

ECE496 Design Final Report: Object Recognition Framework for Streaming Video

Project Number: 2015385

Supervisor: Professor Ravi Adve

Administrator: Tome Kostas, Section 4

Date: Mar. 17, 2016

Jonathon Wong
Sanjana Kambalapally
Rayun Mehrab

Final Report Document Attribution Table

Section	Student Initials			
	1. JW	2. SK	3. RM	4.
Executive Summary	MR, ET	RD		
Group Highlights	RD, ET			
Introduction	RD, ET			
Background and Motivation	RS, RD, MR, ET	RS	RS	
Project Goal	ET		RD	
Project Requirements	ET		RS, RD, MR	
System-level Overview	RS, RD, ET			
Module-level Descriptions	RD, ET			
Assessment of Proposed Design	MR, ET		RD	
Gantt Chart		RD, MR		
Financial Plan	ET	RS, RD, MR		
Validation and Acceptance Tests	ET		RS, RD	
Module-level Test Results	ET	RD	RD	
System-level Test Results	ET	RD	RD	
Conclusion	MR, ET	RD		
All	FP, CM			

Abbreviation Codes:

Fill in abbreviations for roles for each of the required content elements. You do not have to fill in every cell. The “**All**” row refers to the complete document and should indicate who was responsible for the final compilation and final read through of the completed document.

RS – responsible for research of information

RD – wrote the first draft

MR – responsible for major revision

ET – edited for grammar, spelling, and expression

OR – other

“All” row abbreviations:

FP – final read through of complete document for flow and consistency

CM – responsible for compiling the elements into the complete document

OR - other

If you put OR (other) in a cell please put it in as OR1, OR2, etc. Explain briefly below the role referred to:

OR1: enter brief description here

OR2: enter brief description here

Signatures

By signing below, you verify that you have read the attribution table and agree that it accurately reflects your contribution to this document.

Name Jonathon Wong

Signature _____

Date: Mar. 17, 2016

Name Sanjana Kambalapally

Signature _____

Date: Mar. 17, 2016

Name Rayun Mehrab

Signature _____

Date: Mar. 17, 2016

Voluntary Document Release Consent Form¹

To all ECE496 students:

To better help future students, we would like to provide examples that are drawn from excerpts of past student reports. The examples will be used to illustrate general communication principles as well as how the document guidelines can be applied to a variety of categories of design projects (e.g. electronics, computer, software, networking, research).

Any material chosen for the examples will be altered so that all names are removed. In addition, where possible, much of the technical details will also be removed so that the structure or presentation style are highlighted rather than the original technical content. These examples will be made available to students on the course website, and in general may be accessible by the public. The original reports will not be released but will be accessible only to the course instructors and administrative staff.

Participation is completely voluntary and students may refuse to participate or may withdraw their permission at any time. Reports will only be used with the signed consent of all team members. Participating will have no influence on the grading of your work and there is no penalty for not taking part.

If your group agrees to take part, please have all members sign the bottom of this form. The original completed and signed form should be included in the hardcopies of the final report.

Sincerely,
Khoman Phang
Phil Anderson
ECE496Y Course Coordinators

Consent Statement

We verify that we have read the above letter and are giving permission for the ECE496 course coordinator to use our reports as outlined above.

Project ID: 385 Project Title: Object Recognition Framework for Streaming Video

Supervisor: Professor Ravi Adve

Administrator: Tome Koteski

Name	Jonathon Wong	Signature	_____	Date:	Mar. 17, 2016
Name	Sanjana Kambalapally	Signature	_____	Date:	Mar. 17, 2016
Name	Rayun Mehrab	Signature	_____	Date:	Mar. 17, 2016

¹ This form will be detached from the hardcopy of the final report. Please make sure you have nothing printed on the back page.

Executive Summary (author: Jonathon Wong)

Current security monitoring systems available on the market rely heavily on human attention. Incidentally, human inattention or negligence can lead to severe consequences. A simple example of this can be found in security rooms with a single security guard tasked with monitoring several dozens of camera feeds at once. If the guard were to fall asleep or simply leave the room, there would be serious security implications. The goal of our ECE496 project is to develop software for a semi-autonomous security monitoring system to reduce the risk of human error while finding a good balance between machine and human.

Ultimately, this proposed system must be able to detect and outline objects in motion (i.e. objects of interest) and allow users to select and de-select their own objects. Once detected, these objects must be tracked and users must be notified of potential security threats in real-time. In this case, potential threats are assumed to be objects of interest entering within close proximity to each other.

Our final design employed the use of a Raspberry Pi and a RaspiCam to remotely stream and process the video feed. This backend system would be in charge of filtering, detecting and tracking objects of interest and alerting a remote user on his/her Android mobile application over WiFi. Alerts would only be generated if these objects came in proximity of each other. In this way, the monitoring system would only require the user's attention for short periods of time.

Upon completion, several of the team's objectives were met, which included the ability to select and outline objects of interest, and notify users of movement within 7 seconds of receiving the image. Although communication speeds were subpar, achieving only 0.2 frames per second, the team deemed this an area for improvement for future work as it did not significantly hinder the project's overall success.

Group Highlights (author: Jonathon Wong)

Throughout the development cycle of this project, significant effort was invested in the design and architecture of the system. The project can be separated into three main components—front-end graphical user interface (GUI), back-end processing and algorithms, and framework and integration.

The team created a front-end Android app and the communication link between the front-end and the back-end. Video frames from a video file could be sent over the channel by converting them into Base64 Strings. The GUI for the Android app allowed the user to select a region by drawing over a video. The point coordinates would then be sent to the back-end.

Several back-end algorithms for filtering and post-processing on images for object detection and segmentation were also developed. This involved choosing an appropriate edge detection algorithm, in this case Canny Edge detection. It also included finding a method for segmenting objects digitally using various image processing techniques and some algorithms written to identify and supplement the segmented objects with their various properties. Successful segmentation was also achieved on selective backgrounds allowing for objects to be outlined for users. A Kalman filter approach for tracking objects given coordinates was also implemented.

Supporting the front-end and back-end functionality was the framework for integration. This included multithreading, code flow management, data encapsulation, and other various modules to help with debugging and testing.

Overall, the majority of the project's requirements were met which included successful object detection, outlining and tracking, and user notifications within 7 seconds of processing.

Individual Contributions: Jonathon Wong

During the course of this design project, I contributed to a variety of activities which included system-level and module-level design and architecture, framework implementation and Kalman filter tracking. The majority of these tasks revolved around simplifying integration of the team's parts.

I had designed the C++ framework with information flow, code maintainability, refactorability and scalability in mind. This included the custom-made debugging classes for logging and reading property files. Each module was designed to have single responsibilities to allow for decoupling of classes allowing for easy refactoring and simplified unit tests. Examples of modules that I created were the camera, data storage and communication interface modules. Collectively, these made up the skeleton of the entire backend component of the project.

Another component that I had worked on was the implementation and encapsulation of Kalman filters. Kalman filters were used to track and predict coordinates on a screen, typically for use when the motion was linear. These filters were incorporated and associated to a single object. This helped track objects even through temporary occlusions, but unfortunately would scale in required processing power with the number of objects being tracked.

Overall, this project offered several challenges and allowed me to apply my practical knowledge into a real-life project. It also gave me the opportunity to learn about several new aspects of software engineering, such as software architecture, design, integration, testing and development on new platforms with restricting hardware. Ultimately, I am proud of the work that my team and I put forward and the accomplishments we achieved.

Individual Contributions: Sanjana Kambalapally

Throughout my work on the design project, I was primarily responsible for the transmitter/receiver and mobile application modules. One of my first tasks involved designing of the communication channel between the Android application and the Raspberry Pi. Using the OpenCV functionality, I performed encoding/decoding of frames while sending and receiving frames. TCP sends a whole encoded string in pieces instead of all at once. The Android app handles for this by keeping track of the size of the string and the outstanding bytes received. These pieces are then combined and decoded to form an image on the app screen. In addition, a transparent view is present on top of the Android app streaming the camera frames, that allows the user to select regions and objects by pointing and drawing on the screen.

The Android app runs a sender and listener thread. The listener thread receives either a camera frame or a message that an object of interest has moved. It displays the camera on a screen or sends a notification to the user's phone. The sender thread sends any touches by the user to the Raspberry Pi. On the Raspberry Pi, 2 queues are maintained where one is filled with outgoing camera frames or notification messages used by the Transmitter and the other is populated by the Receiver with point coordinates such that the backend uses it.

During the design project, I helped my team members debugging unexpected problems such as installation of libraries on the Raspberry Pi and APIs. Overall, I am content that I implemented my prior knowledge in Android development and networking concepts as well as learnt of new libraries such that we meet the requirements of this project.

Individual Contributions: Rayun Mehrab

My contribution to the project include working on the modules Filtering, Object Detection, Object Evaluator and Data Storage. First of all, the camera feed from the raspicam has to be processed on the raspberry pi before transmitting to the user's device. To do this, I was responsible for applying a filtering function to the frames to strip them of any unnecessary data, in order to prepare for the edge detection. Edge detection allows us to detect and outline the different objects in the frames that our system can use in the future to interpret the user's action. Next, after considering algorithms such as Sobel Operator, Robert Cross Operator, Prewitt's Operator, Laplacian of Gaussian, Canny Edge Detection etc, I chose Canny Edge Detection as the best candidate for performing better than these other options in almost all test scenarios. Canny Edge Detection has a few parameters that needed tweaking, and I performed many tests over a controlled environment to determine the values that suits our needs the most. A challenge at this stage was to show the user the canny edges as white outlines over the original image, since canny shows them over a black background instead. However, this issue was successfully encountered.

Once the user selects the object of interest, more processing needs to be done at the Object Evaluator module to ensure the object of interest outlined by the canny edges is constantly monitored. In order to constantly monitor this region, it needed to be simply defined as a rectangle to enclose the object of interest; and then watched for movements in the rectangular region instead. In order to find this rectangular region from user's touch input in the form of a point coordinate, I wrote an algorithm that first applies a distance transform over the image. A distance transform gives each coordinate point on the picture a value that's the distance of that point to the closest edge. Afterwards, my algorithm takes the user's touch point, then finds the closest edge and subsequently the closest contour by iteration. This is the outline for the object of interest. The rectangular region is selected by looking at the furthest edges from the touch point in each direction. The task is then to create alerts based on a motion in the given region stored in the Data Storage module. I achieve this by taking the difference between two subsequent frames in the region and looking for any changes between them that'd indicate a motion occurred; and an alert event is created.

The above summarizes the contributions I made to the project, and how they relate to the big picture of the project in order to make it a successful one.

Acknowledgements

Our team would like to acknowledge the contributions made by our Administrator Tome Koteski and our Supervisor Professor Ravi Adve. Tome Koteski provided us with guidance and feedback throughout the year through the evaluation of our submitted documents. Professor Ravi Adve met with our team on a weekly basis to review our progress and offer suggestions.

Table of Contents

1. Introduction (author: Jonathon Wong)	1
2. Project Description	1
2.1 Background and Motivation (author: Jonathon Wong).....	1
2.2 Project Goal (author: Rayun Mehrab).....	1
2.3 Project Requirements (author: Rayun Mehrab)	1
2.3.1 Functions.....	1
2.3.2 Objectives	2
2.3.3 Constraints	2
3. Technical Design (author: Jonathon Wong)	2
3.1 System-level Overview	2
3.2 Module-level Descriptions	4
3.2.1 Camera	4
3.2.2 Filtering	4
3.2.3 Object Detection	5
3.2.4 Object Evaluator	5
3.2.5 Data Storage	6
3.2.6 Controller	6
3.2.7 Transmitter/Receiver	6
3.2.8 Mobile Application.....	7
3.3 Assessment of Final Design	7
4. Work Plan and Results (author: Sanjana Kambalapally)	8
4.1 Gantt Chart.....	8
4.2 Financial Plan.....	8
5. Testing and Verification	8
5.1 Validation and Acceptance Tests (author: Rayun Mehrab)	8
5.2 Module-level Test Results (author: Rayun Mehrab, Sanjana Kambalapally).....	9
5.2.1 Camera	9
5.2.2 Filtering and Object Detection.....	9
5.2.3 Object Evaluator, Data Storage and Controller	10
5.2.4 Transmitter/Receiver	10
5.2.5 Mobile Application.....	10
5.3 System-level Test Results (author: Rayun Mehrab, Sanjana Kambalapally)	11
6. Conclusion (author: Jonathon Wong).....	13
7. References	15
8. Appendices	17
8.1 Appendix A: Gantt Chart	17
8.2 Appendix B: Financial Plan	18
8.3 Appendix C: Validation and Acceptance Tests	18
8.4 Appendix D	19
8.5 Appendix E.....	20
8.6 Appendix F.....	20
8.7 Appendix G	21
8.8 Appendix H	22
8.9 Appendix I.....	23

1. Introduction (author: Jonathon Wong)

This report summarizes and highlights the motivation, design, implementation and testing of the development of a semi-autonomous security monitoring system for the final year design project course, ECE496. Within, details on the project goal and requirements are also summarized. The report concludes with suggestions of improvements and future work.

2. Project Description

2.1 Background and Motivation (author: Jonathon Wong)

With the world becoming evermore digitized, there is a heavy emphasis on cyber security and protection. Consequently, physical security has been severely underplayed. Each year, home owners and companies invest thousands to millions of dollars into monitoring and security systems. These systems usually involve few sensors to detect motion or return video feeds, and require human labour to monitor or assess situations. Human error—in particular, error caused by human inattention and negligence^[1]—has major consequences in security. In extreme cases, it is also one of the leading contributors to security breaches according to numerous studies^[2]. A more desired solution would be one that will be more economical and more reliable by reducing the dependency of human inspection. In this project, a semi-automated monitoring software system will be designed that will take input from a camera and, after processing the output, will be able to relay information about moving objects to an end user in real-time.

2.2 Project Goal (author: Rayun Mehrab)

The goal of this project is to develop a real-time semi-automated monitoring system that reduces the load on an end user.

2.3 Project Requirements (author: Rayun Mehrab)

Throughout the rest of this document we will define “objects of interest” as being any objects in motion with respect to the background or any object specified by the user.

2.3.1 Functions

- Objects of interest shall be outlined for an end user;

- Users shall receive relevant information about the objects of interest remotely, if requested;
- Users shall be able to select and de-select objects of interest for tracking.

2.3.2 Objectives

- Ensure system is able to detect objects with at least 75% accuracy;
- Design the system to track objects of interest with a 75% accuracy rate;
- Relay relevant information about the objects of interest to the end-user with an accuracy rate of 75% and in real-time.

2.3.3 Constraints

- Video must be processed at a minimum of 1 frame per second;
- Background of the video must be stationery;
- The software must be able to handle standard MP4, MPEG and FLV encodings;
- Camera must have a field of view of at least 90 degrees;
- The overall cost of the project must be less than the average cost of a monitoring system, which is 45\$/month^{[3][4]};
- Notifications must be sent to the user within one minute.

3. Technical Design (author: Jonathon Wong)

3.1 System-level Overview

The overall design of the system involves a camera overlooking a scene to create a live video feed (Figure 1). The images of this video are then passed into the software data processing module.

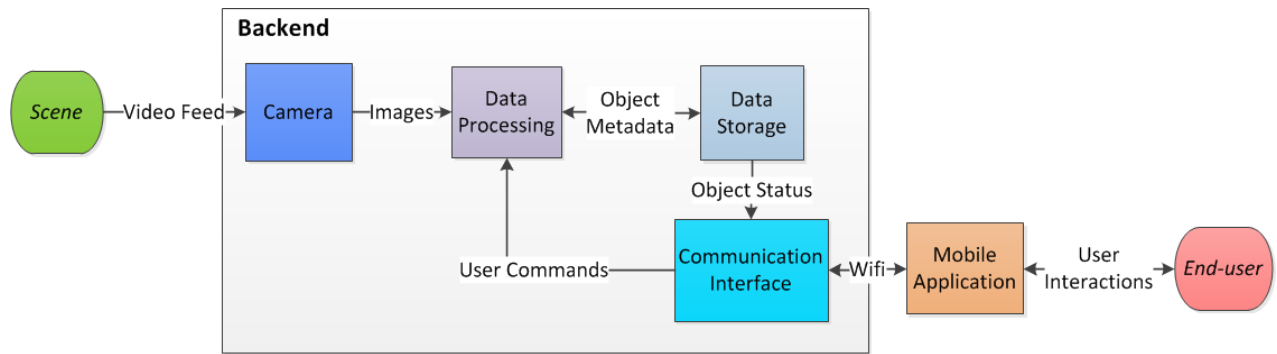


Figure 1. High-level System Overview.

The data processing module (Figure 2) consists of three sub-modules in which images are filtered and objects are detected from the filtered images. The objects are then evaluated, and relevant metadata about each object are sent for storage. The metadata includes information about the objects' coordinates, velocities, and centres of mass. This information is used afterwards as input to the object detection module to help track the objects.

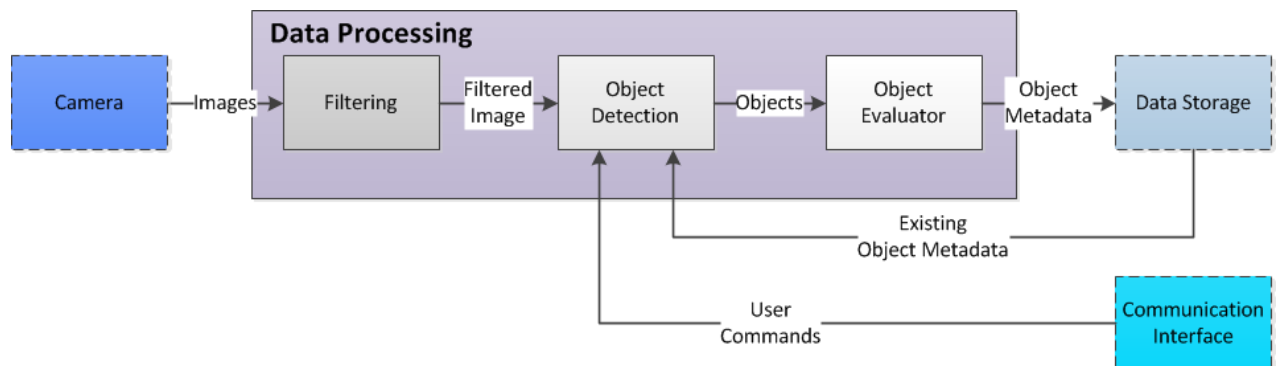


Figure 2. Diagram demonstrating the sub-modules within the data processing module.

Additionally, updates in the object metadata notify the communication interface module (Figure 3). The updates are sent directly to the controller sub-module, which determines the information to be deliver to the front-end's mobile application. End-users will be allowed to interact with the mobile application and specify commands back to the communication interface, such as selecting an object. The controller will then parse the command and call on the data processing module to execute it.

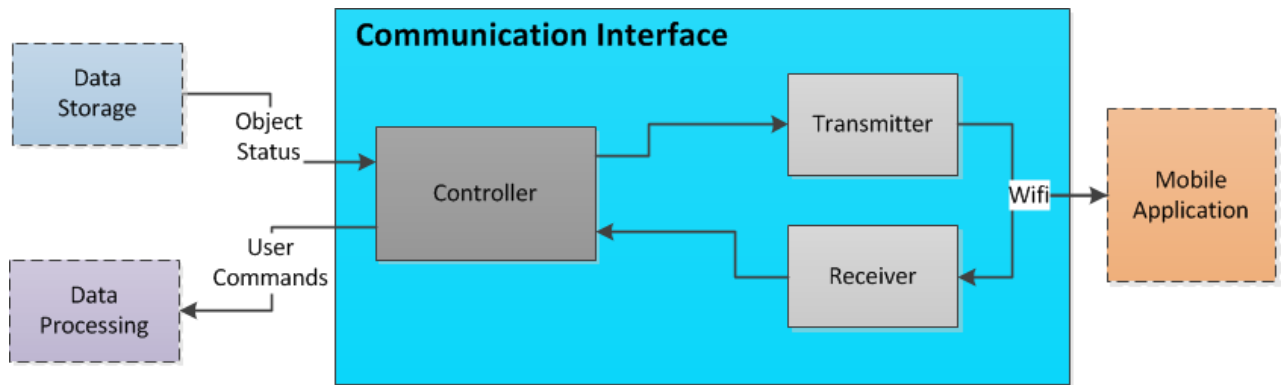


Figure 3. Diagram demonstrating the sub-modules within the Communication Interface Module.

3.2 Module-level Descriptions

The following sections describe the inputs, outputs, and functions of the modules in the previous section.

3.2.1 Camera

Table 1: Camera Module from Figure 1

Camera
Inputs: <ul style="list-style-type: none"> • Light • Real-world scenery
Output: Digitized image (stored as an OpenCV Mat object)
Function: The camera module digitizes reality and relays the resulting real-time raw video to the back-end processes.

3.2.2 Filtering

Table 2: Filtering Module from Figure 2

Filtering
Inputs: <ul style="list-style-type: none"> • Digitized image (stored as an OpenCV Mat object)
Output: Filtered image (stored as an OpenCV Mat object)

Function:

The filtering module performs pre-processing on the incoming images and outputs a filtered image removing any extraneous data.

3.2.3 Object Detection

Table 3: Object Detection Module from Figure 2

Object Detection
Inputs: <ul style="list-style-type: none">• Filtered image (stored as an OpenCV Mat object)
Output: List of meta-objects (custom class)
Function: <p>The object detection module takes the filtered images and performs post-processing. This will extract objects of interest within the image. Information about previously seen objects are used as input to prevent objects from seemingly disappearing. These objects are then segmented and sent to the object evaluator.</p>

3.2.4 Object Evaluator

Table 4: Object Evaluator Module from Figure 2

Object Evaluator
Inputs: <ul style="list-style-type: none">• Meta-object (custom class)
Output: Meta-object (custom class) with meta-data (coordinates, centre of mass, borders, colour)
Function: <p>The object evaluator module uses the segmented, detected objects to further analyze them. Attributes are then extracted from each object of interest and the corresponding metadata, such as coordinates and borders, are processed.</p>

3.2.5 Data Storage

Table 5: Data Storage Module from Figure 1

Data Storage
Inputs: <ul style="list-style-type: none">• Meta-object (custom class)
Output: Meta-object (custom class)
Function: <p>The data storage module stores the metadata about the objects of interest. Inputs can include object coordinates, centre of masses and colours. This metadata can be extracted upon request by other modules.</p>

3.2.6 Controller

Table 6: Controller Module from Figure 3

Controller
Inputs: <ul style="list-style-type: none">• List of meta-objects (custom class)• User commands (enumerated)
Output: List of meta-objects (custom class)
Function: <p>The controller module controls the flow of information to and from the user and the back-end processes. User commands are inputted into the controller and the controller schedules and routes the requests to the appropriate destination. Object statuses are also received by the controller, allowing for the controller to determine the information that are relayed to the transmitter.</p>

3.2.7 Transmitter/Receiver

Table 7: Transmitter/Receiver Module from Figure 3

Transmitter/Receiver
Inputs:

<ul style="list-style-type: none"> • List of meta-objects (custom class) • Byte stream
Output: Byte stream, user commands (enumerated)
Function: The transmitter and receiver module will manage the Wi-Fi connection between the back-end and the client. It will transmit requested information as well as receive requests from the user via the mobile application.

3.2.8 Mobile Application

Table 8: Mobile Application Module from Figure 1

Mobile Application
Inputs: <ul style="list-style-type: none"> • Byte stream
Output: Graphical user interface displaying object information
Function: This module relays relevant information to the user and allows remote interaction with the data. Specifically, this module will take as input information over Wi-Fi from the backend and display the information in an intuitive way to the end-user. The user may then interact with the module to initiate commands to send back to the server.

3.3 Assessment of Final Design

The final design was chosen mainly with flexibility and speed of development in mind. The back-end system hardware employs the use of the Raspberry Pi camera module coupled with Raspberry Pi. These were chosen for their inherent synergy and available support, which expedited development time. They were also chosen due to their relatively low costs and flexibility. Although this was encompassed in a relatively small form-factor, the hardware lacked significantly in terms of processing power and memory. Linux and C++ were chosen as the operating system and programming language due to their low processor overhead, speed and flexibility. This also gave the team convenient access to the open source OpenCV C++ libraries [6][7] for optimized image processing algorithms. Since Linux and C++ were already familiar subjects known to the team, these required little learning overhead. Wi-Fi was chosen as the

communication protocol for its remote access capabilities and availability across all platforms. For the mobile application, Android was chosen for its large user base and ease of access. As compared to iOS, Android development did not require proprietary hardware, which decreased required costs.

4. Work Plan and Results (author: Sanjana Kambalapally)

4.1 Gantt Chart

Please refer to Appendix A for a breakdown of the individual task timeline.

4.2 Financial Plan

The design for this project revolves mainly around a software application, which does not require much capital. Any monetary expenses listed below derive from the hardware components, where the software will run on. The hardware required is already provided for by the students. The labour capital below is derived from the average Computer Engineer salary, which is \$37/hour in Toronto ^[8]. See Appendix B for the financial plan along with the actual expenses. There were no additional expenses since the project proposal, but the actual number of hours in the student labour section has been updated.

5. Testing and Verification

5.1 Validation and Acceptance Tests (author: Rayun Mehrab)

1. The software passes if it is able to correctly detect events, when the object of interest is in motion, more than 75% of the time for a set of benchmark test scenarios. These scenarios will contain variations of different objects of interest and backgrounds. To pass, the event must be correctly detected 75% of the times.

According to the original validation and acceptance test (see appendix C), our system was required to detect all moving objects to a 75% accuracy to pass this test. However, our project requirement has since been changed to only track objects in a smaller region (determined by the location of the object of interest) and as such, all moving objects are not detected.

2. The edge-detection of objects is tested by comparing the number of expected object outlines with the number of objects detected by the algorithm. The foreground test passes if more than 80% of foreground object outlines are detected.
3. The software's measure of real-time is tested by ensuring that the time between detecting a potential threat and sending the user an alert is less than a minute. The frame rate must also be processed at a rate of at least 1 frame per second (fps).

According to the original test (appendix C), the target frame rate was 20 fps. However, this has since been changed to 1 fps, giving the system more time to process the frames more accurately. Moreover the design does not require fluid motion in each frame to achieve its goals, making 1 fps or less acceptable as well.

5.2 Module-level Test Results (author: Rayun Mehrab, Sanjana Kambalapally)

5.2.1 Camera

Requirement	Target specification	Final Result	Compliance? (Pass/Fail)	Comments and Documentation
Camera must have a field of view of at least 90 degrees	90 degrees	53.50 degrees	Fail	The only camera compatible with the Raspberry Pi has a field view of 53.5 degrees ^[10] .

5.2.2 Filtering and Object Detection

Requirement	Target specification	Final Result	Compliance? (Pass/Fail)	Comments and Documentation
Objects of interest shall be outlined for an end user	Outline borders/edges of each object	Rectangular boxes created around each object in frame	Pass	From the camera feed of a test scenario, the canny edge detection algorithm was applied to outline objects of an image. This was tested at the module level by checking that objects were properly outlined after the Filtering and Object Detection modules of the Raspberry Pi. See Appendix D Section 1 for sample images of their output.

5.2.3 Object Evaluator, Data Storage and Controller

Requirement	Target specification	Final Result	Compliance? (Pass/Fail)	Comments and Documentation
Users shall be able to select and deselect objects of interest for tracking.	Store information about objects	Screen coordinates of the object of interest and its outline	Pass	At the module level, the location of the selected object was stored and used to make future decisions about the tracking region. As seen in Appendix E, the point coordinates sent by the user are stored in the Data Storage module.

5.2.4 Transmitter/Receiver

Requirement	Target specification	Final Result	Compliance? (Pass/Fail)	Comments and Documentation
Video must be processed at a minimum of 1 frame per second (fps)	1 fps (Modified and explained in section 5.1)	1fps	Pass	This test was performed by sending video frames to the mobile application. The fps was tuned to the transmitter module's performance level. Thread.sleep() was used to control the thread speed. At 2 fps or higher, out of memory errors occurred, where the size of the video frame buffer was too large. This error can be seen in Appendix F. This error did not occur at 1 fps.

5.2.5 Mobile Application

Requirement	Target specification	Final Result	Compliance? (Pass/Fail)	Comments and Documentation
Users shall be able to select and deselect objects of interest for tracking.	Press objects on mobile app and send coordinate information to backend	Pressing objects on the app sends point coordinates of the touch to the backend	Pass	Selecting and deselecting objects in the mobile app, consisted of sending the point coordinates of the touch to the backend. A test listener was created to receive the coordinates running on a laptop. Refer to Appendix G for sample output and test listener code.

Users shall receive relevant information about the objects of interest remotely, if requested	<ul style="list-style-type: none"> - When an object moves, trigger a notification - Display camera frames 	<ul style="list-style-type: none"> -When an object moves, the backend sends a message to the app that the object selected has moved -Displays camera frames 	Pass	The test for this requirement was done by sending a message from the transmitter module in the Pi. When the app receives a message beginning with the keyword “notification”, a notification is generated in the user’s application. Refer to Appendix H for screenshots of notification and display of camera frames in the app.
---	---	---	------	---

5.3 System-level Test Results (author: Rayun Mehrab, Sanjana Kambalapally)

Requirement	Target specification	Final Result	Compliance? (Pass/Fail)	Comments and Documentation
Objects of interest shall be outlined for an end user	Outlined objects should appear on Android app	Rectangular boxes appear around objects and appears on user’s app	Pass	This test was passed system-wide by performing Canny Edge detection on the Raspberry Pi to outline objects in the camera feed, and then successfully transmitting the outlined feed to the user on the mobile app. Further evidence can be seen in Appendix D sections 2 and 3, where the screenshot of the mobile app that shows the outlined image, as well as a regular image can be seen. These demonstrate that all objects were outlined successfully.
Users shall be able to select and deselect objects of interest for tracking.	<ul style="list-style-type: none"> - When object is touched on app screen, backend Pi keeps track of each selected object - When a selected object is 	When object is touched, the point coordinates of the touch is used in the backend to keep track of the selected object.	Partial Pass	As seen in Appendix E, when a user selects an object on the Android app, a point coordinate is sent to the Raspberry Pi. However, the functionality of deselecting objects wasn’t implemented due to time limitations.

	touched, it is deselected			
Design the system to track objects of interest with a 75% accuracy rate	75%	50%	Fail	Once the user selects an object interest by using a touch input, the input point coordinate is then processed by the system to evaluate a rectangular region where the object of interest resides. It is able to select a region bigger than the object of interest 100% of the time. However, since it is bigger than the object of the interest, 50% of the time it detects an event of motion when something other than the object of interest (but still within the region) is moved. This has been tested by running multiple tests in a controlled environment. As such, our target of 75% accuracy is breached and this test fails.
Ensure system is able to detect objects with at least 75% accuracy	75%	100%	Pass	The system is able to use edge detection to outline 100% of the objects in the camera feed. This test has been performed by comparing the image output of edge detection with a regular image. As evidenced by Appendix D section 3, 100% of the objects were detected and outlined.
Relay relevant information about the objects of interest to the end-user with an accuracy rate of 75% and in real-time	75%	100%	Pass	The system relays relevant information by communicating over TCP with an android app on the user's mobile device. The user can view a live stream of the camera feed from the Raspberry Pi, or choose to receive alerts when their chosen object of interest is detected to be moving. This test has been verified through consistently receiving alerts 100% when a motion

				detection event is triggered in the controlled scenarios and is demonstrated by the sample notification alert in Appendix H section 2.
Video must be processed at a minimum of 1 frame per second;	1 fps	0.2 fps	Fail	When all the modules were integrated, the processing on the frames increased the streaming latency. For any frame rate higher than 0.2fps an out of memory error was observed similar to the one in Appendix E.
The software must be able to handle standard MP4, MPEG and FLV encoding	Compatible with MP4, MPEG and FLV encoding	Not Applicable	Pass	The images captured by the Raspberry Pi camera are raw and are never encoded or decoded in a specific format with OpenCV.
The overall running cost of the project must be less than the average cost of a monitoring system, which is 45\$/month	\$45/month	\$25/month ^[11]	Pass	The running cost of the project is incurred from the internet plan of the user. The cheapest internet plan offered by Rogers is \$25/month ^[11] .
Notifications must be sent to the user within one minute.	1 minute	7 seconds	Pass	The end to end time is measured by subtracting the timestamp of the camera frame captured by the Pi from the timestamp when the notification message/camera frame is received by the Android app. Refer to Appendix I to see the result.

6. Conclusion (author: Jonathon Wong)

The team's design of integrating a Raspberry Pi and camera to remote end-user mobile devices is an appropriate solution that addresses the problem of the lack of balance between the human- and machine-driven security systems currently available in the market. In an attempt to reduce human

error and utilize the performance benefits of software applications, the chosen design allows the user to actively make decisions to guide the software, in order to monitor objects of interest. The final result of the team's work enabled users on their Android mobile platforms to select objects of interest and have them outlined. The remote backend Raspberry Pi was then able to process a video feed and communicate to the user at approximately 0.2 frames per second. Although this frame rate did not meet the team's specifications, it was still capable of meeting and exceeding the requirement of notifying a user within a minute. From these results, it is arguable that this design may be able to lay the groundwork for a more preferable alternative solution to current systems, which are heavily reliant on human participation.

While the current design is able to perform in simple cases with few moving objects and a stationary background such as baby monitoring or household monitoring, there are more complex scenarios that would require much more processing power and stronger algorithms. To further improve this design, the backend hardware would need to be enhanced with a larger amount of processing power and memory to accommodate the increasing number of objects to be tracked. Furthermore, with the increased number of objects, proximity becomes less viable as a metric for threat, and the design would therefore require a new metric for measuring threat. With these improvements, this project could be employed in practically any security system.

7. References

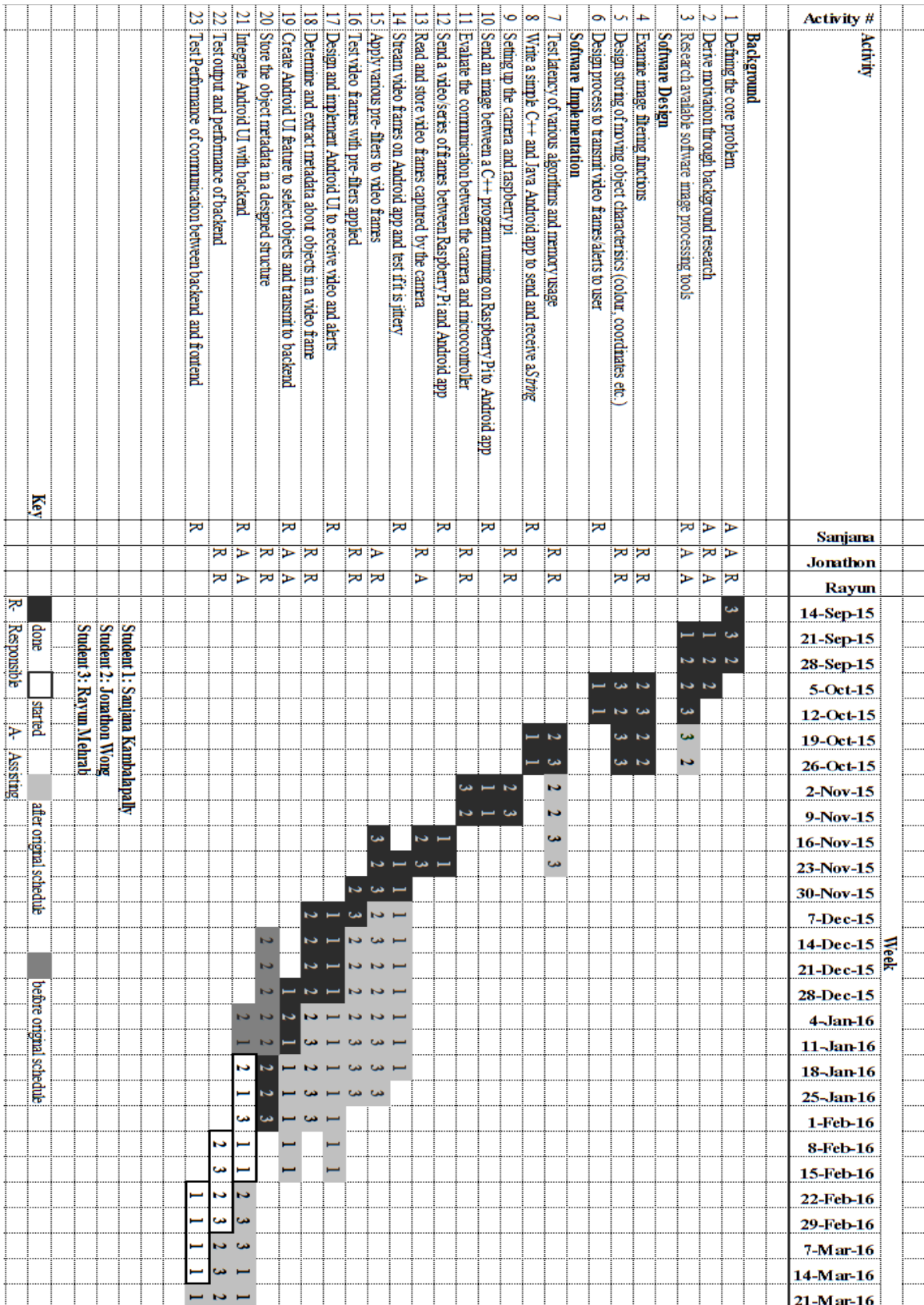
- [1] J. Reason (2000). Human Error. [Online] Accessed Mar 10, 2016. Available: <http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1070929/>
- [2] SCMagazine (2015). Human error cited as leading contributor to breaches, study shows. [Online] Accessed Oct 6, 2015. Available: <http://www.scmagazine.com/study-find-carelessness-among-top-human-errors-affecting-security/article/406876/>
- [3] ADT Home Security (2015). ADT Monitoring Packages. [Online] Accessed Oct 3, 2015. Available: <http://www.homesecurity.io/adt-packages>
- [4] Rogers (2015). Smart Home Monitoring Solutions. [Online] Accessed Oct 3, 2015. Available: <http://www.rogers.com/web/content/smart-home-monitoring-solutions-new>
- [5] Amazon (2015). Raspberry Pi Camera Module. [Online] Accessed Oct 3, 2015. Available: http://www.amazon.ca/Raspberry-Pi-100003-Camera-Module/dp/B00E1GGE40/ref=sr_1_3/188-0129279-6904471?ie=UTF8&qid=1444105472&sr=8-3&keywords=camera+raspberry+pi
- [6] OpenCV (2011). Motion Analysis and Object Tracking. [Online] Accessed September 21, 2015. Available: http://docs.opencv.org/modules/video/doc/motion_analysis_and_object_tracking.html
- [7] Wordpress (2012). What is OpenCV? OpenCV vs. MATLAB — An insight. [Online] Accessed Oct 5, 2015. Available: <https://karanjthakkar.wordpress.com/2012/11/21/what-is-opencv-opencv-vs-matlab/>
- [8] Living in Canada (2015). Computer Engineer Salary Canada. [Online] Accessed Oct 5, 2015. Available: <http://www.livingin-canada.com/salaries-for-computer-engineers-canada.html>
- [9] Security Camera King (2015). What is fps frame per second. [Online] Accessed Feb 18, 2016. Available: <http://www.securitycameraking.com/securityinfo/what-is-fps-frames-per-second/>

[10] Raspberry Pi (2015). Camera. [Online] Accessed Dec 15, 2015. Available:
<https://www.raspberrypi.org/documentation/hardware/camera.md>

[11] Rogers (2016). Packages. [Online] Accessed Mar 2, 2016. Available:
<http://www.rogers.com/consumer/internet#packages>

8. Appendices

8.1 Appendix A: Gantt Chart



8.2 Appendix B: Financial Plan

Item	Cost/Unit	Quantity (# or hrs)	Funding Required	Kept/Paid by Students	Total	Actual Expenses
Capital Requirement						
Laptop	\$1000	3	N	Y	\$3000	-
Raspberry Pi 2 Kit	\$130	1	Y	Y	\$130	\$130
Raspberry Camera Module	\$30 ^[5]	1	Y	Y	\$30	\$30
Total Capital Requirement					\$3160	\$160
Student labor						
Sanjana	\$37	200			\$7400	\$5920 (160hrs)
Jonathon	\$37	200			\$7400	\$6150 (165hrs)
Rayun	\$37	200			\$7400	\$6179 (167hrs)
Total Student Labour (Unfunded)					\$22200	\$18249
Total Cost of Project					\$25360	\$18409
Total Cost Requiring Funding					\$160	\$160

Table 1: Detailed financial breakdown of design project.

8.3 Appendix C: Validation and Acceptance Tests

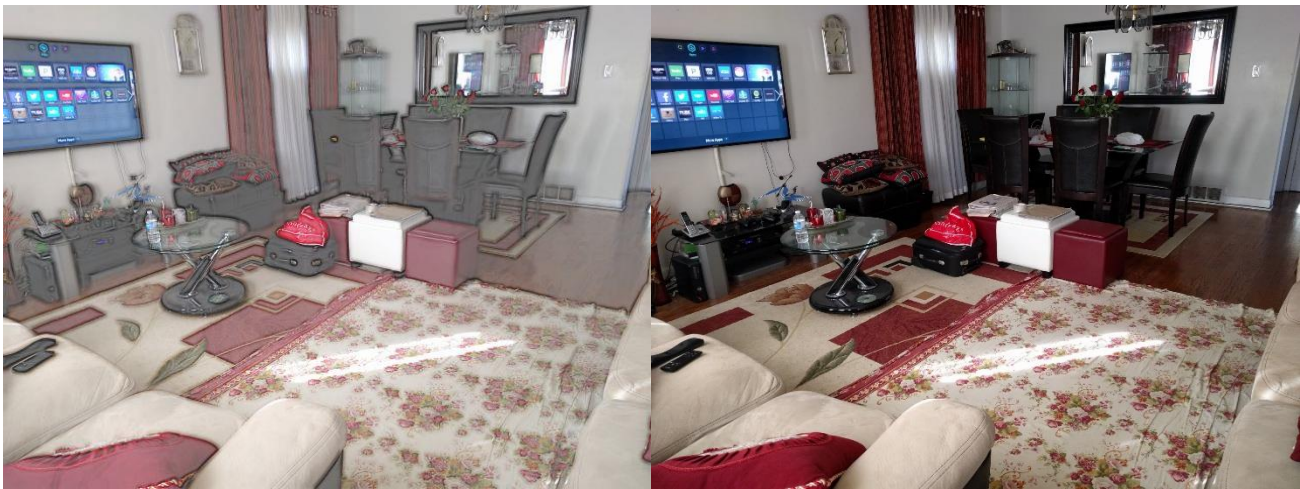
Validation and Acceptance Tests (copied from proposal)

1. The software passes if it is able to correctly detect moving objects more than 75% of the time for a set of benchmark test scenarios. These scenarios will contain variations of different objects and backgrounds. Objects that move at a speed of at least 20 pixels per second must be detected in order for a passing condition.
2. The edge-detection of objects is tested by comparing the number of expected object outlines to be detected with the number of objects detected by the algorithm. The foreground test passes if more than 80% of foreground object outlines are detected.

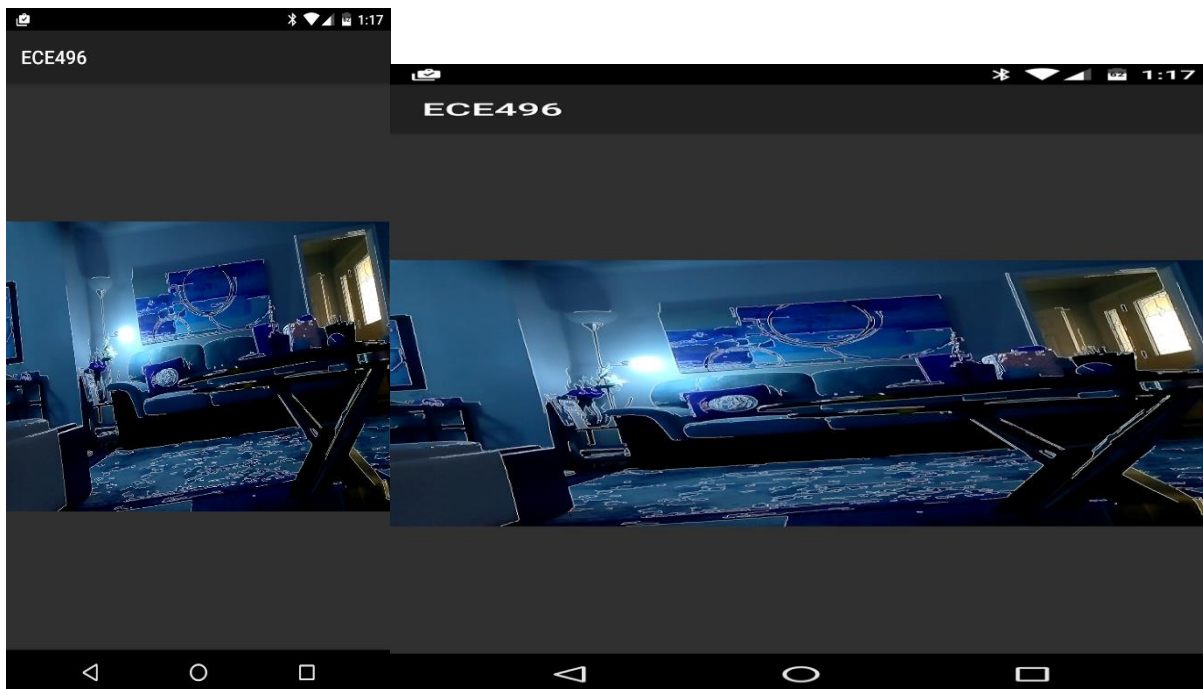
3. The software's measure of real-time is tested by ensuring that the time between detecting an issue and sending the user an alert is less than a minute. The frame rate must also be processed at a rate of at least 20 frames per second.

8.4 Appendix D

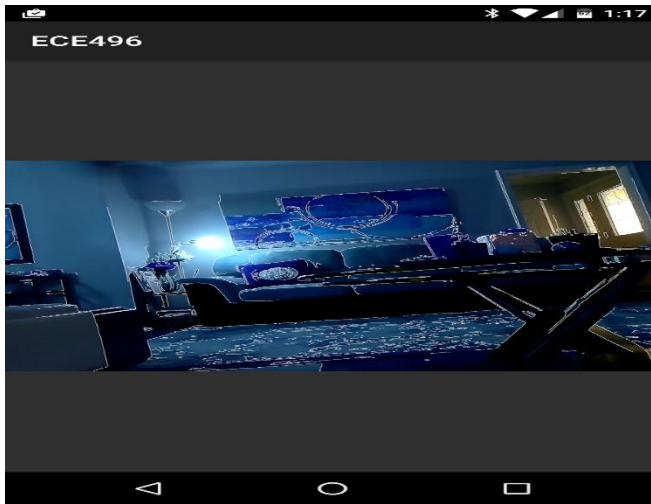
- 1) Objects in a controlled location image outlined by the canny edge detection on the raspberry pi, seen against the original image



- 2) Screenshots of the feed from Raspberry Pi with different objects clearly outlined as seen on the Android application



3) The above screenshot compared with a regular photo of the same location



8.5 Appendix E

```
pi@raspberrypi:~/Desktop/ObjectTracker/Debug $ ./ObjectTracker
17-03-2016 01:58:49 Main TRACE Logger initialized.
17-03-2016 01:58:49 PropertyReader TRACE Opening file: settings/settings.txt
17-03-2016 01:58:49 PropertyReader INFO File opened: settings/settings.txt
17-03-2016 01:58:49 PropertyReader INFO Loading properties file...
17-03-2016 01:58:49 PropertyReader INFO Mapped TRACE to LOGGING_LEVEL
17-03-2016 01:58:49 PropertyReader INFO Mapped 100 to CAM_BUFFER_SIZE
17-03-2016 01:58:49 PropertyReader INFO Mapped 15 to CAM_POLL_FREQUENCY
17-03-2016 01:58:49 PropertyReader INFO Mapped 0 to CAM_INDEX
17-03-2016 01:58:49 PropertyReader DEBUG Loaded 4 properties successfully
17-03-2016 01:58:49 Controller INFO Constructor controller
17-03-2016 01:58:49 Transmitter INFO socket retrieve success
17-03-2016 01:58:49 Transmitter INFO binding done
17-03-2016 01:58:49 Transmitter INFO done listening
17-03-2016 01:59:09 Transmitter INFO accepting
17-03-2016 01:59:09 Main TRACE initializing controller
17-03-2016 01:59:09 Controller INFO Initializing Controller
17-03-2016 01:59:09 Main TRACE initializing dataprocessor
17-03-2016 01:59:09 CameraInterface INFO Initializing camera
17-03-2016 01:59:09 CameraInterface INFO Opening video capture stream at position 0
17-03-2016 01:59:09 Receiver INFO Starting Receiver Thread execution loop
17-03-2016 01:59:09 ControllerThread INFO Starting Controller Thread execution loop
17-03-2016 01:59:09 Transmitter INFO Starting Receiver Thread execution loop
17-03-2016 01:59:09 Receiver INFO Received point: 393.45, 450.68
17-03-2016 01:59:09 Receiver INFO Received point: 100.0, 50.9
17-03-2016 01:59:09 Receiver INFO Received point: 170.784, 410.45
17-03-2016 01:59:09 Main TRACE running controller
17-03-2016 01:59:09 DataProcessor INFO polling
17-03-2016 01:59:10 CameraInterface INFO Read successful
17-03-2016 01:59:12 Controller INFO sending to transmitter
17-03-2016 01:59:13 Transmitter INFO pushing into buff
17-03-2016 01:59:13 Transmitter INFO getting from buffer
```

8.6 Appendix F

The following error message appears when the streams are sent at a 2fps or higher frame rate.

'OpenCV Error: Insufficient memory (Failed to allocate 3686400 bytes) in OutOfMemoryError, file /home/pi/opencv-3.0.0/modules/core/src/alloc.cpp, line 52'

```
17-03-2016 03:16:24 CameraInterface INFO Read successful
17-03-2016 03:16:24 Controller INFO sending to transmitter
17-03-2016 03:16:25 Transmitter INFO sending string: 3653
17-03-2016 03:16:25 Transmitter INFO pushing into buff
17-03-2016 03:16:25 DataProcessor INFO polling
17-03-2016 03:16:25 Transmitter INFO getting from buffer
OpenCV Error: Insufficient memory (Failed to allocate 3686400 bytes) in OutOfMemoryError, file /home/pi/opencv-3.0.0/modules/core/src/alloc.cpp, line 52
OpenCV Error: Assertion failed (u != 0) in create, file /home/pi/opencv-3.0.0/modules/core/src/matrix.cpp, line 411
terminate called after throwing an instance of 'cv::Exception'
what(): /home/pi/opencv-3.0.0/modules/core/src/matrix.cpp:411: error: (-215) u != 0 in function create
Aborted
```

8.7 Appendix G

1) Point Coordinates of the touches on the Android app received by a test C++ program listening.

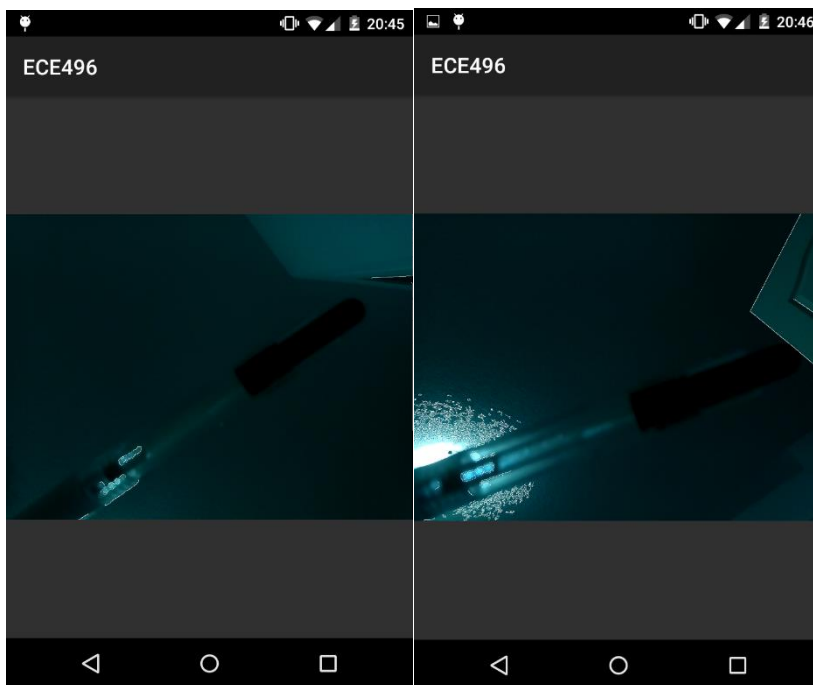
```
C:\Users\srkambbs\Desktop\FourthYear\design project\Code>java TCPServer
Received: 331.0, 460.0
Received: 328.51495, 463.94025
Received: 328.0, 466.0
Received: 328.0, 466.0
Received: 362.0, 461.0
Received: 362.0, 461.0
Received: 376.0, 476.0
Received: 376.0, 476.0
Received: 365.0, 511.0
Received: 365.0, 511.0
Received: 350.0, 348.0
Received: 350.0, 348.0
```

2) C++ code of test listener

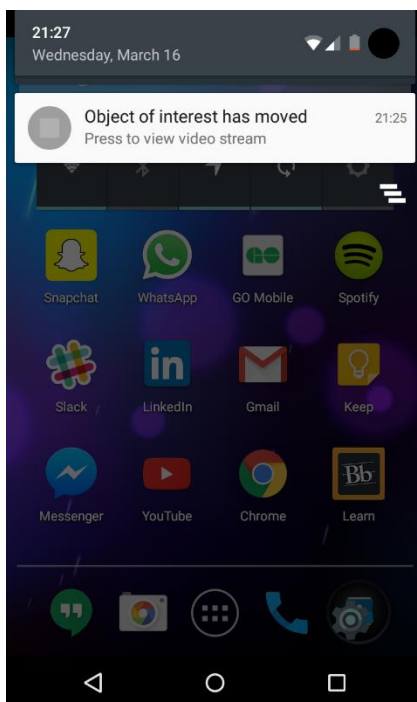
```
1  import java.io.*;
2  import java.net.*;
3
4  class TCPServer
5  {
6      public static void main(String argv[]) throws Exception
7      {
8          String clientSentence;
9          String capitalizedSentence;
10         ServerSocket welcomeSocket = new ServerSocket(5001);
11         Socket connectionSocket = welcomeSocket.accept();
12         BufferedReader inFromClient = new BufferedReader(new InputStreamReader(connectionSocket.getInputStream()));
13         DataOutputStream outToClient = new DataOutputStream(connectionSocket.getOutputStream());
14
15         while(true)
16         {
17             clientSentence = inFromClient.readLine();
18             System.out.println("Received: " + clientSentence);
19
20         }
21     }
22 }
23 }
```

8.8 Appendix H

1) Screenshots of camera feed from Raspberry Pi being streamed on Android application



2) Notification appears on notification bar of user's phone, when triggered by backend.

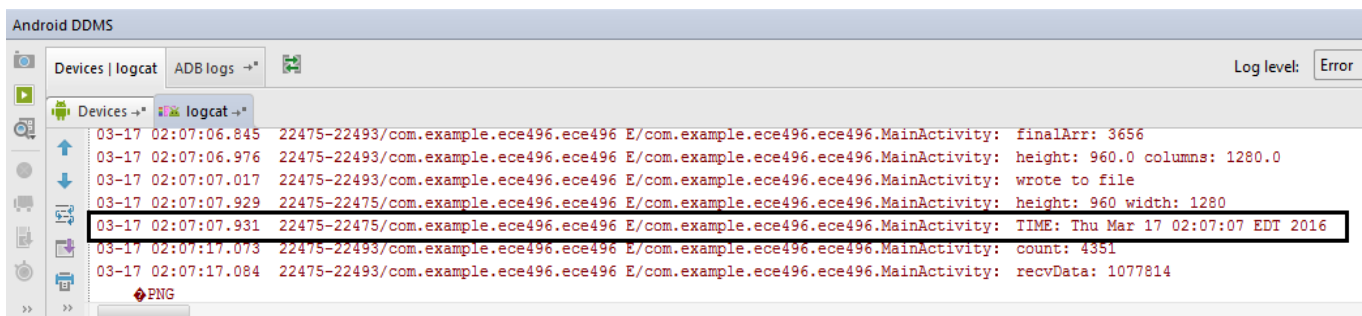


8.9 Appendix I

1) Below is a timestamp from the Raspberry Pi of the first frame captured by the camera in UTC, which is 2:07:00 EST.

```
17-03-2016 06:06:59 Receiver INFO Received point: 100.0, 50.9
17-03-2016 06:06:59 Receiver INFO Received point: 170.784, 410.45
17-03-2016 06:07:00 Main TRACE running controller
17-03-2016 06:07:00 DataProcessor INFO polling
17-03-2016 06:07:00 CameraInterface INFO Read successful
17-03-2016 06:07:03 Controller INFO sending to transmitter
17-03-2016 06:07:03 Transmitter INFO pushing into buff
17-03-2016 06:07:03 Transmitter INFO getting from buffer
17-03-2016 06:07:06 Transmitter INFO sending string: 3656
17-03-2016 06:07:08 DataProcessor INFO polling
17-03-2016 06:07:08 CameraInterface INFO Read successful
17-03-2016 06:07:11 Controller INFO sending to transmitter
17-03-2016 06:07:11 Transmitter INFO pushing into buff
17-03-2016 06:07:11 Transmitter INFO getting from buffer
17-03-2016 06:07:16 DataProcessor INFO polling
17-03-2016 06:07:16 CameraInterface INFO Read successful
17-03-2016 06:07:17 Transmitter INFO sending string: 1077814
17-03-2016 06:07:19 Controller INFO sending to transmitter
17-03-2016 06:07:19 Transmitter INFO pushing into buff
17-03-2016 06:07:19 Transmitter INFO getting from buffer
```

2) Below is the timestamp when the frame is displayed in the Android app, which is 2:07:07 EST.



```
Android DDMS
Devices | logcat ADB logs -> Log level: Error
Devices -> logcat ->
03-17 02:07:06.845 22475-22493/com.example.ece496.ece496 E/com.example.ece496.ece496.MainActivity: finalArr: 3656
03-17 02:07:06.976 22475-22493/com.example.ece496.ece496 E/com.example.ece496.ece496.MainActivity: height: 960.0 columns: 1280.0
03-17 02:07:07.017 22475-22493/com.example.ece496.ece496 E/com.example.ece496.ece496.MainActivity: wrote to file
03-17 02:07:07.929 22475-22475/com.example.ece496.ece496 E/com.example.ece496.ece496.MainActivity: height: 960 width: 1280
03-17 02:07:07.931 22475-22475/com.example.ece496.ece496 E/com.example.ece496.ece496.MainActivity: TIME: Thu Mar 17 02:07:07 EDT 2016
03-17 02:07:17.073 22475-22493/com.example.ece496.ece496 E/com.example.ece496.ece496.MainActivity: count: 4351
03-17 02:07:17.084 22475-22493/com.example.ece496.ece496 E/com.example.ece496.ece496.MainActivity: recvData: 1077814
```

From 1) and 2), the maximum end to end delay between backend and frontend is 7 seconds.