1]	0x0D	0x0A

Figure 4.16: *Data buffer from Raspberry to ARM.*

- *isTracking* refers to the status whether the suitcase if following the owner or not. If *isTracking* is 1, the suitcase is following. If it is 0, the suitcase is lost.
- Since the *error_Position* and *error_Distance* are 16-bit signed integer, we convert each to 2 bytes using little endian before sending.

```
tmp_P = eP.to_bytes(2, byteorder = "little", signed = True)
eP_L = tmp_P[0]
eP_H  = tmp_P[1]
tmp_D = eD.to_bytes(2, byteorder = "little", signed = True)
eD_L = tmp_D[0]
eD_H = tmp_D[1]
#Send data 7 bytes
resp = spi.xfer2([isTrack, eP_L, eP_H, eD_L, eD_H, 0x0D, 0x0A])
```

Figure 4.17: *Data transfer to ARM.*

4.2.3.2 Tracking result transfer to Server

- Raspberry Pi represents the suitcase to send tracking result to the server every second.

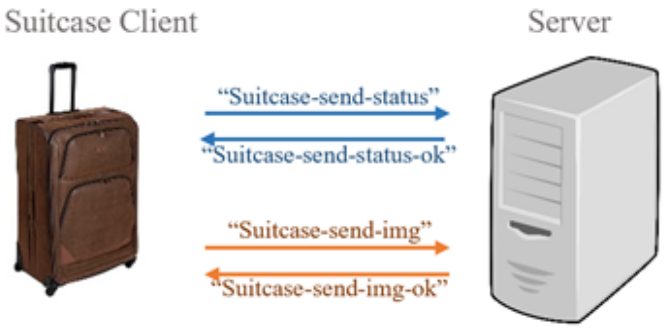


Figure 4.18: *Raspberry Pi (Suitcase) requests to Server.*

- We use the Socket.IO library, which will be introduced in *4.4.2*, to connect to server.
- The message "suitcase-send-status" includes:
  - *isTracking*: refers to the *isTracking* variable which is the tracking status of the suitcase.
  - *lostTime*: refers to the lost period, which is counted in this timer. If the suitcase is tracking, this variable will be 0.

```
mydict = {"isTracking": isTracking, "lostTime": timerLost}
sio.emit('suitcase-send-status', mydict)
```

Figure 4.19: *suitcase-send-status Message.*

- The message "suitcase-send-img" is only sent when the suitcase is lost. It includes:
  - *Image*: refers to the encoded image using base64. Base64 is a way to represent binary data in an ASCII string format.
  - *CapTime*: refers to the time when the image is captured.

```
if (isTracking == 0):
    CaptureImage()
    with open("/home/pi/saved_images/image.jpg","rb") as file:
        jpg_as_text = base64.b64encode(file.read())
    capturedTime = datetime.datetime.now().strftime("%Y-%m-%d %H-%M-%S")
    mydict_img = {"Image": "" + jpg_as_text.decode("utf-8"), "CapTime":capturedTime}
    sio.emit('suitcase-send-img', mydict_img)
```

Figure 4.20: *suitcase-send-img Message.*

4.3 ARM

ARM STM32F4 is responsible for communicating with Raspberry Pi to get suitcase data and controlling the left and right motors of the suitcase. The purpose of using ARM microcontroller is to minimize the calculation, boost efficiency of image processing in Raspberry Pi as well as control the owner-tracking and obstacle avoidance at the same time.

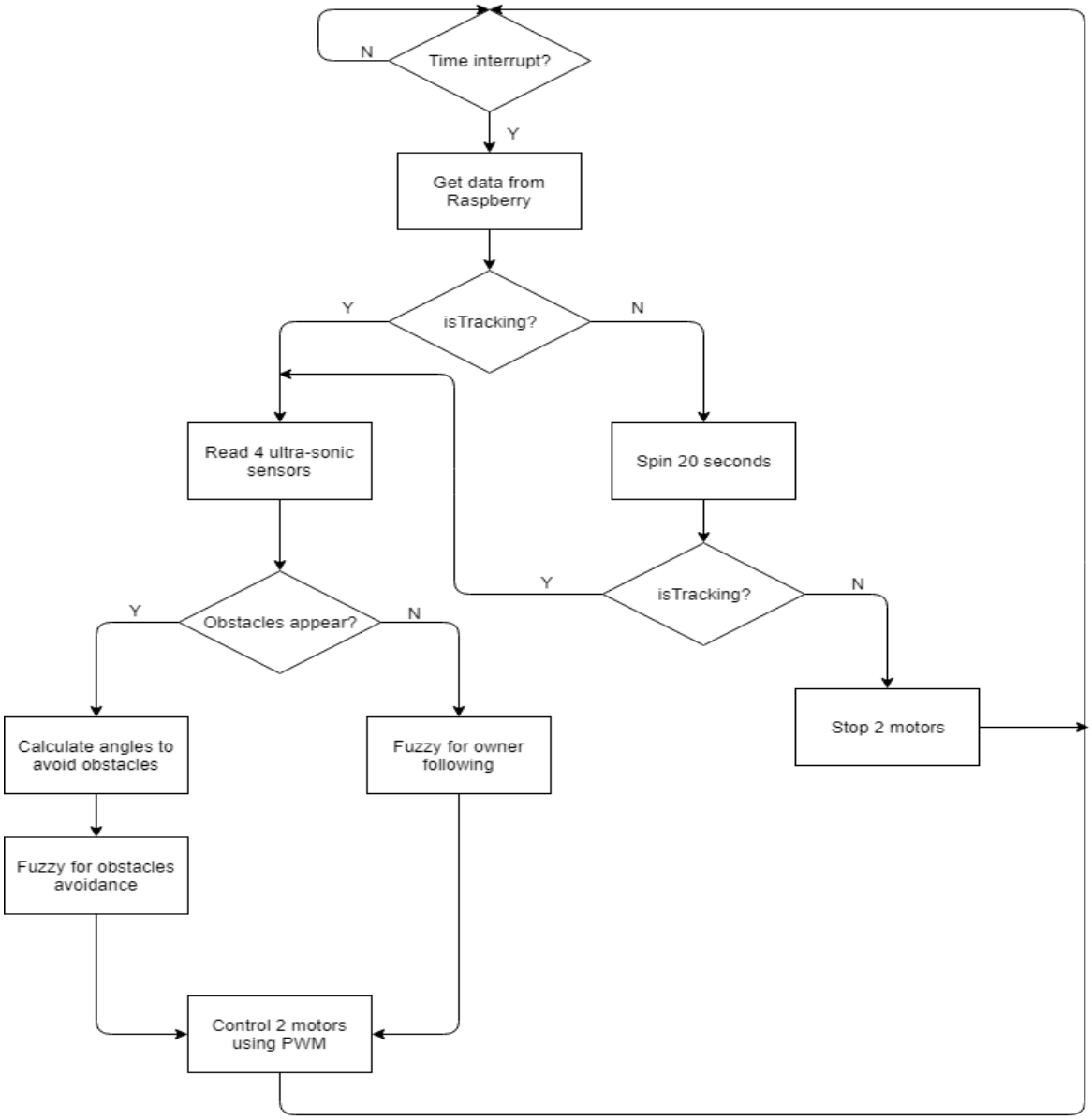


Figure 4.21: *ARM's operation diagram.*

The diagram loops every 100ms.

4.3.1 Receiving suitcase data

Configuration for ARM as the slave-side of the SPI connection:

```
/* SPI1 parameter configuration*/
hspi1.Instance = SPI1;
hspi1.Init.Mode = SPI_MODE_SLAVE;
hspi1.Init.Direction = SPI_DIRECTION_2LINES;
hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
hspi1.Init.NSS = SPI_NSS_HARD_INPUT;
hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
hspi1.Init.CRCPolynomial = 10;
if (HAL_SPI_Init(&hspi1) != HAL_OK)
{
  Error_Handler();
}
```

ARM receives the owner-tracking result from Raspberry via SPI every 100ms.

```
//Get data from Raspberrry through SPI
HAL_SPI_Receive_DMA(&hspi1,&receivebuffer[0],7);
```

The received buffer consists of 7 bytes including: *isTracking*, *error_Position*, *error_Distance*.

Since SPI transfer using bytes, we need to convert 2 bytes to int for each error:

```
error_Position= (int16_t)(((int16_t)receivebuffer[2]<<8)|(int16_t)receivebuffer[1]);
error_Distance= (int16_t)(((int16_t)receivebuffer[4]<<8)|(int16_t)receivebuffer[3]);
```

4.3.2 Reading ultrasonic sensors

There are 4 ultrasonic sensors in front of our suitcase for obstacles detection.

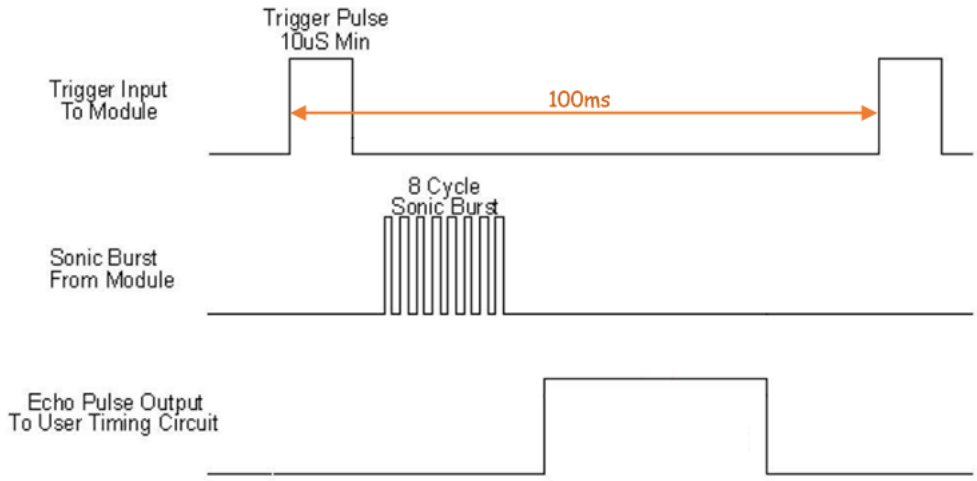


Figure 4.22: *Ultrasonic sensor operation.*

- The trigger pulse is set every 100ms with a period of 10us.
  - Timer 100ms:

```
//Set trigger signal
HAL_GPIO_WritePin(GPIOD, GPIO_PIN_1,GPIO_PIN_SET);
HAL_GPIO_WritePin(GPIOD, GPIO_PIN_3,GPIO_PIN_SET);
HAL_GPIO_WritePin(GPIOD, GPIO_PIN_4,GPIO_PIN_SET);
HAL_GPIO_WritePin(GPIOD, GPIO_PIN_5,GPIO_PIN_SET);
HAL_TIM_Base_Start_IT(&htim2);
```

  - Timer 10μs

```
HAL_GPIO_WritePin(GPIOD,GPIO_PIN_1,GPIO_PIN_RESET);
HAL_GPIO_WritePin(GPIOD,GPIO_PIN_3,GPIO_PIN_RESET);
HAL_GPIO_WritePin(GPIOD,GPIO_PIN_4,GPIO_PIN_RESET);
HAL_GPIO_WritePin(GPIOD,GPIO_PIN_5,GPIO_PIN_RESET);
HAL_TIM_Base_Stop_IT(&htim2);
```

- Echo pulse is calculated by GPIO external interrupt with rising and falling edge using a 1us timer.
  - GPIO external interrupt:

```
if (HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_1))
{
  en_sensor1=1;
  time_sensor1=0;
}
else
{
  echo_sensor1=time_sensor1;
  en_sensor1=0;
}
```

– Timer 1μs

```
if (en_sensor1==1) time_sensor1++;
```

- Distance measurement every 100ms:

```
//Calculate distance
distance1=echo_sensor1*0.0001*340/2;
```

4.3.3 Calculate angle to avoid

The suitcase uses four ultrasonic sensors placing in front of the suitcase at -60°, -30°, 30°, 60° in order to detect obstacles in four directions respectively.

Based on the shape of the suitcase, we define a suitable sensitivity bubble for our model to find obstacles without letting noises affect the suitcase performance. If an object gets inside this bubble, it will be considered as an obstacle on the way.

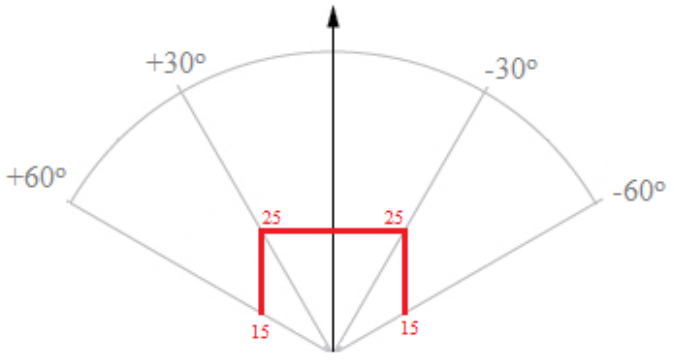


Figure 4.23: *Sensitivity bubble of the suitcase.*

When an obstacle is detected, the rebound angle is measured by the simple, real-time obstacle avoidance algorithm formula:

```
alpha=(-distance1*60-distance2*30+distance3*30+distance4*60)/(distance1+distance2+distance3+distance4);
```

However, this angle formula causes a problem when there is an obstacle like the figure below. In this case, our solution is that whenever the two middle ultrasonic sensors detect obstacles at the same time, the angle will be 90°.

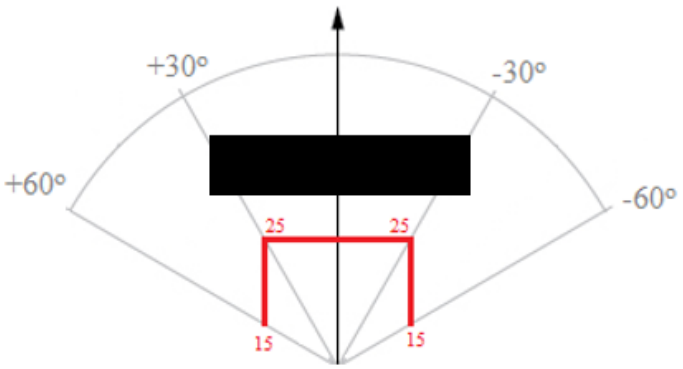
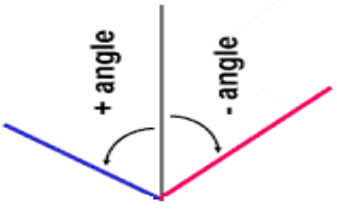


Figure 4.24: *A special case of obstacle.*

4.3.4 Fuzzy for obstacle avoidance

- **Input:**
  - ***Angle:*** refers to the direction where the suitcase needs to follow to avoid obstacles. This angle has already been measured in the previous step.
    - If $angle > 0$: The suitcase is forced to turn left to avoid obstacle.
    - If $angle < 0$: The suitcase is forced to turn right to avoid obstacle.



  - ***Current PWM:*** refers to the current PWM on each motor. The higher this value is, the faster the speed is.

- **Output:** PWM difference value for each motor. The result refers to the difference in speed for each motor to control obstacle avoidance.
- **Rules for the two motors:**

RIGHT		Angle				
		NB	NS	ZE	PS	PB
Current_PWM	LO	NS	ZE	PS	PM	PB
	ME	NM	NS	ZE	PS	PM
	HI	NB	NM	NS	ZE	PS

LEFT		Angle				
		NB	NS	ZE	PS	PB
Current_PWM	LO	PB	PM	PS	ZE	NS
	ME	PM	PS	ZE	NS	NM
	HI	PS	ZE	NS	NM	NB

Figure 4.25: *Fuzzy rules of obstacles avoidance.*

4.3.5 Fuzzy for owner following

- **Input:**
  - ***Position error (ePosition):*** refers to the difference between the center of the camera and the center of the detected frame. This input is measured and sent from Raspberry Pi via SPI. The purpose of this variable is to ensure that the owner is right in front of the suitcase.

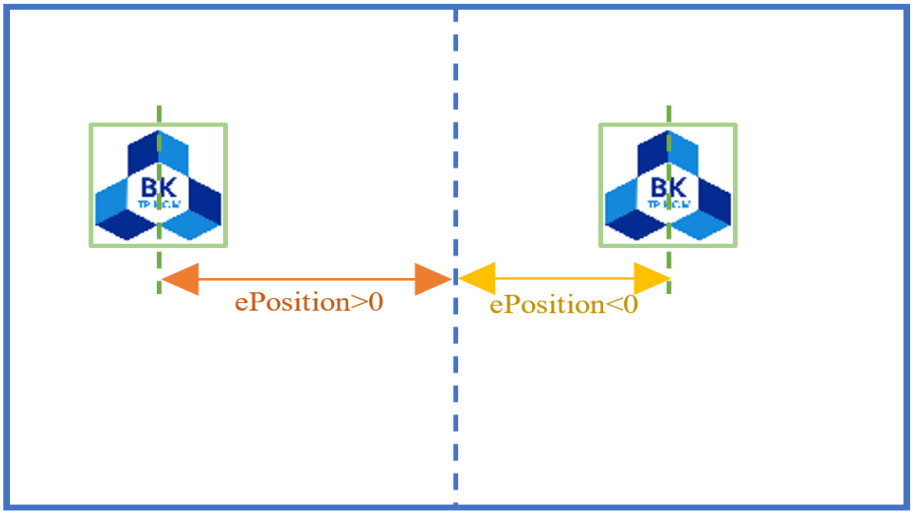


Figure 4.26: *ePosition calculation.*

* If $ePosition > 0$: The owner is to the left of the camera view.
* If $ePosition < 0$: The owner is to the right of the camera view.

– ***Distance error (eDistance):*** refers to the difference between the height of the detected frame and the expected height provided by the owner. This input is also measured and sent from Raspberry Pi via SPI. The purpose of this variable is to ensure a specific distance between the suitcase and the owner. Since there can be a lot of interference from people passing by, using an ultrasonic sensor for distance measurement was crossed out.

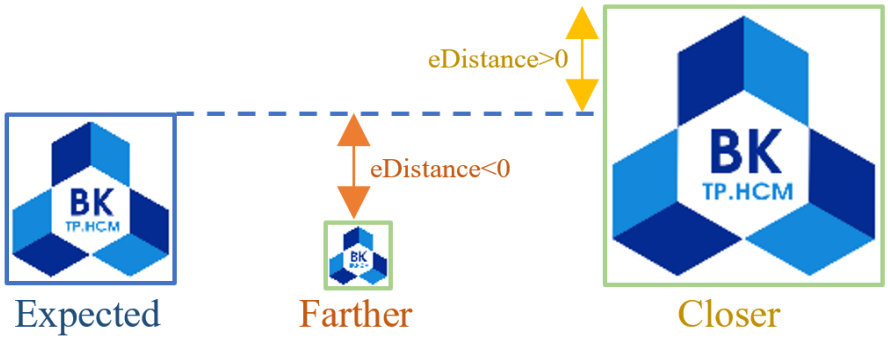


Figure 4.27: *eDistance calculation.*

* If $eDistance < 0$: The owner is further than expected.
* If $eDistance > 0$: The owner is closer than expected.

- **Output:** PWM difference value for each motor. The result refers to the difference in speed for each motor to keep track of the owner.

- **Rules for the two motors:**

RIGHT		ePosition				
		NB	NS	ZE	PS	PB
eDistance	NE	NS	ZE	PS	PM	PB
	ZE	NM	NS	ZE	PS	PM
	PO	NB	NM	NS	ZE	PS

LEFT		ePosition				
		NB	NS	ZE	PS	PB
eDistance	NE	PB	PM	PS	ZE	NS
	ZE	PM	PS	ZE	NS	NM
	PO	PS	ZE	NS	NM	NB

Figure 4.28: *Fuzzy rules of obstacles avoidance.*

4.4 Server diagram

Server is responsible for receiving data from the suitcase and transmit to Android client. Server is written in JavaScript, which uses the powerful open source server environment Node.js.

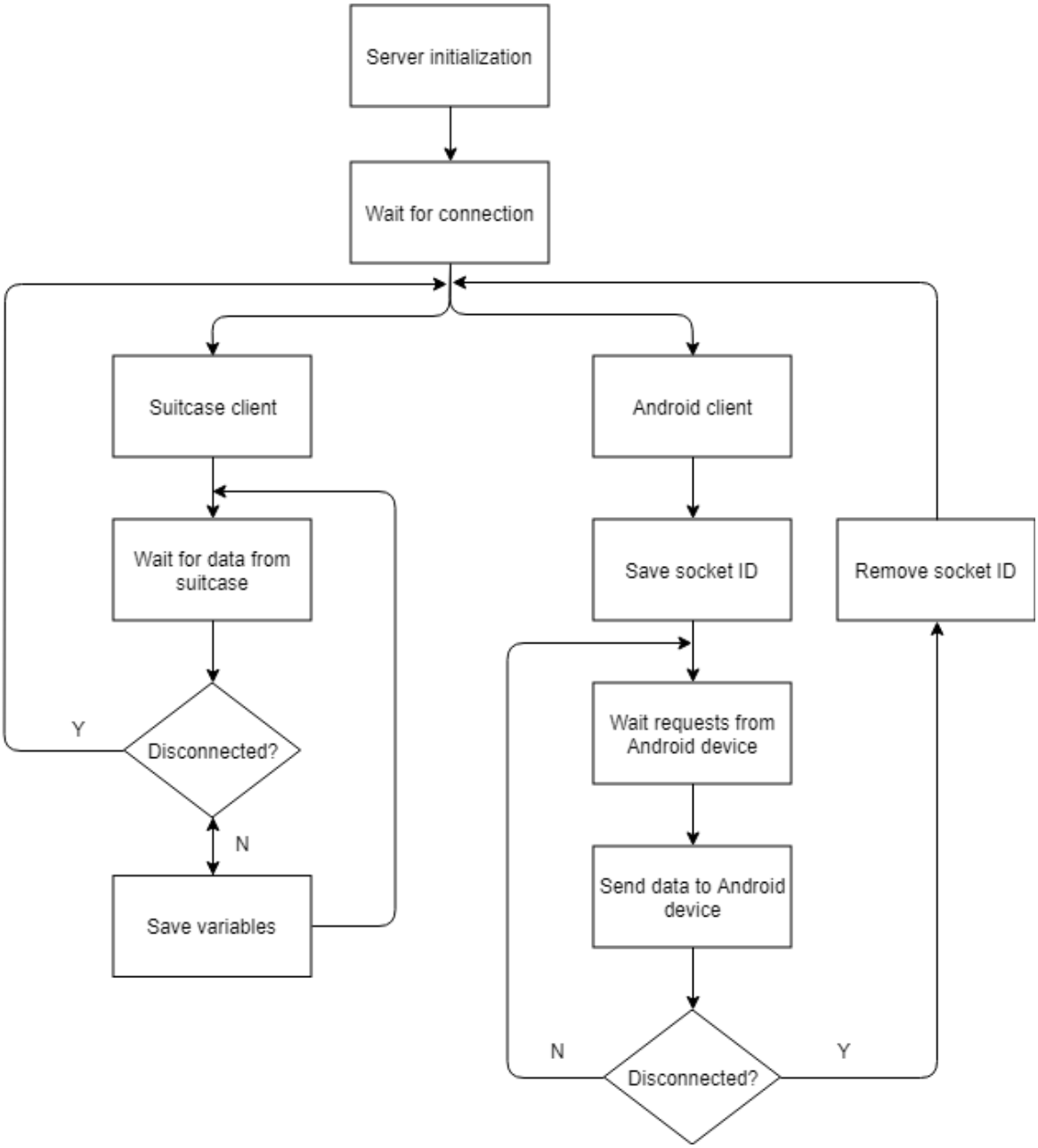


Figure 4.29: *Server diagram.*

4.4.1 Introduction to Node.js server environment

Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine [16]. It runs single-threaded, non-blocking, asynchronously programming, which is very memory efficient. Node.js allows requests to be processed without any delays, even when it is handling with thousands of concurrent connections. It is a good choice for

event-driven servers, data-intensive and real-time applications.

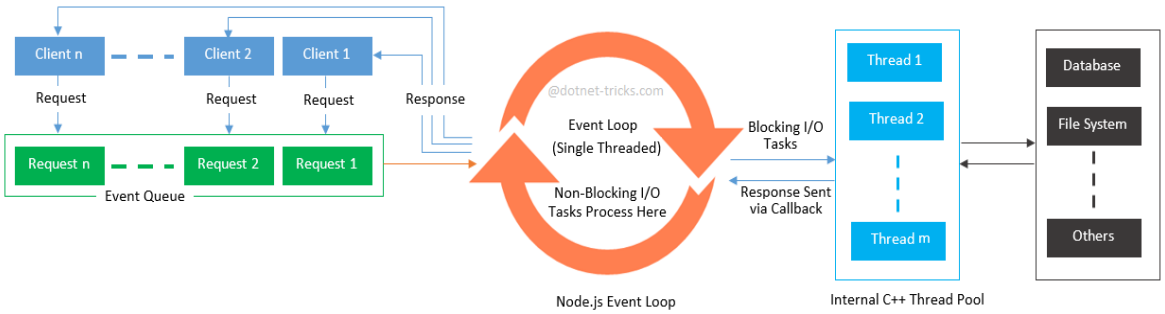


Figure 4.30: *Node.js execution model.*

Steps of code execution:

- Clients send their request to Node.js server
- Node.js server receives and puts them into an Event Queue
- Then, Node.js uses JavaScript Event Loop to continuously process each client request from the Event Queue.
- If the client request blocking I/O operations, it will access to the internal thread pool to handle the request. If the client request does not require any blocking I/O operations, the tasks will be processed in the Event Loop
- Finally, the Event Loop send the response back to the respective client.

4.4.2 Introduction to Socket.IO library

Socket.IO is a library that uses the WebSocket to provide the interface. It enables real-time, bidirectional and event-based communication between the browser and the server [17]. It consists of:

- Node.js server.
- A Javascript client library for the browser.

We choose socket.IO due to its capability of event-based and broadcasting application between server and clients.

4.4.3 How our Server works

- There are two clients expected to connect to our server, one is the suitcase, and another is any Android devices.
- At first, the server waits and listens for a client to make a connection request.
- When there is a new connection to an Android client:
  - The server will notify this client whether the suitcase client is ON or OFF.
  - At the same time, it will save the Android ID to a list for later use and of course, delete that ID if it disconnects.
- When there is a new connection to the Raspberry Pi client (suitcase client):
  - The server will inform all Android clients in the mentioned list.
- After that, the server will receive messages simultaneously from all clients and give a respective response:
  - Receive suitcase status (tracking or lost) from the Raspberry Pi client and send it to Android client(s) when it is requested.
  - Receive captured image (only when the suitcase is lost) from the Raspberry Pi and send it to Android client(s) when it is requested.

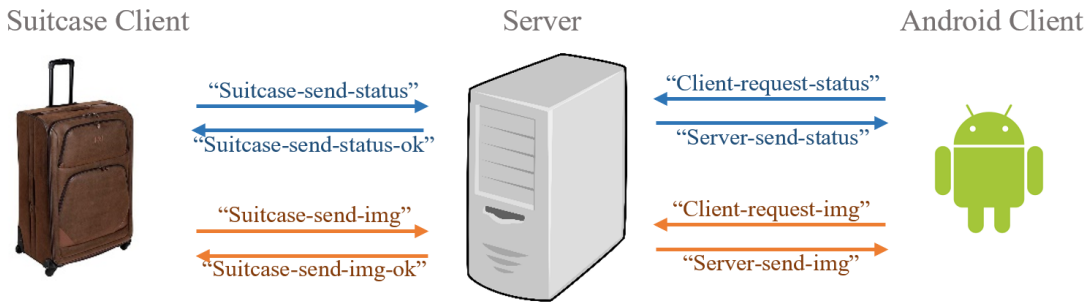


Figure 4.31: *Client-server request and response overview.*

- In order to publicly access the server, we use Heroku, which provides a free cloud web hosting services to run apps, blogs or bots without the hassle of managing servers [18]. Our server interface can be found at `https://suitcase-server.herokuapp.com`. This interface is made for mobile phones with another operation system, for example, iPhone devices to keep track of the suitcase.

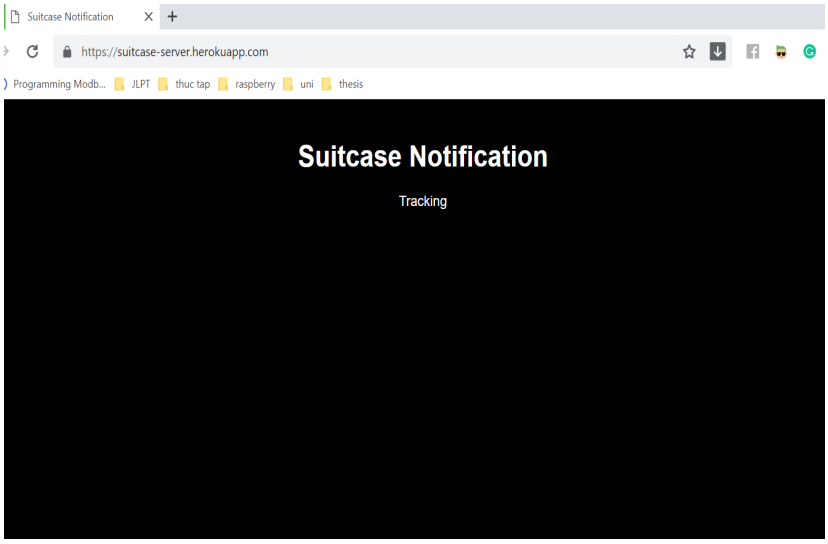


Figure 4.32: *Web page when the suitcase is tracking.*

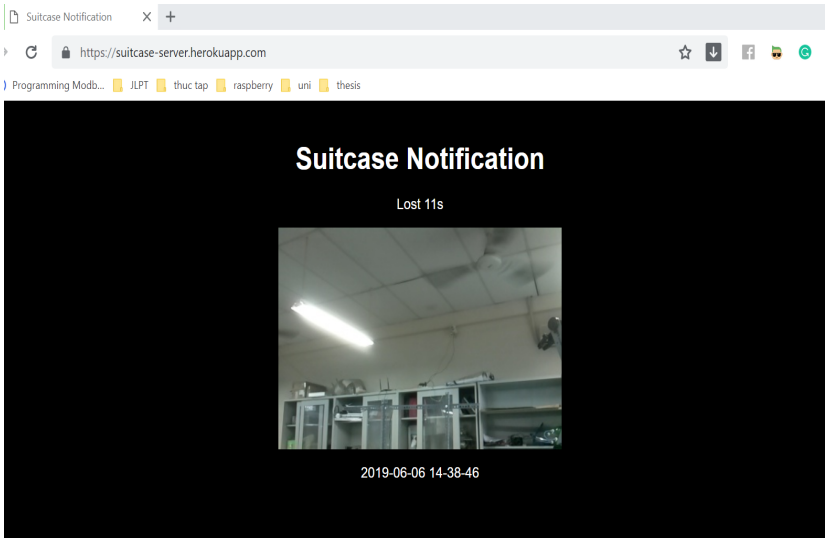


Figure 4.33: *Web page when the suitcase is lost.*

4.5 Android diagram

Unlike Apple's iOS, Google Android offers users better experience as an open-source Linux-bases operation system. So far, Java is the most common language for programmers to get started with application development with its object-oriented programming principle.

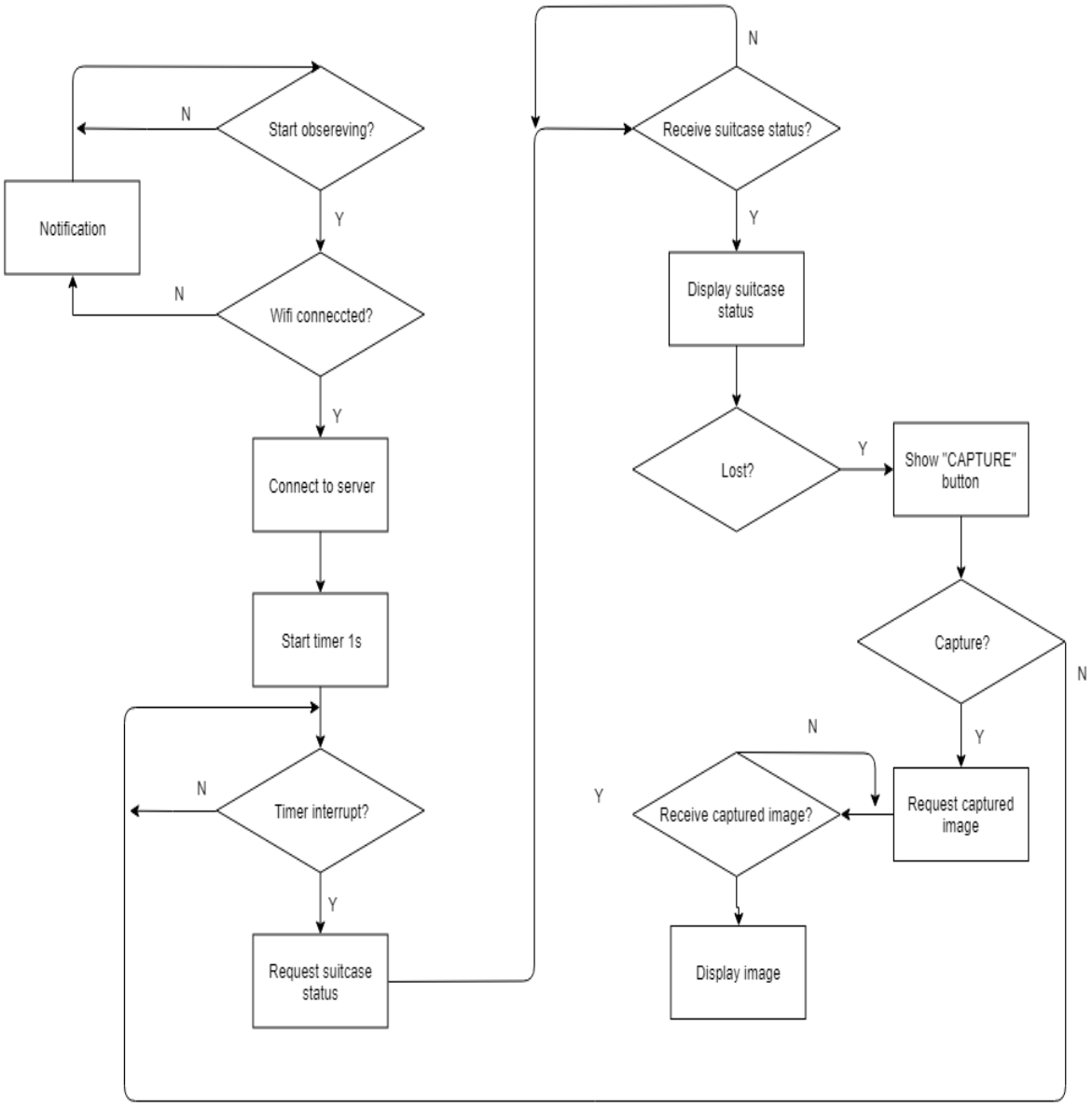


Figure 4.34: *Android operation diagram.*

We built a simple Android application to help the owner get the real-time status of the suitcase. This is an improvement in our suitcase since we have found some dead point in our suitcase that may cause luggage lost. This application is written in Java for Android OS.

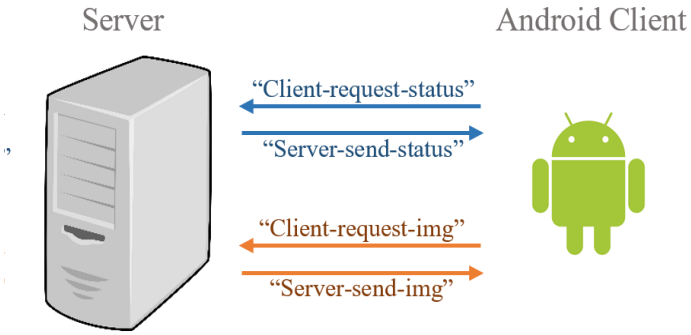


Figure 4.35: *Android requests to Server.*

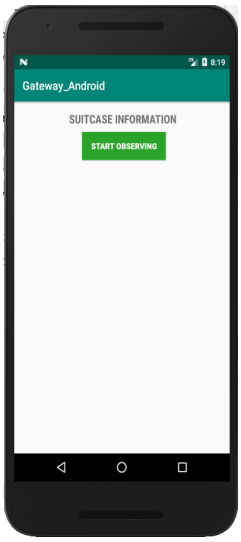


Figure 4.36: *Classes of Object detection are declared in label mapping file.*

When the user pushes the "START OBSERVING" button, a message will be sent to server saying that this Android device wants to access to the information relating to the suitcase. We also use the Socket.IO library for server connection:

```
/*----HEROKU CONNECTION----*/
private void Connect2Server(){
    try {
        mSocket = IO.socket(url_heroku);
        mSocket.connect();
        Toast.makeText( context: this,  text: "Connected to Server!", Toast.LENGTH_SHORT).show();
    } catch (URISyntaxException e) {
        Toast.makeText( context: this,  text: "Server fails to start...", Toast.LENGTH_SHORT).show();
        e.printStackTrace();
    }
}
```

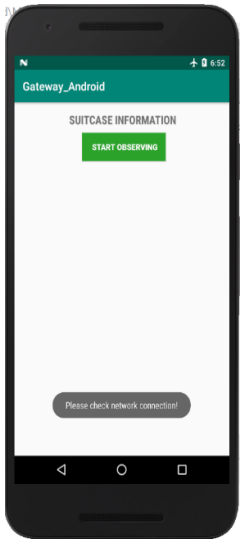


Figure 4.37: *A pop-up notification if there is no wifi connection.*

If connected, the Android application will continuously send requests of status every second to the server and listen to messages including the suitcase mode and the suitcase status.

```
/*----HANDLER FOR UPDATING----*/
private Runnable updateTimerThread = () → {
        mSocket.on( event: "suitcase-off",suitcase);
        mSocket.emit( event: "client-request-status");
        mSocket.on( event: "server-send-status", mydata);
        customHandler.postDelayed( r: this,  delayMillis: 1000);
};
```

Figure 4.38: *A pop-up notification if there is no wifi connection.*

If the suitcase loses its owner, the application will show a “GET LATEST CAPTURE” button. In this case, the owner can see the latest image taken by the Raspberry Pi camera, recognize the features in that shot and know where the suitcase is.

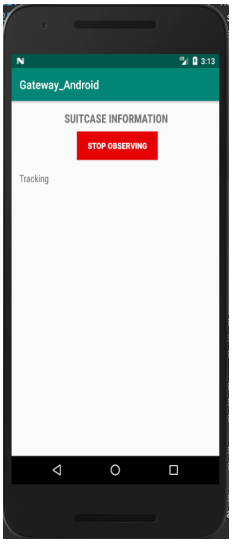


Figure 4.39: *An example of the tracking status.*

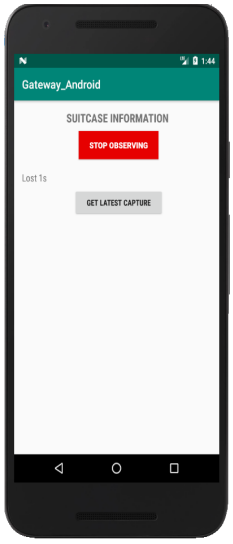


Figure 4.40: *An example of the lost status.*

When the user pushes the "GET LATEST CAPTURE" button, the application will request the server for the image:

```
/*----BUTTON CAPTURE----*/
btnCapture.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        mSocket.emit( event: "client-request-img");
        mSocket.on( event: "server-send-img", mydata_img);
    }
});
```

Meanwhile, there is also a notification if the user is not observing the application (when using another application or the screen has been turned off). Noted that this notification only works if you have opened the app and press "START OBSERVING" button

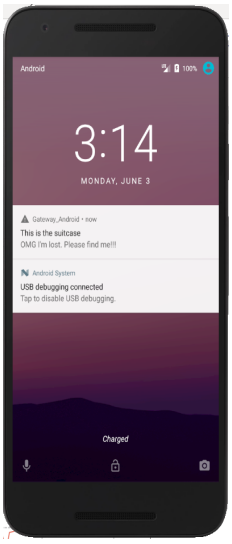


Figure 4.41: *Pop-up notification on lock screen.*

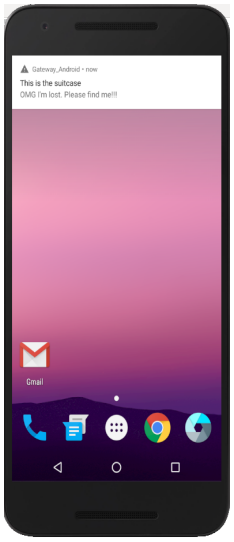


Figure 4.42: *Pop-up notification on main screen.*

Chapter 5

RESULTS AND CONCLUSION

5.1 Results

5.1.1 Tracking by camera

The tracking result of Raspberry Pi with Intel Movidius works in many cases including different distances, different rotations, different background or even when a part of the logo is hidden.

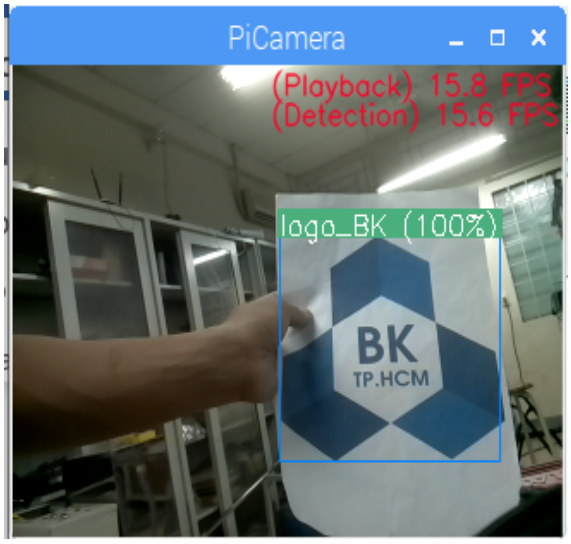


Figure 5.1: *Tracking result of normal condition.*

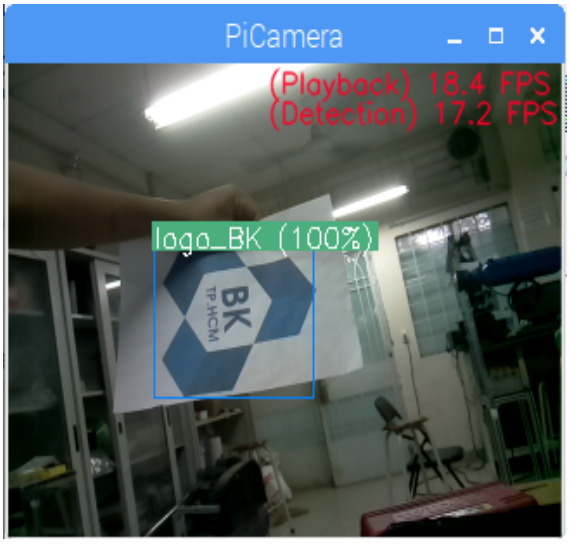


Figure 5.2: *Tracking result of rotated object.*

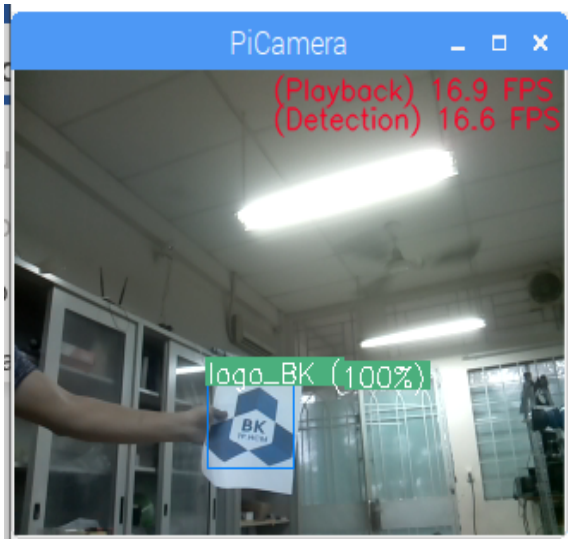


Figure 5.3: *Tracking result in far distance.*

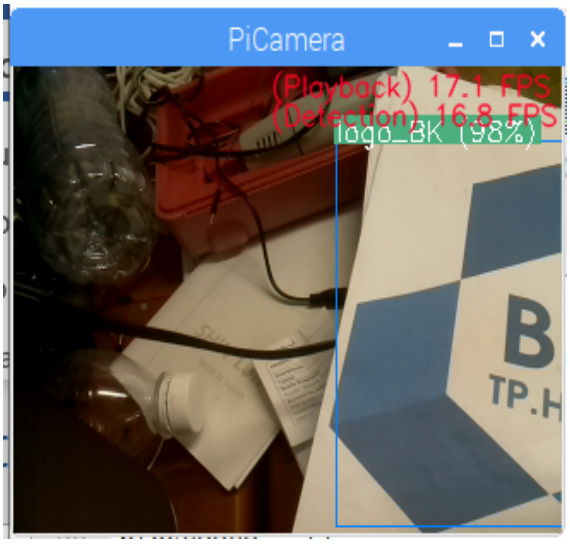


Figure 5.4: *Tracking result when a part of object is hidden.*

The detection speed is around 15-18fps, which allows the Raspberry Pi to transfer data with the sample rate 100ms.

In a low-light environment, camera cannot detect objects.

5.1.2 Following owner

The ARM STM32F4 receives the exact tracking result from Raspberry Pi every 100ms. The results can be seen as the following figures.

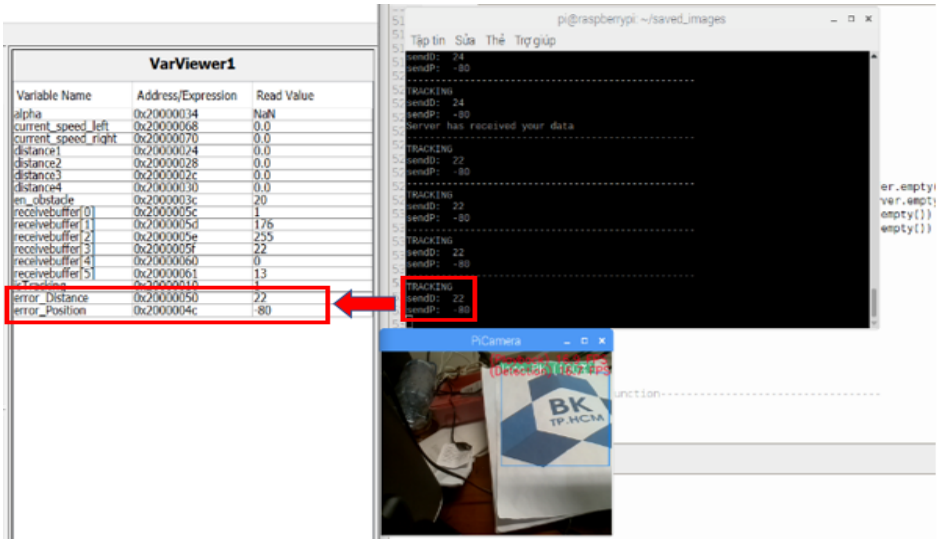


Figure 5.5: *First result of data transfer from Raspberry Pi to ARM.*

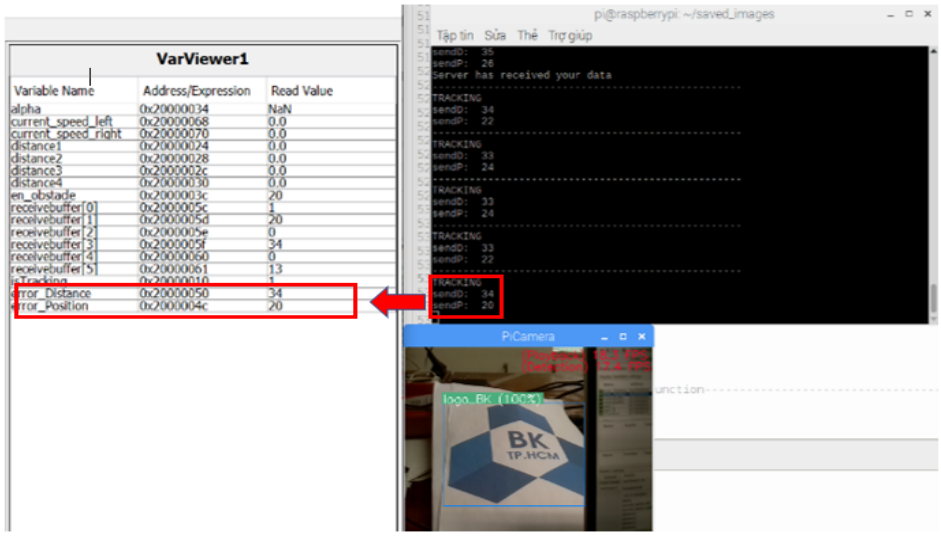


Figure 5.6: *Second result of data transfer from Raspberry Pi to ARM.*

The following charts will show the results of fuzzy logic controller.

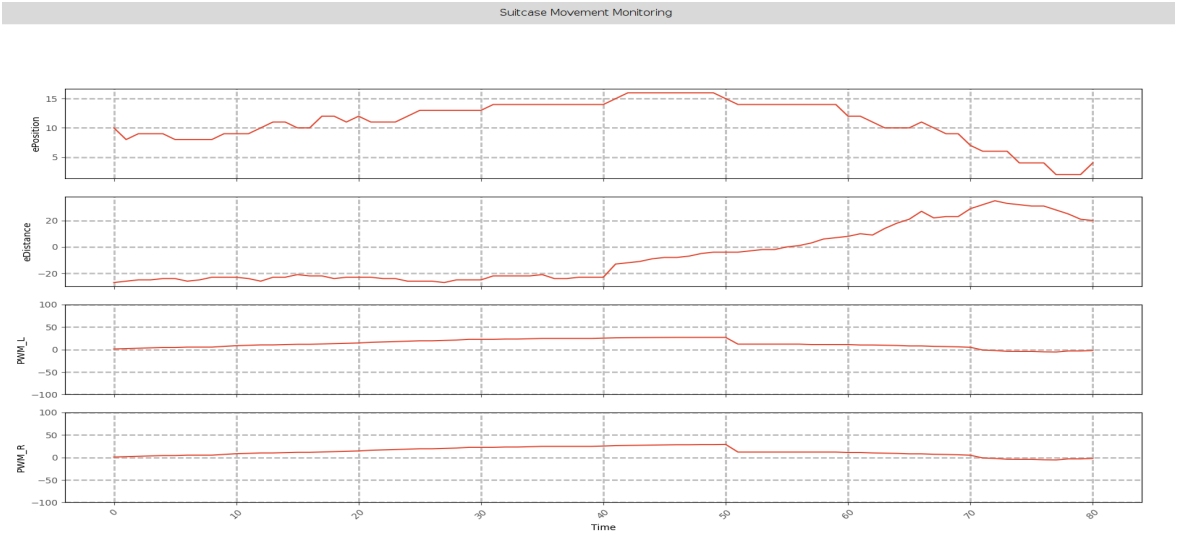


Figure 5.7: *The owner is a little far and around the middle.*

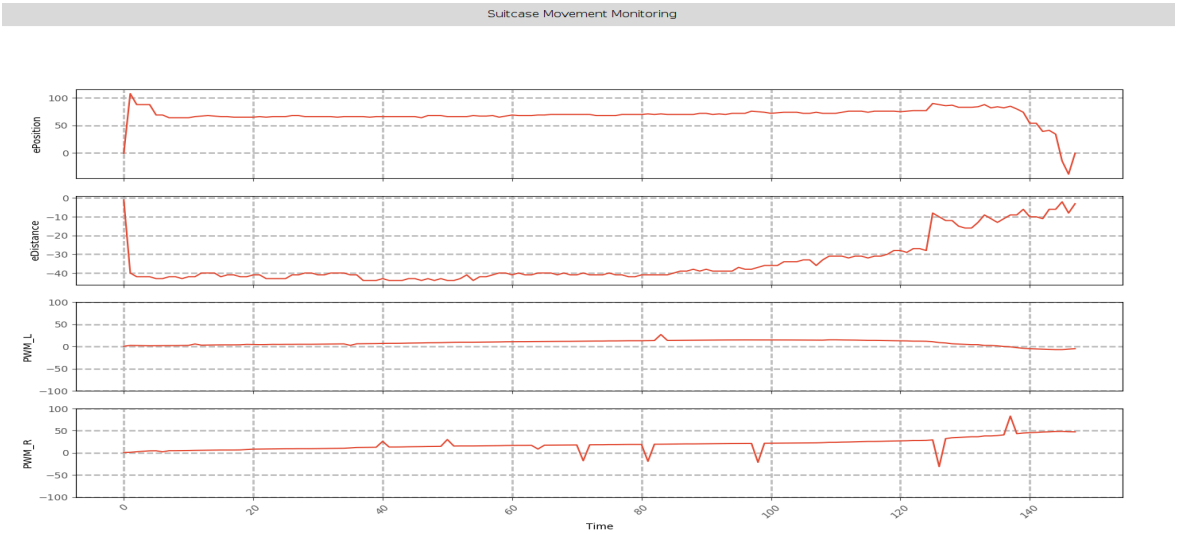


Figure 5.8: *The owner is far and to the left.*

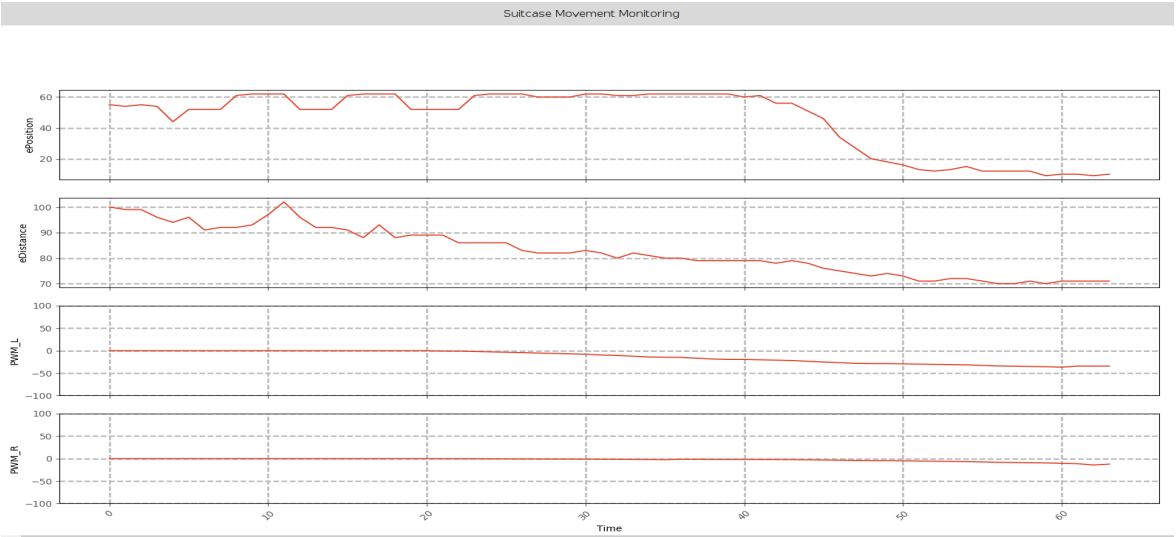


Figure 5.9: *The owner is near and to the left.*

5.1.3 Avoiding obstacle

Four ultrasonic sensors allow the suitcase to avoid obstacles in many cases:

- One obstacle:

Figure 5.10: *Result of one obstacle avoidance.*

- Two obstacles:

Figure 5.11: *Result of two obstacles avoidance.*

- Special case when obstacle is right in the middle of the way:

Figure 5.12: *Result of avoidance when one obstacle is in the middle of the way.*

5.1.4 Using server and Android application

The server receives messages and responses accordingly.

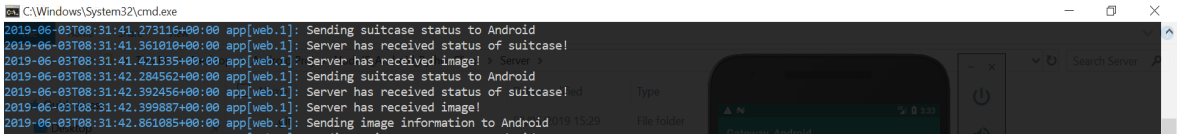


Figure 5.13: *Response from server to Android device.*

The Android application sends request and receives response from the server successfully, which allows the owner to:

- Observe whether the suitcase is on or off.
- Observe whether the suitcase is tracking or has been lost.
- Get the latest image (if lost) in order to manually find the suitcase.
- Get all necessary notifications.

The website interface also allows the user with non-Android device to keep track of the suitcase.

It takes a second for the suitcase to send data to the Android application through server including the status and the image.

5.2 Discussion

5.2.1 Merit

- The suitcase model has been finished including simple owner tracking and object detection.
- Using a server and an Android app enables the owner to keep track of their suitcase, which is partly helpful when the suitcase is lost due to tracking failure.
- Without turning on the system, the smart suitcase can perform as a two-wheel suitcase.

5.2.2 Defect

- The suitcase cannot detect its owner in a low-light environment.
- Usually, after avoiding obstacles, the suitcase will lose its owner and start to spin for searching.
- The suitcase needs a wifi connection to update to server.
- The motion is still slow.

5.3 Future work

- Improve the model to an exact tracking-owner and avoiding-obstacle suitcase with a better appearance.
- Improve computer vision algorithm to detect in high accuracy just by selecting the owner in the camera screen.
- Form an indoor GPS to spot the suitcase location in case it is lost.

List of Abbreviations and Acronyms

PWM: Pulse Width Modulation

ARM: Advanced RISC Machine

CNN: Convolutional Neural Network

ConvNet: Convolutional Neural Network

CPU: Central Processing Unit

DSC: Depthwias Separable Convolutions

FPS: Frames Per Second

GPS: Global Positioning System

GPU: Graphical Processing Unit

LMDB: Lightning Memory-Mapped Database

MVCC: Multiversion Concurrency Control

NCS: Neural Compute Stick

ROI: Region of Interest

SoC: System-on-chips

SPI: Serial Peripheral Interface

VFH: The Vector Field Histogram Algorithm

VPU: Vision Processing Unit

References

[1] Assoc. Prof. Huynh Thai Hoang, Ho Chi Minh University of Technology, *Course of Introduction to Intelligent Control*

[2] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam, *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*, April 2017.

[3] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg, *SSD: Single Shot MultiBox Detector*, December 2016.

[4] V. Lumelsky and T. Skewis, *Incorporating Range Sensing in the Robot Navigation Function*, IEEE Transactions on Systems, Man and Cybernetics, pp. 1058-1068, 1990.

[5] M. I. Ribeiro, *Obstacle Avoidance*, November 2005.

[6] O. Khatib, *Real-time Obstacle Avoidance for Manipulators and Mobile Robots*, IEEE International Conference on Robotics and Automation, March 1985.

[7] I. Susnea, V. Minzu and G. Vasiliu, *Simple, Real-Time Obstacle Avoidance Algorithm for Mobile Robots*, Vols. ISBN: 978-960-474-144, January 2009.

[8] J. Borenstein and Y. Koren, *The Vector Field Histogram - Fast Obstacle Avoidance for Mobile Robots*, IEEE Journal of Robotics and Automation, pp. 278-288, 1991.

[9] Wikipedia, *Convolutional Neural Network*, `https://en.wikipedia.org/wiki/Convolutional_neural_network`

[10] *Data Structure and Algorithms - Queue*, `https://www.tutorialspoint.com/data_structures_algorithms/dsa_queue.htm`

[11] Jonathan Hui, *What do we learn from single shot object detectors?*, https://medium.com/@jonathan_hui

[12] *What Is IBM i Single Level Storage? And Why Should I Care?*, https://www.source-data.com/

[13] NVIDIA Corporation, *Production Deep Learning with NVIDIA GPU Inference Engine*, https://devblogs.nvidia.com

[14] *Study Confirms Mass Stranding of Whales Caused by Sonar Mapping*, https://inhabitat.com/

[15] Texas Instruments Incorporated, *LM2596 Datasheet*, http://www.ti.com/lit/ds/symlink/lm2596.pdf

[16] Linux Foundation, *NodeJS*, https://nodejs.org

[17] *socket.io*, https:/socket.io

[18] *Heroku*, https://www.heroku.com

[19] Official website of Raspberry Pi Foundation, https://www.raspberrypi.org/

[20] Official website of STMicroelectronics, https://www.st.com/content/st_com/en.html

Referencesstmdata ARM Cortex-M4 32b MCU+FPU Datasheet, STMicroelectronics, https://www.st.com/resource/en/datasheet/dm00037051.pdf

[21] Intel Software Developer Zone, https://software.intel.com/en-us/neural-compute-stick

[22] Website of Intel® Movidius™ Myriad™ X VPU, https://www.movidius.com/myriadx

[23] OpenVINO™ toolkit Documentation , https://docs.openvinotoolkit.org/latest/index.html