

**Data fusion of video camera and Laser Range Finder (LRF)
for obstacle avoidance**

By

Somia Mohamed Ali Dousa

17876

Dissertation submitted in partial fulfilment of
the requirement for the
Bachelor of Engineering (Hons)
(Electrical and Electronics)

JANUARY 2016

Universiti Teknologi PETRONAS

Bandar Seri Iskandar

31750 Tronoh

Perak Darul Ridzuan

CERTIFICATE OF APPROVAL

Data fusion of video camera and Laser Range Finder (LRF) for obstacle avoidance

By

Somia Mohamed Ali Dousa

17876

Dissertation submitted in partial fulfilment of
the requirements for the
Bachelor of Engineering (Hons)
(Electrical and Electronics)

Approved by,

Mr. Patrick Sebastian

Universiti Teknologi PETRONAS

TRONOH, PERAK

JAN 2016

CERTIFICATE OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the reference and acknowledgments, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.

SOMIA MOHAMED ALI DOUSA

ABSTRACT

Many people are involved in car accidents yearly, according to the World Health Organization about 1.25 million people die every year as a result of road traffic crashes. [1] This number is expected to increase if no immediate actions are taken. A collision avoidance system is designed to reduce the severity of an accident by detecting any imminent crash using sensors such as radar, camera and laser range finder. Most of current systems rely on only one sensor for detection. However, using only one sensor is proven to be insufficient in providing full information of the nature of the obstacle and crash. This arises the need for systems that fuse the data of two sensors to provide better result for object detection.

In this project, a design of collision avoidance system is presented. The system uses camera and laser range finder (LRF) to detect any possible threat. It is a continuation of previous project that has already implemented the obstacle detection algorithms using motion template technique on embedded platform. The camera is used to identify the moving objects or obstacles in scenery. The data obtained from the LRF on the other hand is used to determine the distance between the car and the detected obstacle. This system is built using three Raspberry Pies boards in order to distribute the load of computations on the three processors using the concept of parallel programming and Message passing interface (MPI) library.

Beside the motion detection algorithm, the motion detection using the frame difference algorithm was developed as well. The results obtained from the motion template were more satisfactory than the frame difference ones, however, both algorithm require the camera to be static in order to obtain accurate result therefore, future work will focus on implementing algorithms that can achieve the motion detection even while the camera is moving. Finally, the fusion of the data was achieved successfully. The network has some delay that is happened as a result transferring large amount of data, this was solved by reducing the frame size.

ACKNOWLEDGEMENT

First and foremost, praise to Allah the almighty for giving me the strength, health and patient to complete my final year project with ease and success.

I would like also to express my ample gratitude to my supervisor Mr. Patrick for his continuous support. This work would not be possible without his enlightened supervision.

Finally, my appreciation and love go to my beloved family for the trust they put on me. I would not achieve any of this without their blessings and prayers.

TABLE OF CONTENTS:

| | |
|---|------------|
| CERTIFICATE OF APPROVAL | i |
| CERTIFICATE OF ORIGINALITY | ii |
| ABSTRACT | iii |
| ACKNOWLEDGEMENT | iv |
| CHAPTER 1 | 1 |
| INTRODUCTION | 1 |
| 1.1. Background of Study | 1 |
| 1.2. Problem statement | 2 |
| 1.3. Objectives and Scope of Study | 3 |
| 1.3.1 Objectives | 3 |
| 1.3.2 Scope of Study | 3 |
| CHAPTER 2 | 4 |
| LITERATURE REVIEW | 4 |
| 2.1. Detection of obstacles movement using video camera | 6 |
| 2.2. Laser Range Finder | 7 |
| 2.3 Raspberry pi network | 8 |
| 2.4 Message Passing Interface (MPI) | 9 |
| CHAPTER 3 | 12 |
| METHODOLOGY | 12 |
| 3.1 Overall methodology | 12 |
| 3.2 Project Flow chart | 13 |
| 3.3 Motion detection algorithms | 15 |
| 3.3.2 Frame Difference algorithm | 15 |
| 3.3.1 Motion Template | 16 |
| 3.4 Tools and software | 17 |
| 3.5 Key milestones | 17 |
| 3.6 Gant Chart | 18 |
| CHAPTER 4 | 20 |
| RESULTS and DISCUSSION | 20 |
| 4.1 Raspberry Pi network | 20 |
| 4.2 Laser range finder (LRF) interface with first Raspberry Pi | 22 |

| | |
|--|----|
| 4.3 Camera interface with the second Raspberry Pi for obstacles movement detection ----- | 23 |
| 4.4 Data fusion of LRF data and camera in central Raspberry Pi ----- | 26 |
| 4.5 Network Analysis ----- | 29 |
| 4.6 New network topology ----- | 32 |
| CHAPTER 5----- | 34 |
| CONCLUSION AND RECOMMENDATION ----- | 34 |
| 5.1 CONCLUSION----- | 34 |
| 5.2 RECOMMENDATION----- | 34 |
| REFERENCES----- | 35 |
| APPENDICES----- | 36 |

TABLE OF FIGURES:

| | |
|---|----|
| FIGURE 1: PROPOSED COLLISION AVOIDANCE SYSTEM ----- | 2 |
| FIGURE 2: TRANSFORMATION FROM LRF COORDINATE TO CAMERA COORDINATE----- | 5 |
| FIGURE 3: SOUTHAMPTON UNIVERSITY RASPBERRY PI CLUSTER ----- | 9 |
| FIGURE 4: GENERAL STRUCTURE OF MPI PROGRAM ----- | 10 |
| FIGURE 5: COMMON COLLECTIVE OPERATIONS ----- | 11 |
| FIGURE 6: PROGRAM FLOW ----- | 13 |
| FIGURE 7: FIRST TWO STEPS IN IMAGE PROCESSING----- | 15 |
| FIGURE 8: STEPS OF FRAME DIFFERENCE ALGORITHMS----- | 15 |
| FIGURE 9: STEPS OF MOTION TEMPLATE ALGORITHM ----- | 16 |
| FIGURE 10: RASPBERRY PI NETWORK CONNECTION ----- | 20 |
| FIGURE 11: ILLUSTRATION OF MPI COMMUNICATOR----- | 21 |
| FIGURE 15: LRF CONNECTION WITH RASPBERRY PI----- | 23 |
| FIGURE 16: THE MOVING OBJECT WAS DETECTED----- | 24 |
| FIGURE 17: THE MOVING OBJECT WAS DETECTED 2 ----- | 24 |
| FIGURE 18: THE MOVING OBJECT WAS DETECTED 3 ----- | 24 |
| FIGURE 19: MULTIPLE MOTION DETECTIONS FOR THE SAME OBJECT -- | 25 |
| FIGURE 20: MOTION TEMPLATE RESULT (COLORED)----- | 26 |
| FIGURE 21: MOTION TEMPLATE RESULT (GRAY SCALE)----- | 26 |
| FIGURE 22: CAMERA AND LRF CENTRES ALIGNMENT ----- | 26 |
| FIGURE 23: LRF AND CAMERA SETUP ----- | 27 |
| FIGURE 25: DATA FUSION RESULT (GRAY SCALE) ----- | 28 |
| FIGURE 24: DATA FUSION RESULT (COLORED) ----- | 28 |
| FIGURE 27: DATA FUSION RESULT (COLORED) ----- | 28 |
| FIGURE 26: DATA FUSION RESULT (GRAY SCALE) ----- | 28 |
| FIGURE 28: DATA FUSION RESULT ----- | 28 |
| FIGURE 29: DATA FUSION RESULT ----- | 29 |
| FIGURE 30: SAMPLE OF WIRESHARK NETWORK ANALYSIS RESULT (FRAME DIFFERENCE)----- | 30 |
| FIGURE 31: MPI BUFFER CONCEPT ----- | 31 |
| FIGURE 32: NEW PROPOSED NETWORK TOPOLOGY ----- | 32 |
| FIGURE 33: MEASURING THE CAPTURE TO DISPLAY DELAY ----- | 33 |

LIST OF TABLES:

TABLE 1: SHOWS THE FOUR CONDITIONS UNDER WHICH THE NETWORK
WERE TESTED -----29

TABLE 2: CHARACTERISTICS OF RPI 1 AND RPI 3 NETWORK INTERFACE
-----30

TABLE 3: CHARACTERISTICS OF RPI 2 AND RPI 3 NETWORK INTERFACE
-----30

CHAPTER 1

INTRODUCTION

1.1. Background of Study

According to the World Health Organization (WHO), about 1.25 billion people die yearly as result of road accidents. This number is expected to increase by year of 2030 to make the road accidents one of the top seventh causes of death worldwide. [1] Driver distraction is one of the most common causes of car accidents. According to recent statistic from U.S. National Highway Traffic Safety Administration (NHTSA), 80% of collision and 65% of near car crashes occurred due to driver's inattention and distraction. [2]

With this increasing number of roads accidents and collisions due to drivers' inattention, a system that detects the obstacles such as pedestrians and vehicles and notifies the driver with any possible threat is needed. Such system is usually implemented using only one sensor such as camera, LRF or ultra-sonic, however one sensor alone does not provide full information about the road condition and the possible threat. For example, the camera alone can detect and classify the obstacles however, it does not provide information on the distance between the vehicle and that obstacle. Therefore, the fusion of multiple sensor data is needed for better detection.

In this project, obstacles avoidance system was developed using both camera and laser range finder. LRF can be used to accurately measure the distance between the obstacle and the vehicle using the laser beams. [3] LRF projects the laser beam to the obstacle and then measures the distance by calculating the total time needed for the beam to be reflected by the object. [4]

The project is a continuation of previous project that has already implemented the obstacle detection algorithms using motion templates technique on embedded platform.

Raspberry Pi was chosen as the embedded platform for this project as it is well-known of its low price, small size and low power consumption [6], however, it lacks the processing power which makes processing the data that come from both the camera and LRF quite difficult in a single Raspberry Pi processor, and lead to slowness of the

overall system, therefore, in this project a network of three Raspberry Pis is used. Obstacle detection using video camera and LRF distance measurement data are processed in parallel using two separate Raspberry Pi processors and eventually the data are transferred through network switch to the master Raspberry Pi for analyzing the information and displaying some messages on small screen to the driver. Such network topology is assumed to reduce the processing time and calculation cost.

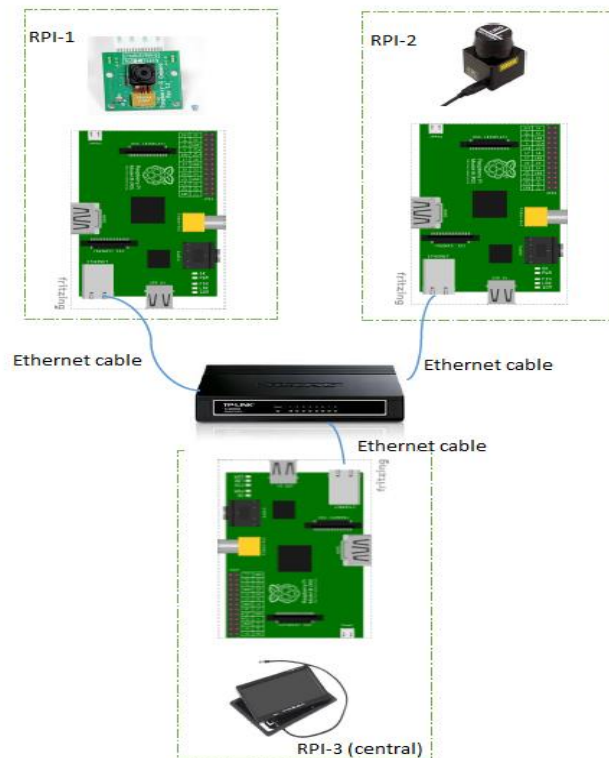


Figure 1: proposed collision avoidance system

1.2. Problem statement

Car accidents is considered as one of the most leading causes of death in recent years, most of these accidents occur due to the driver's inattention and distraction. The current pre-crash alerting systems provide a significant help to drivers to avoid any possible crash or collisions. Most of these systems depend mainly on single sensor such as mono camera or laser range finder to detect any possible crash. However, using only one sensor is not sufficient to provide full information

of the nature of the obstacle and crash. This arises the need for systems that fuse the data of two sensors to overcome this problem.

Implementing such system will require at least two sensors to collect the data from the environment and then process them. This needs a high computational power to enable the system to process and to respond to any possible threat fast and efficiently.

1.3. Objectives and Scope of Study

This project has the following objectives and scope of study:

1.3.1 Objectives

- 1- To develop a collision avoidance system that warns the driver with possible threats
- 2- To fuse data of LRF and camera to provide better estimation of the type and distance of object detected.
- 3- To coordinate distributed computing platforms that process LRF and camera data in parallel and send them to a final platform that makes a decision on possibility of crash or not.

1.3.2 Scope of Study

The project was implemented on Raspberry Pi platform which runs Raspian weezly operating system. The object detection algorithms used in this project were motion template and frame difference using OpenCV library. The data from the real-time video processing for this project were obtained while the camera being static using Raspberry Pi camera (Pi-cam). The camera has 5 MP image resolution and 1080 HD video recording. [7] The communication network between the three Raspberry Pies was implemented using the Message Passing Interface (MPI) library.

CHAPTER 2

LITERATURE REVIEW

In this chapter a literature review and previous related work will be presented.

The existing collision avoidance systems nowadays use different kinds of sensors in order to detect obstacles and notify the drivers with any possible threat. These sensors include multiple options such as ultrasonic, mono camera, LRF and Radar.

Students from both University of VIT and Symbiosis International University have designed a pre-crash system using ultrasonic sensor interfaced with the Raspberry Pi. The system was used to detect the obstacle in the front and in the blind spots of the car. [8] However, the ultrasonic has a short coverage range (25 cm) which makes detecting far objects or threat significantly impossible.

Cameras on the other hand, are usually used in such systems to provide wide information of environment. Such information can be processed in order to detect any possible threat using certain techniques. However, the camera does not perform well during the rain and fog which may affect the overall functionality of the system in such weather condition. [8] Moreover, the distance between the camera and the obstacle information is missing, to obtain the distance information of detected objects, there are some techniques, such as using the stereo camera which has multiple lenses. The stereo cameras take multiple images simultaneously and then use triangulation to calculate the distance. Other sensors like Laser range finder can also be used for measuring the distance. The data acquired from the laser range finder were found to be more accurate and reliable than stereo cameras ones. [3]

Depending on only one sensor has some drawbacks as aforementioned, this is because each sensor has certain limitation, therefore, the concept of fusion of multiple sensor data becomes quite interesting as the decision made based on the output of multiple sensors tended to be more accurate and reliable. [4]

A team of students from, Ritsumeikan University, College of Information Science and Engineering have applied the concept of data fusion of LRF and camera for obstacle avoidance. The team combined between the image data and the LRF data that

represents different depth points by transferring the data obtained from the LRF coordinate to the camera coordinate using Eq. (1):

$$P_c = R \times P_l + t \quad (1)$$

P_c : camera coordinate 3-axis point.

P_l : LRF coordinate 3-axis point.

R, t : the rotation and transition vectors respectively.

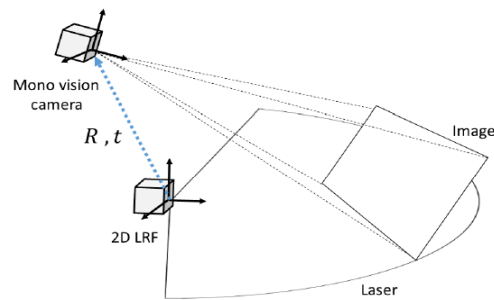


Figure 2: Transformation from LRF coordinate to camera coordinate

Both rotation and transition vectors were obtained using camera and LRF calibration. Using this method, each point in the image now has a depth information. In addition, to decrease the calculation cost, the team applied the concept of parallel programming by running the object detection algorithms in the Graphical processing unit (GPU). [3]

A similar project has been done by a group of master students in Virginia Polytechnic Institute and State University, the project was built for DARPA challenge for building autonomous and unmanned vehicle that can work in the urban environment. They utilized both camera and LRF for the road objects detection and classification. The velocity, size and location of object is specified by LRF. The vision system would then utilize the location information to specify the region of interest in the large image. This region only will be processed in order to classify the object detected. [4]

Therefore, in this project laser range finder will work along with the camera to obtain more detailed information on the status of the obstacles in front of the car.

2.1. Detection of obstacles movement using video camera

Motion detection has been an interesting area of research due to its enormous applications. In general, motion detection algorithms can be classified into three main divisions: Background subtraction methods, Frame subtractions methods and finally optical flow methods. In optical flow methods, optical flow field of the image is calculated, and then based on its optical flow distribution characteristics, the cluster processing is performed. This method has the capability to obtain the complete movement information and to detect the moving object from the background better. Background subtraction in the other hand detects the motion by subtracting the current frame from the background frame. This method is simple yet it is so sensitive to the change in the environment. Finally, the frame subtraction detects the presence of moving objects by calculating the difference between two consecutive images. Motion templates and frame difference algorithms are both examples of such technique. [15]

2.1.1 Motion templates

Motion Templates is a vision technique that is used to detect the movement of objects or obstacles. Thus it is widely used in obstacle detection and autonomous robots project. It was invented by Bobick and Davis in MIT media lab. The most important characteristic of this technique is that the motion can be detected even in small regions of the frame. Motion templates can be implemented using OpenCV library using languages such as C and C++.

The basic requirement of this algorithm is the silhouettes. Silhouettes can be obtained by getting the difference of the two consecutive frames. By using OpenCV library function "cvAbsDiff" the silhouette can be easily obtained. The sequence of silhouettes along with their timestamps construct what is known as Motion History Image (MHI). By taking the gradient of this sequence, the motion in the video can be detected. [15] The main advantage of this algorithm is its capability to show the expected orientation of the motion detected. However, this algorithm works only when the camera is static.

2.1.2 Frame difference

Frame difference is the simplest algorithm for motion detection. The algorithm compare the current frame with a reference frame. One method to implement this

algorithm was proposed by Nishu Singla in his paper motion detection based on frame difference method. The method steps were stated as follows:

1- Difference of Two Consecutive Frames

The absolute difference between two frames is calculated based on Eq. (2):

$$Fd(i, i + 1) = |F(i + 1) - F(i)| \quad (2)$$

Where $i+1$ and i is the value of $i+1$ and i frame respectively.

2- Transforming of absolute differential image to gray scale image

The absolute difference is then transformed to gray Image. This is expected to facilitate the operation since the contours of the motion detected might not be closed and has some holes in them.

3- Filtering and thresholding of transformed gray image

To remove the holes, the difference gray image is filtered using Gauss low pass filter.

2.2. Laser Range Finder

Laser range finder (LRF) is a device that is used to calculate the depth or the distance to certain project.

The basic working principle of LRF is similar to radar or sonar, the only difference is that it uses the laser instead of electromagnetic waves. The LRF emits the laser beam and then measures the time it takes the beam to fly and the then return back and since the speed of the beam is already pre-defined then the distance to the obstacles or the object can be measured. [4]

The advantage of using LRF is that it can detect object for longer distance comparing to radar which makes it suitable for obstacle detection system in cars, however it also has some drawbacks such as: limited view range, and existing of holes sometimes in the scan area due to some part of beams not being reflected. [4]

Lastly, when it comes to process the data that come from LRF, a filtration of the bad returned beams is needed first in order to get the right measurement of object ahead. [4]

2.3 Raspberry pi network

Raspberry Pi is a small computer platform which was designed for children education purpose, but because of the capabilities the Raspberry Pi has, many developers use it nowadays as their embedded platform for their projects. [6]

Raspberry Pi computer is built on an integrated circuit (system on chip) which consists of ARM processes with speed of 700 MHZ and broadcom Videocore graphic processor (GPU) and RAM of size 256/512. Raspberry-Pi has also Hdmi connection and SD card slot for external storage. In term of software, the Raspbian is recommended operating system to run on the Raspberry Pi. This OS is based on Debian linux distribution. [10]

Raspberry Pi is well-known of its low power consumption and small size and its low cost, however it suffers from the low computation power which makes it only suitable for small application, however two or more Raspberry Pies can be combined so this performance problems can be overcome.

When getting to understand how the Raspberry Pies can be combined and communicate between each other some concepts like inter-process communication needs to be defined. Inter-process refers to the activity of distributing or sharing data among many processes using communication protocols. [11] There are many approaches to achieve this such as shared memory, networking, filesystem and using special libraries. In shared memory the operating system maps the same physical memory to many virtual addresses spaces, so if one processors writes to the memory block , this activity will be visible to all others processers. This approach is only viable on processors that supports the shared memory. Other approach is to use networking like sockets and named pipes. But it can be troublesome as numbers of nodes (computers) increases as it will result on coordination and addressing issue. This can be solved by using the last approach which is special libraries or middleware. Special libraries such as Message Passing Interface (MPI) will help reduce the burden of coordination and addressing. [12]

An example of implementing the Raspberry Pi network using MPI can be found in university of Southampton as a super computer has been built from 64 Raspberry Pies connected through switches. The system runs on Raspbian operating system and use the Message Passing Interface or (MPI) on top of the OS to coordinate the 64 nodes and distribute the workload over them. [13]

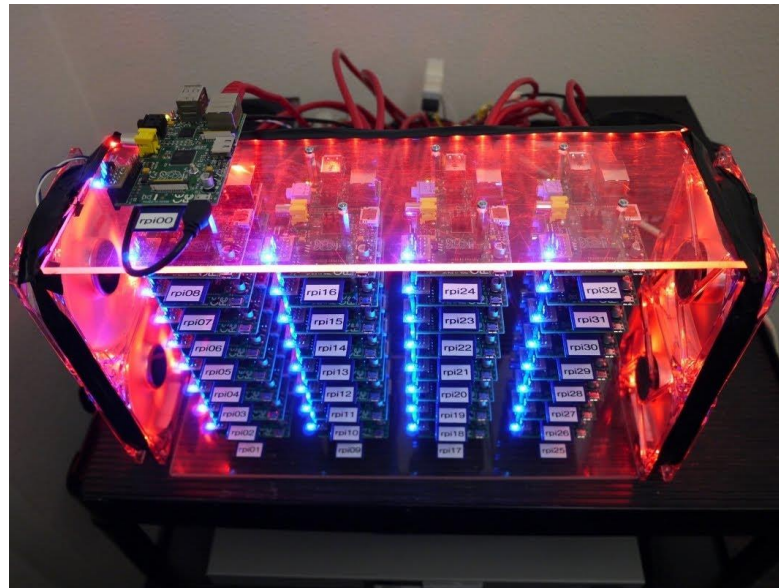


Figure 3: Southampton university Raspberry Pi cluster

Another example of Raspberry Pi cluster was built in Boise state University by a Ph.D. student named Joshua Kiepert. The cluster consists of 33 Raspberry Pies, one raspberry pi works as a master to coordinate the other 32 Raspberry Pies which are responsible for processing the data. The aim of this cluster was to provide him with unlimited access to computer cluster that can do complex processing for his research. This cluster was also built using Linux and MPI. [14]

2.4 Message Passing Interface (MPI):

Message Passing Interface or what is shortly known by MPI is specification for users and software developers who use message passing libraries. It was designed by a group of researchers from academia and industry to provide function on various parallel computers systems. It is not a library but rather a standard or specification of what a library must be. MPI enables the communication between different processors as it handles the movement of data from one address space of one processor to another one via a cooperative operations in every process. [16]

Originally MPI was designed for distributed memories systems, however nowadays MPI can support all types of underlying physical structure of any machine which includes: shared memories, distributed memories and hybrid systems.

There are various well-tested implementation of MPI such as Open MPI library, MPICH. Some of these implementations are open source and free for the public to use

without the need of license. These libraries are supported by many programming languages such as C/C++ and FORTRAN.

Figure 4 shows the general structure of MPI program.

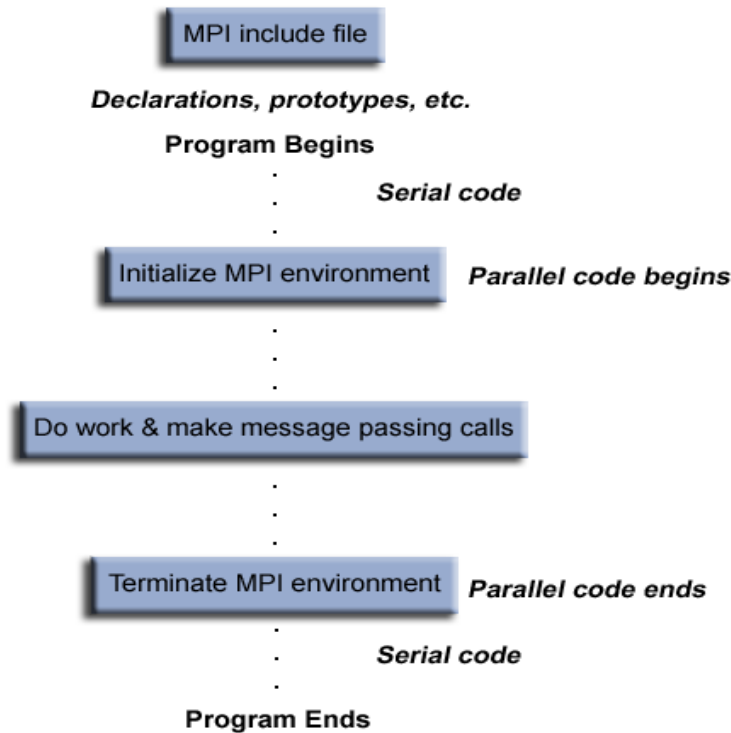


Figure 4: General structure of MPI program

In MPI some important concepts such as communicators and ranks have to be clearly understood in order to develop good MPI program. Communicators in MPI are objects which defines the groups of the processors that are allowed to communicate between each other. Processors in one communicator can exchange data between each other. The default communicator in MPI is called `MPI_COMM_WORLD` which is a pre-defined communicator that includes all the processors in the network. Each processors in one communicator is assigned to a specific integer number or a rank. The system assigns these numbers to each processors when the process initializes. The developers use the rank in order to specify the source and the destination of messages.

The next important concepts is MPI routines. MPI routines can be generally classified into two main groups: The first group include all the MPI environment management routines which are responsible of setting the MPI execution environment and initializing and terminating it. This group include functions such as `MPI_Init` which

initializes the environment and `MPI_Comm_size` which returns the number of processors in specified communicator.

The second group include all the MPI routines which are used for communication. Based on the type of communication, there are two divisions under this group which are: point-to-point routines and collective routines. [16]

The point-to-point routines are used when the communication is only between two processors. On the other hand, the collective routines are used when the communication is between groups of processors. There are three types of collective operation:

- Data Movement which includes broadcast, scatter/gather and all to all.
- Synchronization – The execution of one processor is blocked until all other processors reach to a certain point.
- Collective Computation (reductions) – one processor collects the data from the other group members and performs mathematical operations (min, max, add, multiply, etc.) on that data. [16]

Figure 5 shows some of these collective operations that can be achieved by MPI library routines.

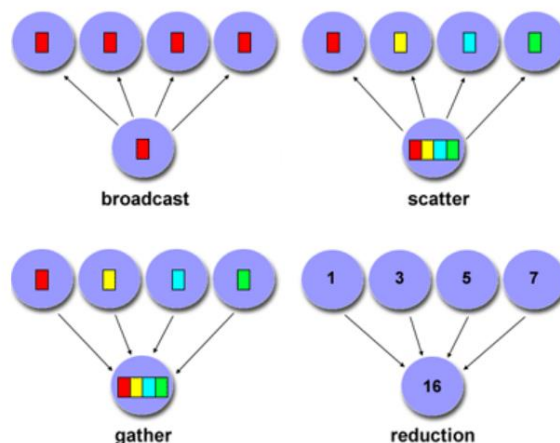


Figure 5: Common collective operations

CHAPTER 3

METHODOLOGY

This section will present the methodology that was used to accomplish this project and the general steps needed to achieve the project goals and objectives.

3.1 Overall methodology

The overall project methodology can be summarized as follows:

i. **Research and Problem Identification**

The first step was doing a research and literature review to understand the project requirement deeply and analyze other projects which have done similar works. Moreover, since this project is a continuation of previous project that has already implemented the object detection algorithms using motion templates technique, a deep understanding of what has been accomplished was needed in order to be able to proceed with this project. This understanding was the ground from which this project evolved.

ii. **Selection of tools and software to be used**

Based on the input from the first stage required tools and software were selected.

iii. **Testing and experimenting the previous project.**

In this stage, the object detection algorithms which has been developed before were tested in order to specify the validity of the data obtained from it, and to understand its functionality and the type of data it produces.

iv. Development phase

In this stage, the project was developed according to the following sequence:

- 1- Setting the network between the three Raspberry Pies computers using switch. The three Raspberry Pies should be able to exchange data between each other.
- 2- Interfacing the Laser Range Finder with Raspberry Pi and obtain the distance measurement data.
- 3- Overall Integration for Laser Range Finder and Camera with the Raspberry Pi network.

v. Testing and optimizing the performance of the system

In this step the overall system was tested to verify the obtained outputs. The network of Raspberry Pies was tested as well to test its overall performance in term of speed and validity of data transferred.

3.2 Project Flow chart

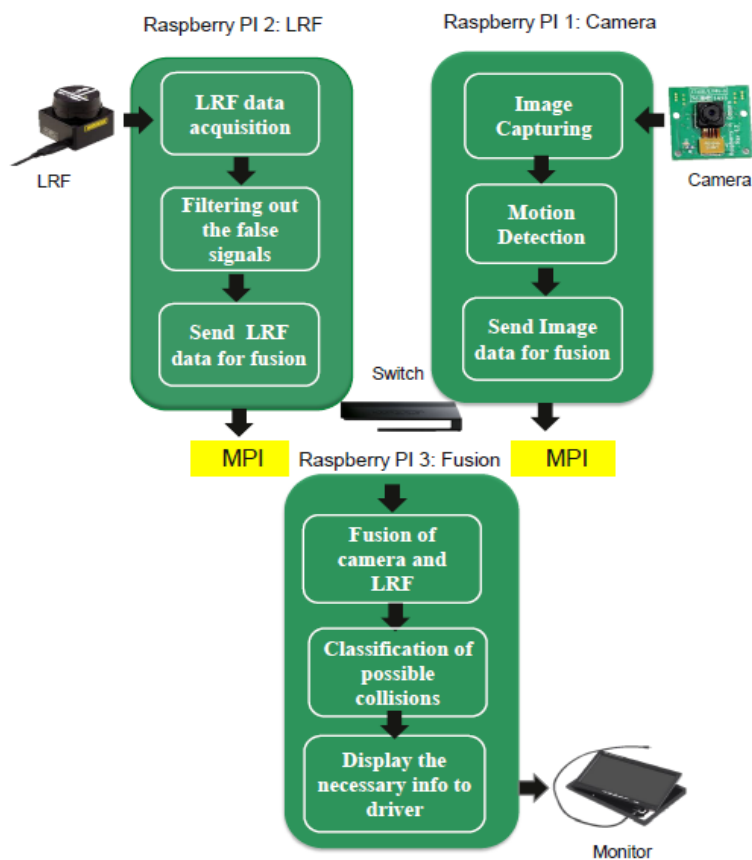


Figure 6: program flow

Step 1: Data Acquisition:

The system starts by capturing the data from the environment. Both the camera and laser range finder read at the same time.

Step 2: Data processing:

The frame data obtained from the camera and the LRF readings are processed independently in two different Raspberry Pies. The images from the camera are processed to detect the motion using two different motion algorithm: motion template and frame difference. The data from the LRF on the other hand are filtered to select only the data that represent the horizontal view of the camera.

Step 3: Sending Data:

Both Raspberry pi will finally send the data to the central Raspberry Pi for fusion using the MPI function "MPI_Send". Since the processing time for each task is different. A delay is introduced in the LRF program in order to synchronize the reading from the LRF and the camera.

Step 4: Data Fusion:

In the central Raspberry Pi and after receiving the processed data from both the LRF and the camera. The fusion of the data takes place. Each reading from LRF represent a specific point in the camera frame, therefore by linking every detected motion with the nearest LRF reading the fusion is achieved.

Step 5: Classification of possible collision:

Based on the reading from the LRF the type of collision is then specified.

Step 4: Display the result to the driver:

Finally the results are displayed to the driver. The image shows both the position of the motion in the camera view and distance of that object from the car.

3.3 Motion detection algorithms

Two motions algorithms were implemented. These algorithms are motion templates and frame difference. They were implemented on specific region of the image instead of the full image. This because we were only interested on the area of the image that the Laser range finder could cover.

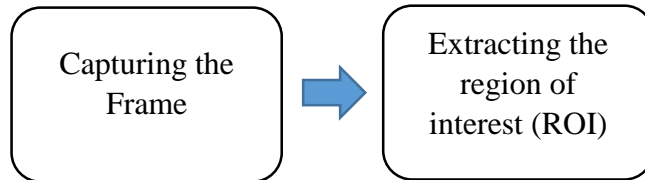


Figure 7: First two steps in image processing

The following diagrams show the methodology of executing Motion Template and Frame Difference in the ROI.

3.3.2 Frame Difference algorithm

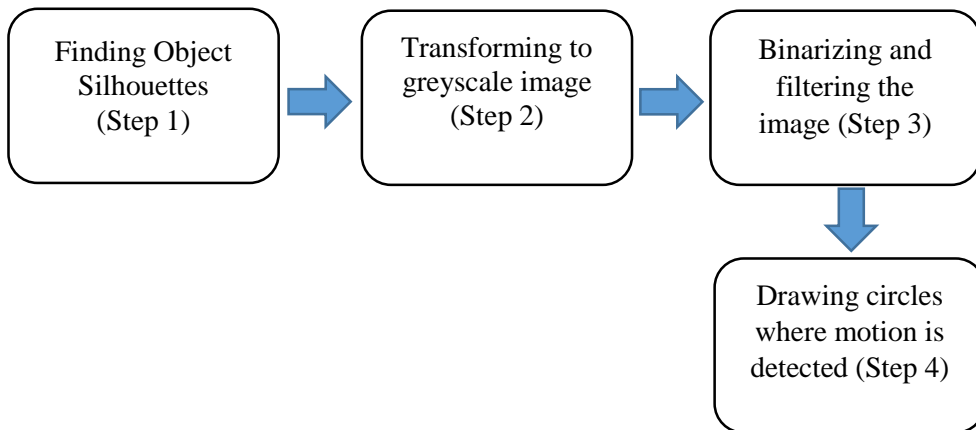


Figure 8: Steps of frame difference algorithms

Step 1:

The object silhouettes are obtained by subtracting the two consecutive frames using the OpenCV function "cvAbsDiff".

Step 2:

The obtained difference image is then transformed into greyscale image to facilitate better operation.

Step 3:

The obtained image from step two is filtered using Gaussian low pass and then thresholded to eliminate the small changes or motion in the scene.

Step 4:

After specifying position of the motion in the current frame. A circle is drawn to show the presence of the motion in that position.

3.3.1 Motion Template

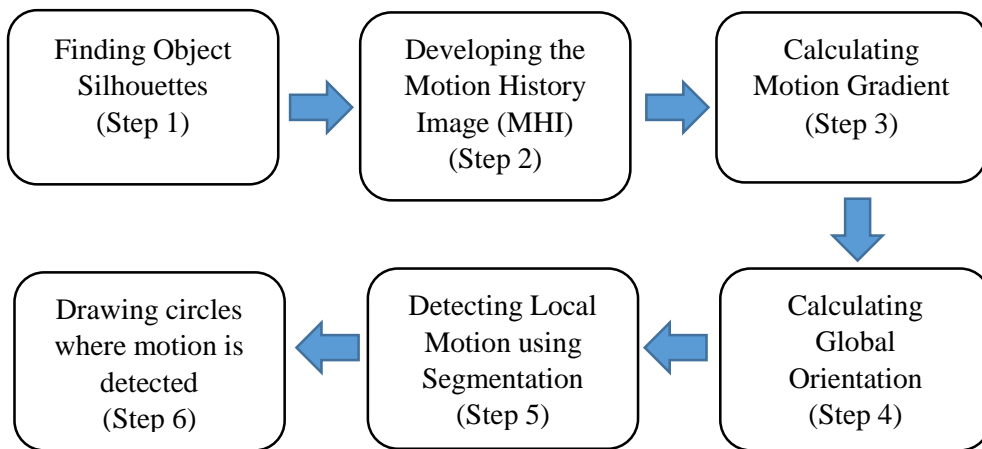


Figure 9: Steps of motion template algorithm

Step 1:

The object silhouettes are obtained by subtracting the two consecutive frames using the OpenCV function `cvAbsDiff` then the output is converted to binary image by using the function `cvThreshold`.

Step 2:

The sequence of the silhouettes that is comprised of the current silhouette and the past ones along with their timestamps (MHI) is then developed. The OpenCV function that is used for this is "`cvUpdateMotionHistory`"

Step 3:

For motion detection, the MHI gradient is then calculated using the function "`cvCalcMotionGradient`".

Step 4:

The motion orientation is generally divided into global and local. Global motion orientation can then be calculated using the function "cvCalcGlobalOrientation". The function takes the output of the "cvCalcMotionGradient" to obtain the required result.

Step 5:

The local motion can be calculated within a region of interest in the image. This can be achieved "cvSegmentMotion" function from OpenCv library. The output is the segmentations of motion. The orientation of each local motion is then calculated using the "cvCalcGlobalOrientation" function.

Step 6:

After specifying the position and the orientation of the motion in the current frame. A circle is drawn to show the presence of the motion in that position.

3.4 Tools and software

The tools and software which were used are listed as follows:

- 1- Three Raspberry pi boards.
- 2- Pi camera.
- 3- Network Switch.
- 4- URG Laser Range Finder.
- 5- URG C library
- 6- OpenCV library.
- 7- Message Passing Interface library (MPICH).

3.5 Key milestones

The development of the project went through some important milestones. These milestones can be summarized as follows:

- Interfacing the LRF with RPI and acquire the necessary measurement data.
- Building the communication network between the three Raspberry Pies.

- Integrating the LRF system and video camera system with the Raspberry Pies network.

3.6 Gant Chart

The following table shows the schedule of the project development for FYP1.

| Task/week | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|-----------|---|---|---|---|---|---|---|---|---|----|----|----|----|----|
| T1 | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ |
| T2 | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | | | | | | |
| T3 | | | | | | ■ | ■ | ■ | | | | | | |
| T4 | | | | | ■ | ■ | ■ | ■ | | | | | | |
| T5 | | | ■ | ■ | ■ | ■ | ■ | | | | | | | |
| T6 | | | | | | ● | | | | | | | | |
| T7 | | | | | | | ■ | ■ | | | | | | |
| T8 | | | | | | | | | ● | | | | | |
| T9 | | | | | | | | | | ■ | ■ | ■ | ■ | |
| T10 | | | | | | | | | | ■ | ■ | ■ | ■ | ● |

Tasks:

- **T1:** Preliminary research and literature review
- **T2:** Learning the necessary tools and software required for the project
- **T3:** Testing and running the object detection code (previous work).
- **T4:** Build the Raspberry pi network and run simple program on it.
- **T5:** Write extended proposal
- **T6:** Submission of extended proposal
- **T7:** Preparation for proposal defense presentation
- **T8:** proposal defense presentation
- **T9:** Interim report Draft
- **T10:** Interim report

FYP-2 Gant chart

| Task/week | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|-----------|---|---|---|---|---|---|---|---|---|----|----|----|----|----|
| T1 | ■ | ■ | ■ | | | | | | | | | | | |
| T2 | | | | ■ | ■ | ■ | | | | | | | | |
| T3 | | | | | | | ■ | ■ | ■ | ■ | ■ | ■ | | |
| T4 | | | | | | | | | | ● | | | | |
| T5 | | | ■ | ■ | ■ | ■ | ● | | | | | | | |
| T6 | | | | | | | | | | | | ● | | |
| T7 | | | | | | | | | | | | ● | | |
| T8 | | | | | | | | ■ | ■ | ■ | ■ | ■ | | |
| T9 | | | | | | | | | | | | | ● | |
| T10 | | | | | | | | | | | ● | | | |

● : Important milestones

Tasks:

- **T1:** Setup the interface between the RPi and LRF
- **T2:** Integrate the three systems together.
- **T3:** Testing and benchmarking the system.
- **T4:** Pre-SEDEX
- **T5:** Preparation and Submission of Progress Report
- **T6:** Submission of Dissertation (soft bound)
- **T7:** Submission of Technical Paper
- **T8:** Preparation of final Viva
- **T9:** Viva
- **T10:** Submission of draft final report

CHAPTER 4

RESULTS and DISCUSSION

This chapter is designed to present the findings of this project. It is divided into six main sections. The first section shows the result of building the Raspberry Pi network. The second section shows the result of interfacing the Raspberry Pi with the LRF, this is followed by the third section which shows the results obtained from interfacing the Raspberry Pi with the camera and performing two different motion detection algorithms. After that, the core section is presented which shows the fusion of the LRF data and camera results. Finally an analysis on the network performance is presented in the last two sections.

4.1 Raspberry Pi network:

The network of Raspberry Pies was built using three Raspberry Pies, the three Raspberry Pi boards were connected through network switch. The following figure shows the physical setup of the network.

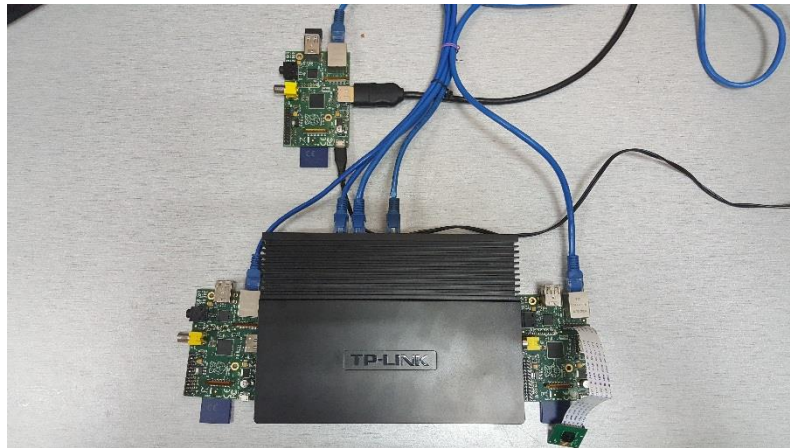


Figure 10: Raspberry Pi network connection

In order to enable the communication between the Raspberry Pi processes, MPICH library was installed. This library implements the Messaging Passing Interface (MPI) standard to provide communication functionality between the three Raspberry Pi processes.

Each Raspberry Pi runs independent MPI programs that is designed to do specific task (e.g. distance measurement using LRF). These programs contain initially the necessary MPI functions such as MPI_Init that allow the initialization and starting of the MPI execution environment in every process.

Whenever the three programs launch, the default communicator (MPI_COMM_WORLD) is used and every Raspberry Pi processor is assigned to a certain rank by MPI as Fig. 11 shows. The importance of such setting is to allow the future communication between the Raspberry Pi processors as the processor rank is crucial in determining the source and the destination of the message.

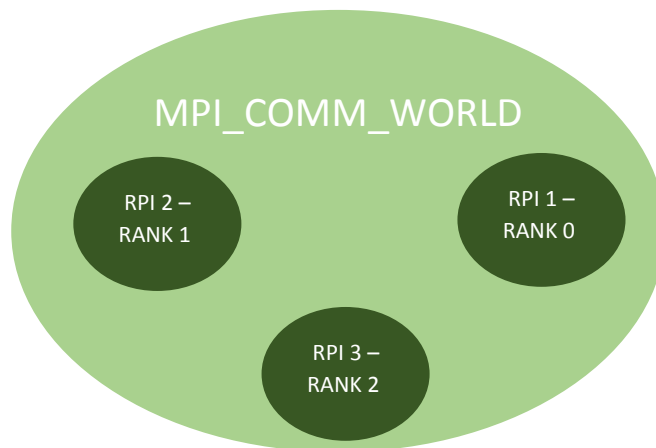


Figure 11: Illustration of MPI communicator

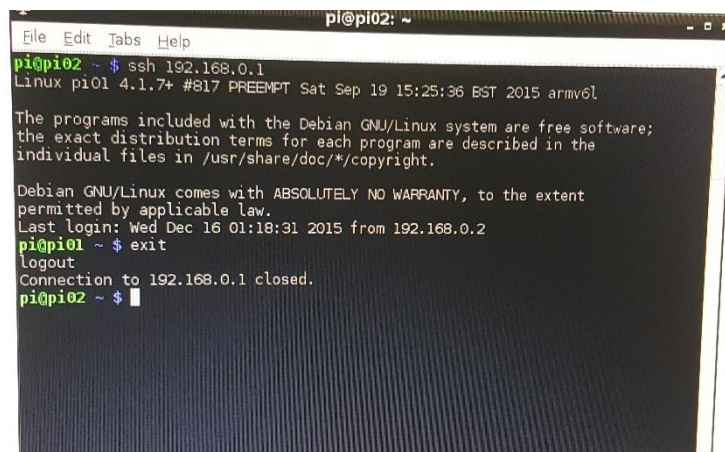


Figure 12: Accessing the network nodes using SSH program

4.2 Laser range finder (LRF) interface with first Raspberry Pi:

Laser range finder was interfaced with the Raspberry Pi and its readings were obtained. LRF scans the environment with angle of 240 degree. It gives 682 readings with angle resolution 0.36 degree and distance coverage up to 5 meters. Before connecting the LRF with Raspberry Pi, the sensor was tested using the URG viewer application in windows platform. The yellow line in Figure 5 shows what the laser range finder detects.

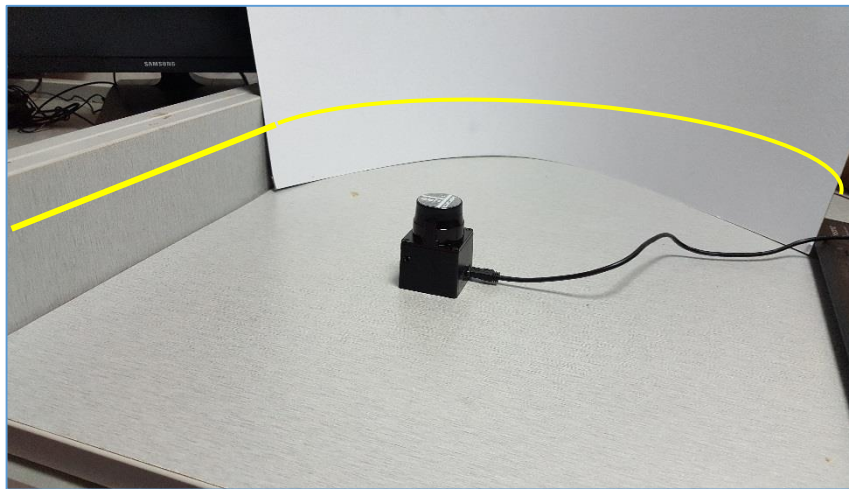


Figure 13: Testing the LRF with viewer Application.

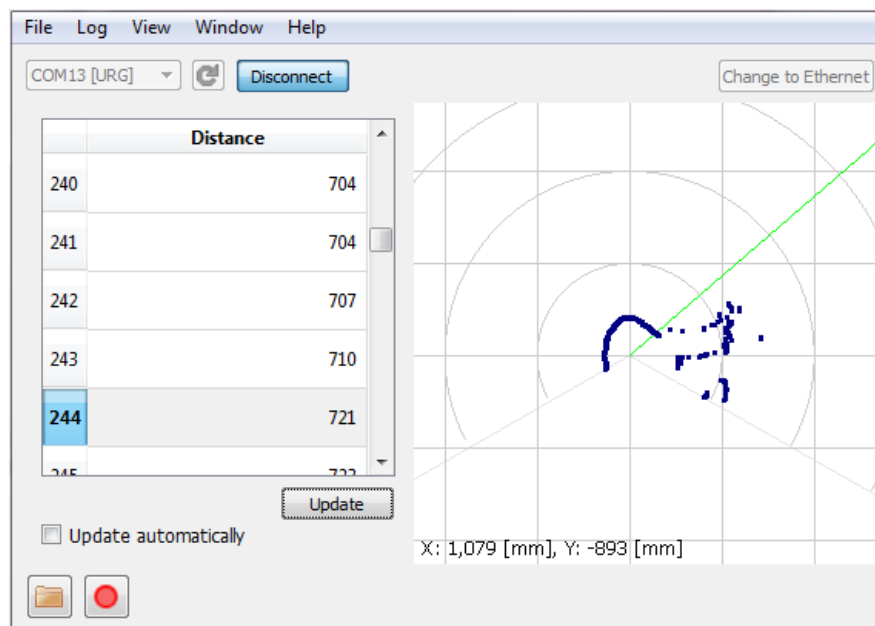


Figure 14: Viewer application displays the data of LRF

In linux environment a C program was written using the URG library to perform the distance reading.

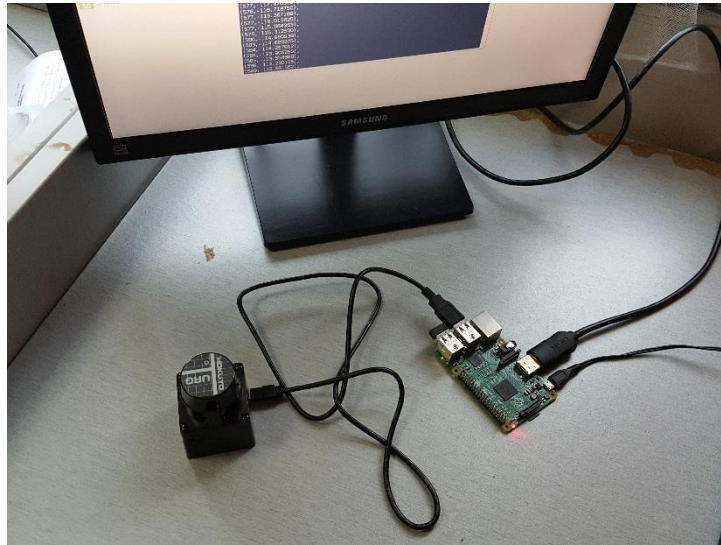


Figure 15: LRF connection with Raspberry PI

4.3 Camera interface with the second Raspberry Pi for obstacles movement detection:

To detect the motion of objects and obstacles, two different motion detection algorithms were used: frame difference and motion templates.

4.3.1 Frame difference

Frame difference algorithm was implemented in Raspberry Pi using OpenCv library. The motion is detected by comparing two consecutive frames and taking the absolute difference between the two frames using OpenCv function "absDiff" and then a rectangle is drawn where the motion is detected. Since the LRF can only detect one dimension from the camera image (one row of the frame), a region of interest (ROI) was created in every input frame. This region includes the area that the LRF detects. Processing only this region reduces the total processing time and the computational power needed by Raspberry Pi processor. The following figures show the result obtained using this method.



Figure 16: The moving object was detected



Figure 17: The moving object was detected 2



Figure 18: The moving object was detected 3

Frame difference algorithms is the simplest algorithm for motion detection, the result obtained is satisfactory to some extent, however, the algorithms is highly affected by the change in the lighting condition. Therefore some parameters like the color threshold has to always be adjusted according to the light condition. Moreover, this algorithm is only suitable when the camera is static. However, in real life scenario the algorithm should be able to detect the motion while the car is moving. This requires new algorithms to be studied.

In addition, the algorithm sometimes gives false result as the same object can be detected as multiple objects as Fig. 19 shows:

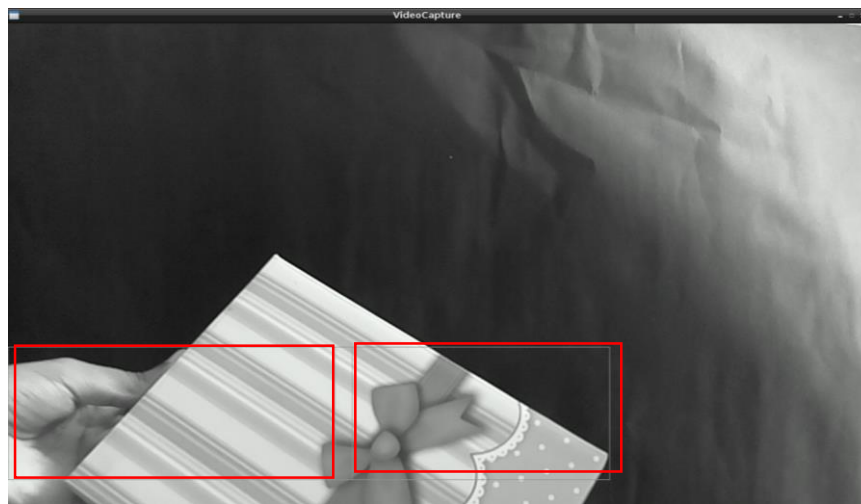


Figure 19: multiple motion detections for the same object

4.3.2 Motion Templates

Motion templates algorithm was implemented in Raspberry Pi. But instead of comparing only two frames, this method relies on the frames history by taking the gradient of the sequence of multiple recent preceding images. Same like frame difference, this algorithm was implements only on specific region on the frame. The result obtained are as follows:



Figure 20: Motion template result (colored)

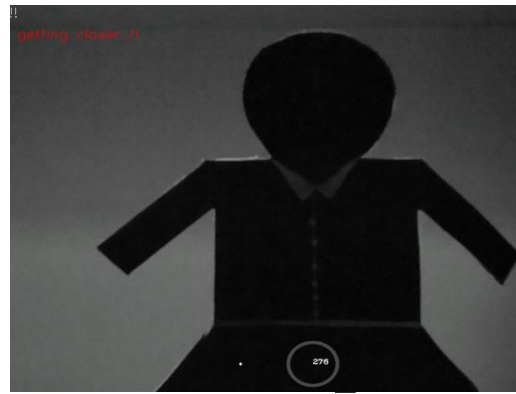


Figure 21: Motion template result (gray scale)

The circles in both images represent the position where the motion has been detected.

In comparison, the result obtained from this method was found to be more reliable than the frame difference. But, in term of processing time that is needed to execute those two algorithm, there was no significant difference.

4.4 Data fusion of LRF data and camera in central Raspberry Pi:

After getting and processing the readings from both the camera and LRF independently as shown in previous sections, the camera and LRF data are now ready for fusion. To successfully project the LRF data onto the image or frame data, the physical pose of both camera and LRF had to be adjusted. Fig. 22 shows the best obtained pose. The centers of the camera and LRF were aligned and the LRF was positioned lower to the camera, therefore the region of interest was chosen to be the lower quarter of the full frame.

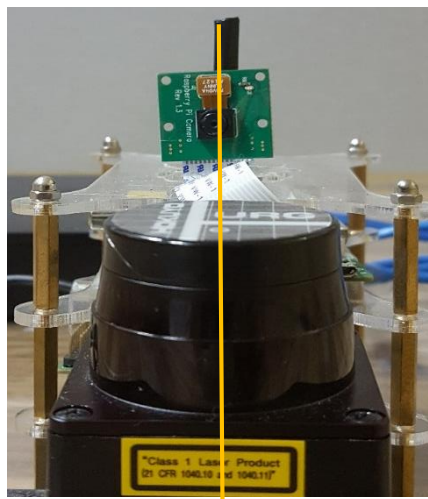


Figure 22: Camera and LRF centres alignment

The motion detection and distance measurement programs in the two Raspberry Pis were then modified to allow the network communication using MPI library. In addition to use the MPI environment management routines which are responsible for initializing the MPI execution environment, MPI_Send and MPI_Recv were used to allow sending the image and LRF data between the three Raspberry Pis.

The first Raspberry Pi implements the motion detection algorithm in each frame and then finally sends it to the central Raspberry Pi. The second Raspberry Pi obtains the reading from the LRF. The position of the LRF was calibrated so its readings represent the size of the camera horizontal field of view. It was found that out of the 682 LRF reading, approximately only 208 values are sufficient to represent the horizontal view of the camera. With the current camera and LRF physical positioning the 208 LRF values that are needed range from index 246 to 454.



Figure 23: LRF and camera setup

As aforementioned, the system was tested under two different algorithms, the following sections show the results of data fusion under two situations:

4.4.1 Motion Templates:

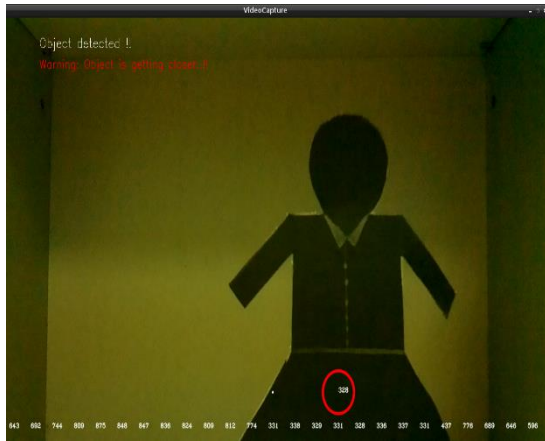


Figure 24: Data fusion result (colored)



Figure 25: Data fusion result (gray scale)

4.4.2 Frame Difference:



Figure 26: Data fusion result (gray scale)



Figure 27: Data fusion result (colored)

The rectangle and circle shapes indicate that the location of the motion, while the values inside indicate the distance from the object to the car.

In Fig. 28 the motion was detected and the LRF reading at that area shows 375 mm.

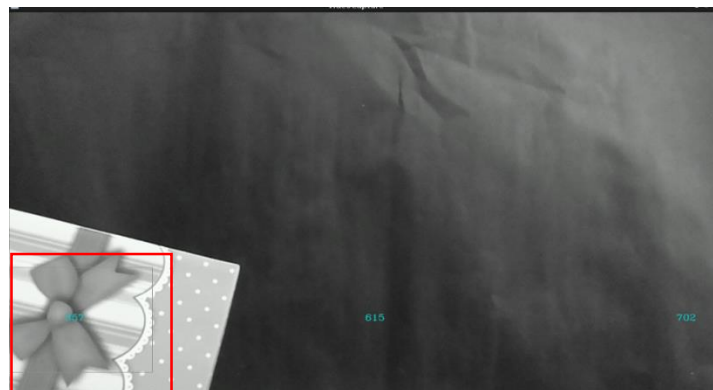


Figure 28: Data fusion result

In Fig. 29 the object has moved and the motion was detected and the distance was recorded to be 383 mm.



Figure 29: Data fusion result

4.5 Network Analysis:

In this section the network analysis is conducted. The purpose of the analysis is to evaluate the network performance under different scenarios in order to find the optimum scenario under which the system is more reliable and closer to a real-time one.

The most important factor which affects the system is the delay that is caused by the network. To investigate this issue, the network was analyzed by capturing the network packets for five minutes using the program "tcpDump". Tcpdump is a free packet analyzer software that displays the IP/TCP and other network packets that being transmitted or received over the network which the computer is being attached to. The investigation took place in four different conditions. In each condition, either the motion detection algorithm on the first Raspberry Pi was changed or the size of frame being sent was changed. The four conditions are listed in the table below:

Table 1: Shows the four conditions under which the network were tested

| | |
|-------------|--|
| Condition 1 | Using motion template algorithm and sending colored frame |
| Condition 2 | Using motion template algorithm and sending greyscale frame |
| Condition 3 | Using frame difference algorithm and sending colored frame |
| Condition 4 | Using frame difference algorithm and sending greyscale frame |

The results from this program were then fed to "Wireshark" program which provides information such as the network speed and the amount of data transferred.

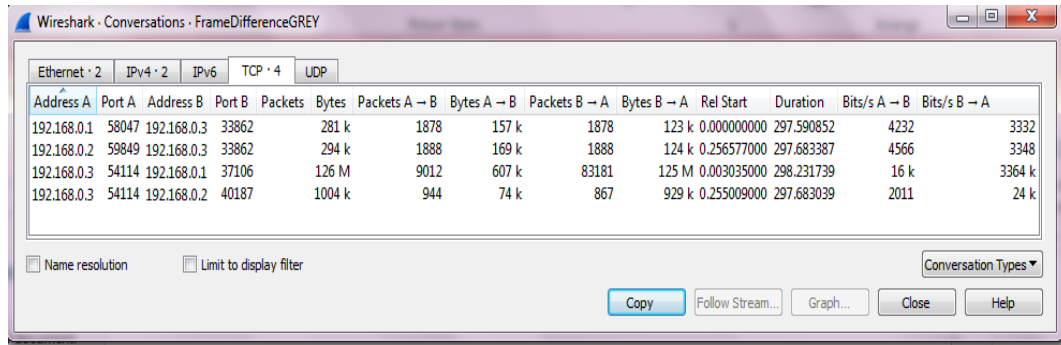


Figure 30: Sample of Wireshark network analysis result (Frame difference)

The delay in the network based on the Wireshark results was found to be as follows:

Table 2 Shows the network time needed to transfer one frame when different motion algorithms were used.

Table 2: Characteristics of RPI 1 (Camera) and RPI 3 network interface

| Motion detection algorithm | Data transferred per frame(byte) | Network speed (bit/s) | Network Delay (s) |
|-------------------------------|----------------------------------|-----------------------|-------------------|
| motion template (Grey Scale) | 1,228,800 | 4.5 M | 2.2 |
| motion template (RGB) | 3,686,400 | 6.7 M | 4.4 |
| Frame difference (Grey Scale) | 1,228,800 | 3.4M | 2.8 |
| Frame difference (RGB) | 3,686,400 | 5.9M | 4.9 |

Table 3: Characteristics of RPI 2 (LRF) and RPI 3 network interface

| Motion detection algorithm | Data transferred per LRF array (byte) | Network speed (bit/s) | Network Delay (s) |
|-------------------------------|---------------------------------------|-----------------------|-------------------|
| motion template (Grey Scale) | 5,464 | 27 k | 0.2 |
| motion template (RGB) | 5,464 | 17k | 0.32 |
| Frame difference (Grey Scale) | 5,464 | 24k | 0.23 |
| Frame difference (RGB) | 5,464 | 17k | 0.32 |

From table 2 above it can be observed that the least delay that could be achieved was when the motion template algorithm was used and the video frames were sent as

greyscale image rather than colored frame (2.2 s). The reduction of the data which are needed to be transferred between camera Raspberry Pi and the central Raspberry Pi have reduced the delay to almost the half.

From table 3 it is observed that the LRF data which are needed to be transferred in the network are quite few compared to the camera data. Therefore, the delay caused can be negligible.

Another factor that affects the network performance beside the network delay was the desynchronization. Obtaining and processing the LRF data needs less computational power compared to the image processing on the camera RPI, this uneven distribution of processing load causes synchronization problem in the network which leads to some reading of LRF to be fused with the wrong camera frame. The main cause of this problem is that in MPI, when certain processor sends some message data to other processor using "MPI_Send" function, there is no mechanism to ensure that the message has been successfully received by the other processor. The sender can keep sending data and these data are stored in the receiver buffer and when the receiver processor reaches the receiving statement "MPI_Receive" it reads from that buffer which can include either too old or too new distance measurements data.

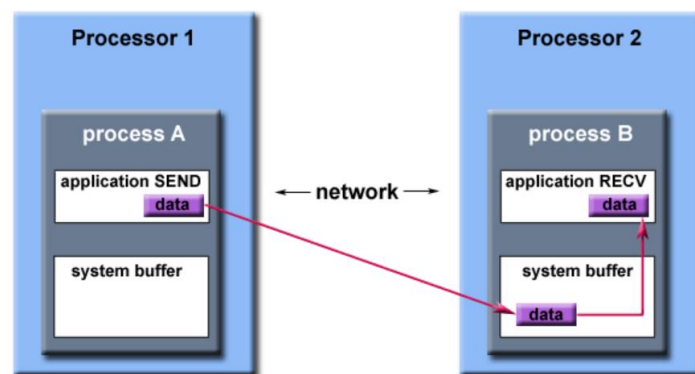


Figure 31: MPI buffer concept

To solve this issue, a deliberate delay was introduced to the LRF program to delay or defer the execution of the instructions after the MPI_Send command. So the LRF process does not read and send distance data for fusion unless the camera process is done and also ready to capture image from the environment. The value of the delay increases based on the complexity of the motion detection algorithm in the other side

and the time required to complete the data transferring between the camera RPI and the central or fusion RPI.

4.6 New network topology

From the previous section, it was observed that the main source of network delay was caused by transferring the frame data from the camera Raspberry Pi to the central Raspberry Pi and the minimum delay achieved was 2.2 seconds when only a greyscale frames are transferred and the motion template algorithm was used. 2.2 seconds delay is relatively small however, in this application the instant results that reflect the environment are highly required. The driver should obtain the information about the imminent crash as fast as it occurs. Therefore, in order to optimize the network performance and enhance the system ability to become a real time one, a new network topology was introduced. The network contains only two Raspberry Pis. One for LRF and the other for the image processing but instead of sending the frames data over the network for fusion, the fusion takes place in the same Raspberry Pi dedicated for the image processing task.

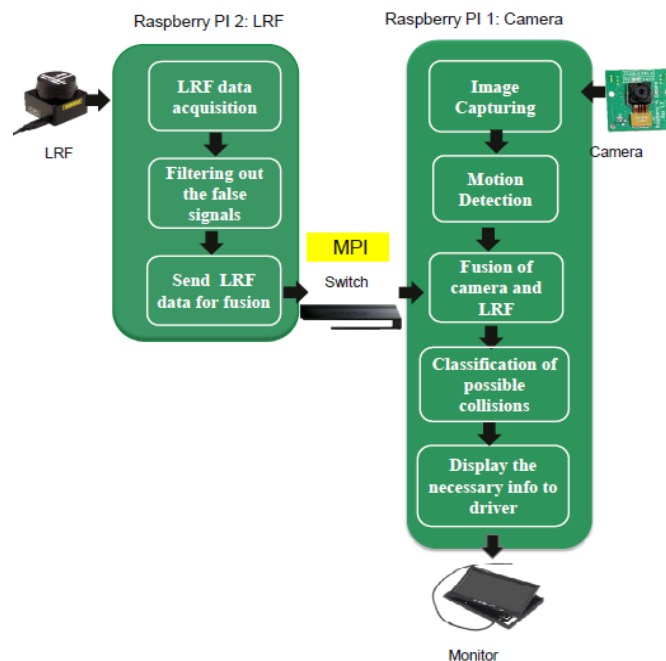


Figure 32: New proposed network topology

The time from the image capture to display was then measured by reading the system time before the program starts capturing the image and after displaying it to specify the delay caused in such network. Under all the four conditions described in previous

section, it was found that only one second delay is the price in such network topology.

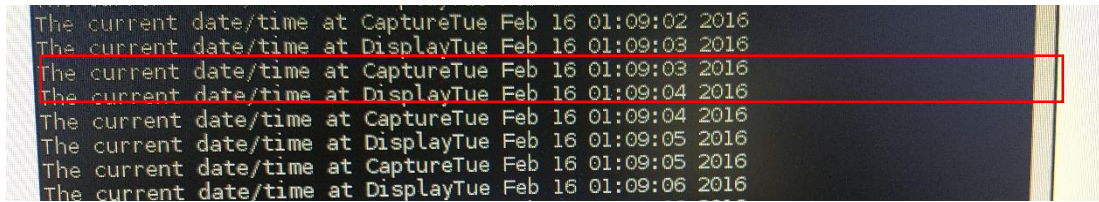
A terminal window screenshot showing a sequence of log messages. The messages are: 'The current date/time at CaptureTue Feb 16 01:09:02 2016', 'The current date/time at DisplayTue Feb 16 01:09:03 2016', 'The current date/time at CaptureTue Feb 16 01:09:03 2016', 'The current date/time at DisplayTue Feb 16 01:09:04 2016', 'The current date/time at CaptureTue Feb 16 01:09:04 2016', 'The current date/time at DisplayTue Feb 16 01:09:05 2016', 'The current date/time at CaptureTue Feb 16 01:09:05 2016', and 'The current date/time at DisplayTue Feb 16 01:09:06 2016'. A red rectangular box highlights the two lines: 'The current date/time at DisplayTue Feb 16 01:09:03 2016' and 'The current date/time at DisplayTue Feb 16 01:09:04 2016', indicating a one-second interval between the first and second display events.

Figure 33: Measuring the capture to display delay

The delay of one second compared to 2.2 second achieved in the previous network topology has significantly improved the system performance to more than 50%. The new network topology yields in delay reduction to almost half of the delay caused by the previous network. Therefore, this network is highly recommended to be adopted in such system as it produces better result in all conditions.

In summary, the fusing of LRF data and camera in the central RPI results are quite satisfactory when the motion template algorithm is used and the data transferred is reduced to the third by transferring the greyscale image instead of colored one. Reducing the image quality and resolution is expected to improve the performance as well and make it closer to the real-time application one. Lastly, a new network structure was introduced. It consisted of two Raspberry Pies instead of three as the tasks of image processing, the fusion of the data and the results displaying were executed by only one Raspberry Pi. This structure eliminated the delay caused by the need of transferring the camera frames over the network and has enhanced the overall performance by more than 50%.

CHAPTER 5

CONCLUSION AND RECOMMENDATION

5.1 CONCLUSION

Vehicle obstacle avoidance system is necessary to maintain the safety of drivers on the road. Such system is usually implemented using one sensor such as camera, laser range finder, ultrasonic and stereo camera to detect any possible threat on the road, however the data obtained from one sensor is not enough to formulate a complete alerting and notifying system for the driver. Therefore, in this report a system that utilizes both camera and LRF sensors to detect the position and the depth of static obstacles was presented. The system was built using Raspberry Pi as embedded platform. Raspberry Pi lacks the computational power that is needed to support such system. To overcome this problem a network of three Raspberry Pies was used. The data from LRF and camera were processed independently and in parallel in two Raspberry Pi boards, and then the data were finally fused in central Raspberry Pi board. The Raspberry Pies network was built using MPICH library. Two algorithms have been implemented for motion detection which are motion template and frame difference and from the result obtained, it was found that the result of motion template is more reliable than the frame difference algorithm's.

Moreover, the data of LRF and camera were successfully fused in the central RPI as shown in chapter 5. The delay problem in the network was minimized by reducing the quality of the images and by processing only specific region of the frame. Analyzing the network has shown that most of the delay is due to transferring large size image over the network, this problem was alleviated by reducing the size of frame transferred and by introducing new network topology where the camera frames do not need to be transferred.

5.2 RECOMMENDATION

The future work should focus on developing the final prototype and optimizing the image processing part in order to minimize the false reading that sometimes occur. Moreover, an algorithm that detects the motion while the camera is moving should be developed in order for the system to be applicable in real life.

REFERENCES

- [1] World Health organization , "Road traffic injuries," October 2015. [Online]. Available: <http://www.who.int/mediacentre/factsheets/fs358/en/>. [Accessed 23 December 2015].
- [2] "Distracted driving," [Online]. Available: <http://distracteddriving.caa.ca/education/>. [Accessed 23 December 2015].
- [3] Y. Saito, T. Azumiy, N. Nishio and S. Kato, "Sensors Fusion of a Camera and a LRF on GPU for obstacles avoidance".
- [4] S. J. Cacciola, "Fusion of Laser Range-Finding and Computer Vision Data," 2007.
- [5] M. Yeong, "Video object avoidance implementaion of embedded platform," Tronoh , 2015.
- [6] P. Abrahamsson, S. Helmer, N. Phaphoom and L. Nicolodi, "Affordable and Energy-Efficient Cloud Computing Clusters: The Bolzano Raspberry Pi cloud cluster experiment," in *IEEE International Conference on Cloud Computing Technology and Science*, 2013.
- [7] RaspberryPi , "camera," RaspberryPi , [Online]. Available: <https://www.raspberrypi.org/documentation/hardware/camera.md>. [Accessed 24 December 2015].
- [8] S. Garethiya, L. Ujjainiya and V. Dudhwadkar, "PREDICTIVE VEHICLE COLLISION AVOIDANCE SYSTEM USING RASPBERRY - PI," *ARPN Journal of Engineering and Applied Sciences*, vol. 10, no. 8, 2015.
- [9] OpenCV, OpenCV, [Online]. Available: <http://opencv.org/>. [Accessed October 2015].
- [10] wikipedia.org, "Raspberry_Pi," [Online]. Available: https://en.wikipedia.org/wiki/Raspberry_Pi. [Accessed October 2015].
- [11] Wikipedia , "Inte process communication," [Online]. Available: https://en.wikipedia.org/wiki/Inter-process_communication. [Accessed October 2015].
- [12] H. Iliev, "Introduction to MPI Programming, part 1," March 2014. [Online]. Available: https://www.youtube.com/watch?v=LBgx_S5ougk. [Accessed December 2015].
- [13] Southhampton university, "RaspberryPi at southhampton," [Online]. Available: <http://www.southampton.ac.uk/~sjc/raspberrypi/>. . [Accessed October 2015].
- [14] Boise state university, "Creating a Raspberry Pi-Based Beowulf Cluster," 22 October 2013. [Online]. Available: http://coen.boisestate.edu/ece/files/2013/05/Creating.a.Raspberry.Pi-Based.Beowulf.Cluster_v2.pdf. [Accessed 23 December 2015].
- [15] N. Singla, "Motion Detection Based on frame difference method," *International Journal of Information & Computation Technology*, vol. 4, no. 15, pp. 1559-1565, 2014.
- [16] B. Barney, "Message Passing Interface (MPI)," Lawrence Livermore National Laboratory, 17 6 2015. [Online]. Available: <https://computing.llnl.gov/tutorials/mpi/>. [Accessed 17 4 2016].

APPENDICES

1. Central Raspberry Pi code:

```
#include "opencv2/video/tracking.hpp"
#include "opencv2/imgproc/imgproc_c.h"
#include "opencv2/imgproc/imgproc.hpp"
#include <opencv2/opencv.hpp>
#include "opencv2/highgui/highgui.hpp"
#include <time.h>
#include <stdio.h>
#include <ctype.h>
#include <math.h>
#include <raspicam/raspicam_cv.h>
#include <raspicam/raspicam.h>
#include <string>
#include <sstream>
#include <iostream>
#include <ctime>
#include <cstdlib>
#include <fstream>

// To help me convert from long value to string
namespace patch
{
template<typename T> std::string to_string(const T& n)
{
std::ostringstream stm;
stm<<n;
return stm.str();
}
}

/*****MPI*****/
#include<mpi.h>
/*****/

using namespace std;
using namespace cv;

int i =1;
int z;

int main(int argc, char** argv)
{

/*****MPI-intitialization*****/
MPI_Init (NULL, NULL) ;
int world_size;
MPI_Comm_size(MPI_COMM_WORLD, &world_size);
int world_rank;
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

char processor_name[MPI_MAX_PROCESSOR_NAME];
```

```

int len;

MPI_Get_processor_name(processor_name, &len);

/*****DONE*****/

//std::cout<<"Iam processor : "<<processor name<< " rank:
"<<world_rank<<std::endl;

    namedWindow("VideoCapture", CV_WINDOW_AUTOSIZE);
    Mat img = Mat::zeros(960,1280,CV_8UC3);

Mat img2;
//cv::cvtColor(img,img2,cv::COLOR_BGR2GRAY,3);

    int imageSize = img.total()*img.elemSize();
    uchar Data[imageSize];
    long lrfData[682];
    int XY[50];
    long oldData[682] = {0};
    int check [1] = {0};
    int XYcount;
    MPI_Status status;
    MPI_Status lrfStatus;

    MPI_Status XYstatus;

while(1){

    MPI_Recv(lrfData,682,MPI_LONG,1,1,MPI_COMM_WORLD,&lrfStatus);

std::cout<<"lrData[341] : "<<lrfData[341]<<" "<<std::endl;

MPI_Recv(Data,imageSize,MPI_UNSIGNED_CHAR,0,0,MPI_COMM_WORLD,&status
);

std::cout<<" Image data received !! "<<std::endl;

MPI_Recv(XY,50,MPI_INT,0,3,MPI_COMM_WORLD,&XYstatus);

    MPI_Get_count(&XYstatus,MPI_INT,&XYcount);

/*
    for(l=2;l<=XYcount ;l=l+2)
        {
            std::cout<< XY[l]<<" ";
        }
*/

int ptr=0;
int k = 0;
for(int i = 0; i< img.rows;i++) {

```

```

    for(int j=0; j<img.cols;j++) {

//img.at<cv::Vec3b>(i,j)=cv::Vec3b(Data[0+ptr],Data[1+ptr],Data[2+ptr]);

img.at<cv::Vec3b>(i,j)=cv::Vec3b(Data[0+ptr],Data[0+ptr],Data[0+ptr]);
    // ptr=ptr+3; // for RGB
    ptr++;

}
}

int x;
k=0;
int count = 0;
int l = 0 ;
    for(x=246;x<454;x=x+8)
    {
        putText(img,patch::to_string(lrfData[x]),cvPoint(1280-k,690),FONT_HERSHEY_COMPLEX_SMALL,0.6,cvScalar(255,255,255),1,CV_AA);
        ;
        if((oldData[x] - lrfData[x]) > 20  && count == 0 )
        {
            putText(img,"Warning: Object is getting closer...!",cvPoint(100,85),FONT_HERSHEY_SIMPLEX,0.7,cvScalar(0,0,255),1,CV_AA);
            ;
            count =2;
        }

k=k+50;
    }

std::cout<<"FROM RPI3 THE SIZE OF MATRIX IS: "<<XYcount<<std::endl;
k =0;
    for(x=246;x<454;x=x+8)
    {

        for(l=2;l<XYcount ;l=l+2)
        {
            if ( (XY[l]<= 1280-k) && (XY[l] > 1280-k-50))

putText(img,patch::to_string(lrfData[x]),cvPoint(XY[l],XY[l+1]),FONT_HERSHEY_COMPLEX_SMALL,0.6,cvScalar(255,255,255),1,CV_AA);
            // std::cout<< XY[l]<<"\n\n\n";
        }
        k= k+50;
    }

std::cout<<l<<std::endl;

IplImage imagea = img; // transform from Mat type to IplImage
IplImage* Image = &imagea;

```

```

    if (&imagea){

    if (!Image)
        break;

        cvShowImage("VideoCapture", Image); // The first parameter is
the the window name and the 2nd param. is pointer to IplImage image
// (address of the image that
was captured by camera)
        if(waitKey(10) >= 0)
            break;

//Update LRF values.
        for( z = 246 ;z < 454 ;z++)
            { oldData[z] = lrfData[z];

                }

    }

/***** For synchronization*****/
MPI_Send(check,1,MPI_INT,1,10,MPI_COMM_WORLD);
/*****/

}

cvDestroyWindow("VideoCapture");

MPI_Finalize();

    return 0;
}

```


2. Distance measurements using URG library in Raspberry Pi 2 code:

```
#include<unistd.h>
#include <stdio.h>
#include<mpi.h>

/*****LRF*****/
#include "urg_sensor.h"
#include "urg_utils.h"
#include "open_urg_sensor.h"
#include <stdlib.h>
/*****/

int main(int argc, char** argv)
{
printf("\n\n\n\n\n\n HELLO FROM THE OTHER SIDE \n\n\n ");

    MPI_Init(NULL, NULL);
int world_size;
MPI_Comm_size(MPI_COMM_WORLD, &world_size);
int world_rank;
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    MPI_Status status;

char processor_name[MPI_MAX_PROCESSOR_NAME];
int len;

int check [1] = {0};

MPI_Get_processor_name(processor_name, &len);
    printf("Hello from processor: %s \n", processor_name) ;
    printf("Hello from processor rank : %d \n", world_rank) ;

/*****/
enum {
    CAPTURE_TIMES = 1,
};
urg_t urg;
long *data = NULL;
long oldData[682] = {0};
long newData[682];

    long time_stamp;
int n;
int i;
int shofta = 0;

    if (open_urg_sensor(&urg, argc, argv) < 0) {
        return 1;
    }

    data = (long *)malloc(urg_max_data_size(&urg) *
sizeof(data[0]));
```

```

        if (!data) {
            perror("urg_max_index()");
            return 1;
        }
    }

while(1)
{
    urg_start_measurement(&urg, URG_DISTANCE, CAPTURE_TIMES, 0);

    for (i = 0; i < CAPTURE_TIMES; ++i) {
        n = urg_get_distance(&urg, data, &time_stamp);
        if (n < 0) {
            printf("urg_get_distance: %s\n", urg_error(&urg));
            free(data);
            urg_close(&urg);
            return 1;
        }
    }

    int k = 0;
    int c;
    int z;

/*
        if (shofta == 0)
    {
        for( z = 0 ;z < 682 ;z++)
            { oldData[z] = data[z];

shofta ++ ;
    }

*/

    MPI_Send(data,682,MPI_LONG,2,1,MPI_COMM_WORLD);

    printf("\n\n\n waiting for check \n\n\n");
    /*****For synchronization*****/
    MPI_Recv(check,1,MPI_INT,2,10,MPI_COMM_WORLD,&status);

    /*****
    /
    printf("\n\n\n Recieved check \n\n\n ");

}

    free(data);
    urg_close(&urg);

    return 0;
}

```

3. Motion detection in Raspberry Pi 1 code (Frame Difference):

```
#include "opencv2/video/tracking.hpp"
#include "opencv2/imgproc/imgproc_c.h"
#include "opencv2/imgproc/imgproc.hpp"
#include <opencv2/opencv.hpp>
#include "opencv2/highgui/highgui.hpp"
#include <time.h>
#include <stdio.h>
#include <ctype.h>
#include <math.h>
#include <raspicam/raspicam_cv.h>
#include <raspicam/raspicam.h>
#include <string>
#include <sstream>
#include <iostream>
#include <ctime>
#include <cstdlib>
#include <fstream>

using namespace std;
using namespace cv;

int main(int argc, char** argv)
{

    raspicam::RaspiCam_Cv Camera;
    Camera.set(CV_CAP_PROP_FORMAT,CV_8UC3);

    if( !Camera.open() )
    {
        std::cerr << "Cannot open the camera" << std::endl;
    }

    Mat frame;
    int imageSize;

    Mat firstFrame;
    Mat gray;
    Mat deltaFrame;
    Mat thresh;
    Rect bounding_rect;

    vector<vector<Point> > contours;

    int minArea;
    int i = 0;

    while(1)
    {

        Camera.grab();
        Camera.retrieve(frame);
```

```

/*****Processing
Code*****/
cv::cvtColor(frame,gray,cv::COLOR_BGR2GRAY,3);
//cv::GaussianBlur(gray,gray,Size(21,21),0);

if(i== 7)
{
cout<<"\n I should only show once \n ";
firstFrame = gray.clone();
i =14;
} else if (i <7)
{
i++;
continue;
} else

{
i--;
cv::absdiff(gray,firstFrame,deltaFrame);

cv::threshold(deltaFrame,thresh,25,255,cv::THRESH_BINARY);

cv::dilate(thresh,thresh,Mat(),Point(-1,1), 2);

cv::findContours(thresh,contours,cv::RETR_EXTERNAL,cv::CHAIN_APPROX_
SIMPLE);

cout<<"\n contour size: " <<contours.size() <<"\n" ;

for(unsigned int j=0;j<contours.size();j++)
{
cout<<"\n contours area : " <<cv::contourArea(contours[j])<<"\n" ;

if (cv::contourArea(contours[j]) > 10000 )
{
cout<<"\n I am here \n";
bounding_rect=boundingRect(contours[j]);
cv::rectangle(frame,bounding_rect,Scalar(0,255,0),1,8,0);
}

}

/*****
*****/

namedWindow("VideoCapture", CV_WINDOW_AUTOSIZE);

IplImage imagea = frame; // transform from Mat type to IplImage
IplImage* Image = &imagea;

```

```
if (&imagea){  
  
if (!Image)  
    break;  
  
    cvShowImage("VideoCapture", Image); // The first parameter is  
the the window name and the 2nd param. is pointer to IplImage image  
                                        //(address of the image that  
was captured by camera)  
    if(waitKey(10) >= 0)  
        break;  
  
}  
  
firstFrame = gray.clone();  
  
}  
  
cvDestroyWindow("VideoCapture");  
  
return 0;  
}
```