Remote Escalation/De-Escalation and Surveillance System (REDSS)

ECE4012 Senior Design Project

Team Pew^3
Dr. Michael E. West
Harris Corporation

Michael Bossi (michaelb@gatech.edu)
Joshua Dixon (josh.dixon@outlook.com)
Gyu Cheol Lim (gclim@gatech.edu)
Seong Ho Yeon (shyeon568@gatech.edu)
Satoshi Yuki (syuki1021@gatech.edu)

Submitted

14 December 2016

Executive Summary

Current military operations frequently target hostile personnel at night in otherwise neutral or friendly areas. This environment requires soldiers to positively announce the presence of military forces to any local inhabitants which approach their positions, a process which invariably reveals the soldier's position, increasing the threat to both the soldier and local inhabitant by increasing tension. The environment also demands assault forces audibly instruct building occupants to exit the building before entry, and confirm hostile intent before firing upon targets with rifles or aircraft. These actions place military personnel at increased risk and reduce combat effectiveness. The Remote Escalation/De-escalation and Surveillance System (REDSS) will reduce this threat by removing the source of visual and audible cues from the soldier and placing it on a robot located several meters away, and will serve as a surveillance system to detect and confirm hostile intent in otherwise obscured areas. REDSS is a lightweight, cargo pocket portable stationary robot which can be thrown or dropped into a position, and can deploy white light to disorient targets designated by IR lasers, announce audible cues to building occupants, and serve as a remote surveillance camera. Operators control REDSS in an "Eyes Up" configuration by using rifle-mounted IR lasers to designate targets and a simple rifle-mounted remote for basic control, and an Android application for video streaming. As of December 2016, REDSS is a complete prototype with the ability to track infrared lasers at 20m on slow moving targets with a range to the operator of 50m, can transmit video stream to a mobile application, and can deploy a white light, red laser, or IR illumination under command. Development costs were $1000 with the completed prototype totaling $592.49 in parts.

Table of Contents

Remote Escalation/De-Escalation and Surveillance System (REDSS)

1. Introduction

The Remote Escalation/De-escalation and Surveillance System (REDSS) is a cargo pocket sized stationary robot which tracks an infrared laser and deploys a white light under command to disorient personnel approaching military positions, and can serve as a remote video surveillance system. Construction of the prototype cost $1000, with the parts on the robot and remote accounting for $592.49. The remaining $400 was spent replacing broken parts, buying spares, or on parts not used.

1.1 Objective

REDSS will serve to increase safety and margin for error in combat environments for both unarmed civilians and armed military forces, reducing the threat to soldiers and thereby increasing the tolerance for civilian actions near military operations. REDSS will accomplish this through four operational modes:

A. REDSS will reduce tension in precision nighttime combat missions in civilian areas by removing the source of visible light and audio cues from the soldier and placing it on a remote robot. In this mode, REDSS will de-escalate a potentially hostile situation by announcing the presence of military forces without revealing the position of the soldiers.

B. REDSS will function as part of an escalation of force procedure, allowing military forces to give instructions while remaining behind cover. REDSS will be thrown near a target building and remotely controlled to relay audible instructions to building occupants, and will provide a video feed enabling operators to visually inspect personnel departing the building to confirm or deny the presence of weapons.

C. REDSS will be placed in areas which could provide tactical advantages to enemy forces, and will serve as a remote surveillance camera.

REDSS differs from traditional robots by using an “Eyes Up” control concept. REDSS tracks a pulsed infrared (IR) laser emitted from a rifle-mounted remote control and provides visual feedback to operators using a red laser mounted on the robot. These innovations will reduce operator distraction and maintain combat effectiveness. Video streams are accessed by an Android mobile application, compatible with most Android platforms.

1.2 Motivation

The primary motivation is to provide an additional means for military forces to control potentially dangerous encounters without compromising their own safety, thereby increasing the tolerance for civilian actions before lethal force becomes necessary. Current escalation and de-escalation tools, such as the GLAREMOUT laser dazzler, reveal the operator’s position when used and increase tension [1]. Other tools, such as the BAGL, require limited air support resources [2]. REDSS strikes a balance between the safety cost when using man-portable devices, and the monetary cost of vehicle-mounted devices.

1.3 <u>Background</u>

Although there are no current products that match the REDSS functionality, each component within REDSS has a counterpart in the current market. In some areas of the United States, police officers wear portable cameras which document the execution of their duties. The cameras, such as the Axon Body 2, are ruggedized to survive drops and impact, contain Wi-Fi connectivity for remote activation and streaming, and have 12 hours of battery endurance [4]. While these cameras don't serve the exact same functionality as REDSS, they are designed with similar constraints and suggest the feasibility of the technological goals. Laser detection and tracking is used to provide targeting guidance to weapons, and to provide control inputs to computers in the civilian market. Military weapons systems use coded laser signals to improve noise tolerance and prevent spoofing, while civilian systems use simple brightness thresholds and movement detection to track lasers [5], [6]. The military systems use purpose-built electronic hardware for laser detection, specifically focal plane arrays and state machines for code detection, while civilian systems use commercially available CCD cameras paired with traditional processors [6], [7].

2. Project Description and Goals

The REDSS consists of two distinct segments; the user segment and the REDSS robot segment. The system view from the operator perspective is shown in Figure 1. The operator interacts with the system through a physical hardware remote mounted on a rifle, which uses an IR laser to identify targets for the REDSS robot and provides basic functionality such as directional control and white light activation using buttons. Video streams are accessed by the Android application. The robot segment contains a camera for both surveillance and laser-tracking, a white light dazzler to disorient approaching personnel, a speaker for audible warnings, IR illumination, a red laser to confirm robot aim and de-escalate in a discrete manner, and all the necessary hardware to enable the previously described devices.

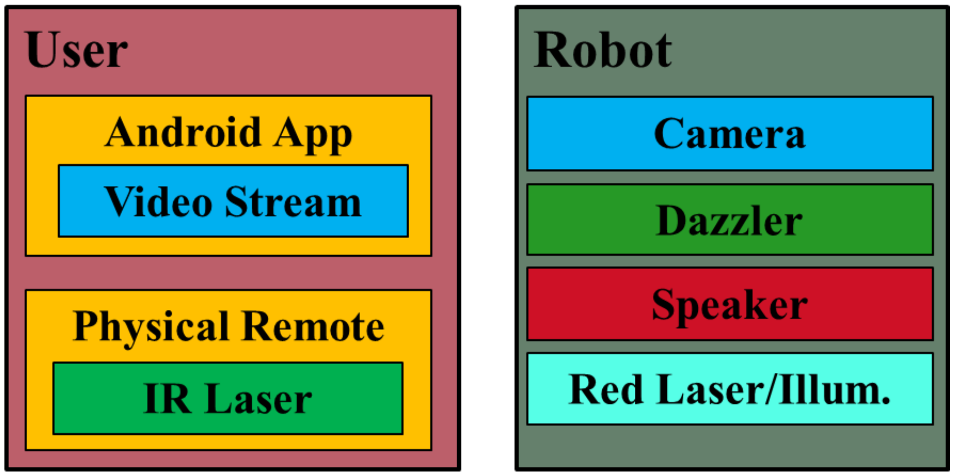


Figure 1. REDSS block diagram from the user perspective

The system's goals are as follows:

- Track an IR laser to aim a white light dazzler at a target.
- Fit in a military uniform cargo pocket.
- Survive being thrown.
- Announce audible warnings at a volume sufficient to penetrate walls.
- Operate for a minimum of 5 hours when in a laser tracking mode.
- Operate for a minimum of 8 hours as a surveillance camera.

- Reduce operator distraction through remote design, motion detection, and visual feedback mechanisms.

3. Technical Specifications & Verification

The REDSS overall performance requirements are shown in Table I. Specific expectations for the processing unit and peripheral components are shown in Table II and Table III. As REDSS is intended for military use, all electronic hardware must be military-grade. Any lasers are power limited at 5mW for safety.

TABLE I SYSTEM SPECIFICATIONS

Feature	Design Specification	Achieved Specification
Dimension	$< 17.8 \times 19 \times 10.1$ cm^3 (Cargo pocket)	20.4 x 10.4 x 20.2 cm^3
Weight	500 g - 1 kg	650 g
Operating Endurance	> 5 hours (Laser tracking mode) > 8hours (Surveillance mode)	> 5 hours (Laser tracking mode) > 8hours (Surveillance mode)
Impact Endurance	> Survive drop from 3.2 m	Not Targeted
Aiming Accuracy	< 0.5 Degrees	1 Degree
Target Distance	> 50 m	20 m
Rotational Speed	> 40 Degrees per Second	40 Degrees per Second
Speaker Range	> 25 m	25 m
Camera Height	> 16 cm	16 cm
Interface	Hardware Remote and Android Mobile App with Video Stream	Hardware Remote and Android Mobile App with Video Stream

TABLE II PROCESSOR SPECIFICATIONS

Feature	Design Specification	Achieved Specification
Calculation of motor angle	< 15 ms	< 15 ms
Communication	WIFI (802.11n)	WIFI (802.11n), Proprietary 2.4GHz
Image Refresh Rate	> 60 fps	2 fps
Image resolution	480p – 1080p	640x480
Audio Capability	Required	Present
Location	GPS	GPS

TABLE III PERIPHERALS SPECIFICATIONS

Peripherals	**Feature**	**Design Specification**	**Achieved Specification**
Camera	Angular Resolution	< 0.4 degrees	0.3 degrees
	Detectable wavelength	Visible light, Infrared light	Visible light, Infrared light
Pan Motor	Angular velocity	60 - 120 degrees/sec	60 degrees/sec
	Active range	0 - 360 degrees	0-180 degrees
Tilt Motor	Angular velocity	20 - 60 degrees/sec	40 degrees/sec
	Active range	10 - 170 degrees	0-180 degrees
Pan/Tilt Module	Maximum payload weight	300 g	400g
IR Laser	Wavelength	850 – 900 nm	854 nm
	Power	< 5 mW	5 mW
	Weight	< 50 g	40 g
Dazzler	Diameter	5 – 10 cm	5 cm
	Intensity	500 lumen	500 lumen
	Weight	< 100 g	100 g

4. Design Approach and Details

4.1 Design Approach

Mechanical Design

The REDSS robot segment is entirely self-contained and supports a camera, dazzler, illumination, and confirmation laser at a height sufficient to reduce terrain interference. A picture of the implemented robot segment is shown in Figure 2. The primary emplacement method for REDSS is being thrown by an operator, and a production model must therefore be mechanically rugged and self-righting while still meeting size and camera height specifications. A production model would need a stowed configuration for portability, and one possible arrangement is shown in Figure 3. Mechanical design constraints were not an emphasis on the prototype and mechanical considerations were incidental to the electrical system. Mechanical objectives were not achieved but were an integral part of the design process, and the completed enclosure is meant to serve as a proof-of-concept for a production configuration.

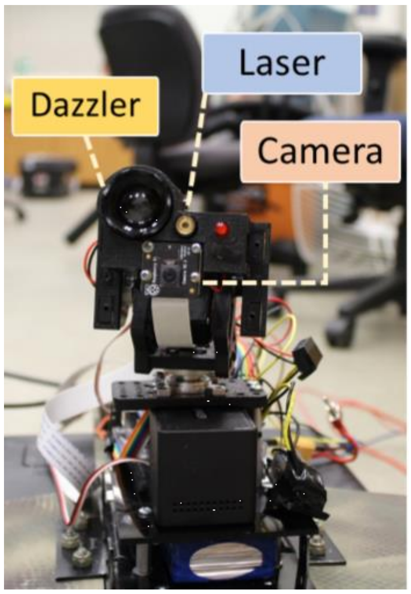


Figure 2. REDSS Robot in the deployed state.

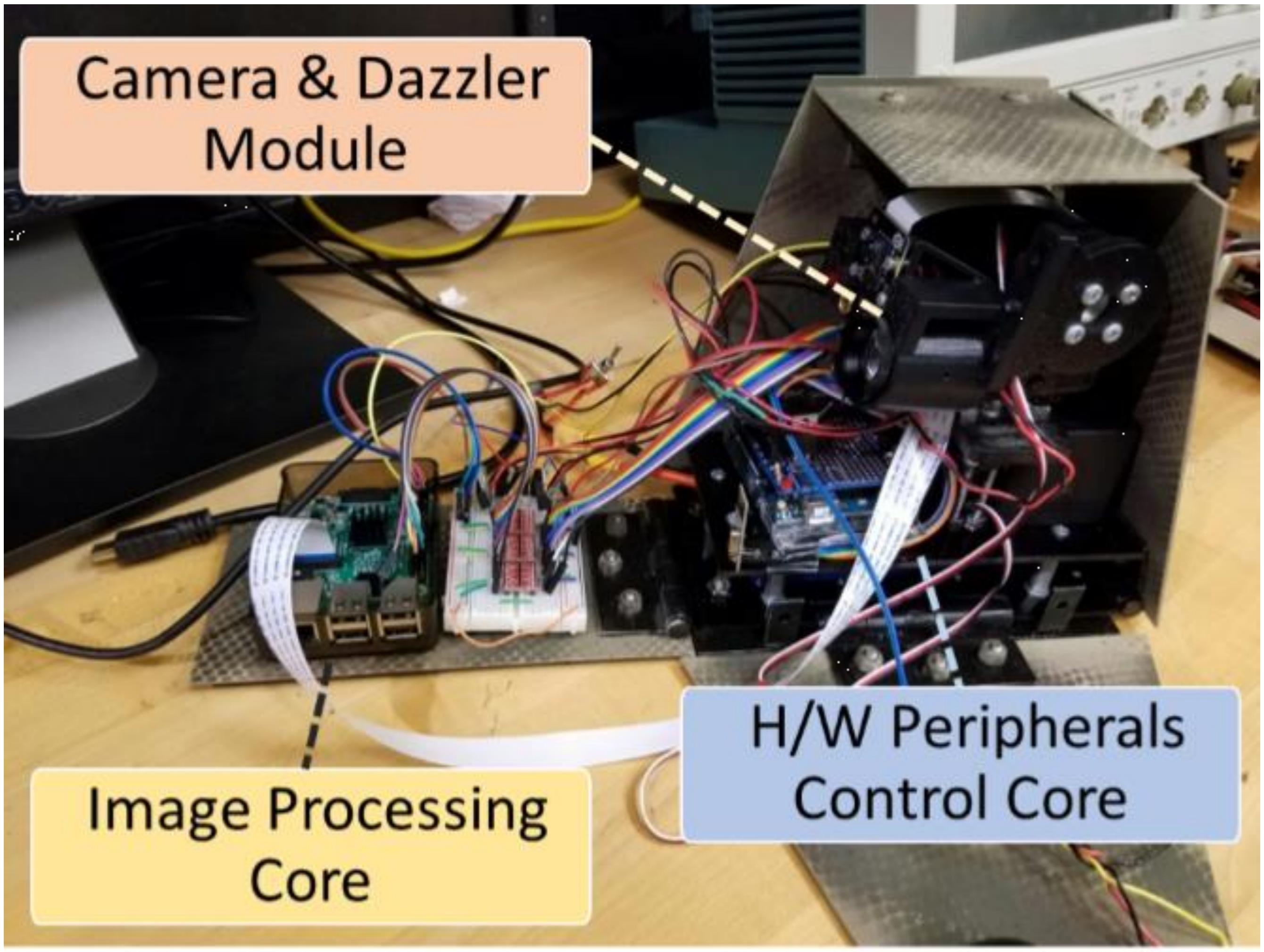


Figure 3. Stowed configuration of the REDSS robot.

Electronics

As described previously, the REDSS consists of user and robot segments. The overall system architecture is shown in Figure 4. Communication between the rifle-mounted remote and REDSS robot system is achieved using an off-the-shelf, proprietary 2.4GHz communication module. Commands are relayed between the Arduino Micro in the remote and the Arduino Uno on the robot, with commands passed to the Rasberry Pi over I^{2}C or GPIO when necessary. The original design used the Raspberry Pi's Bluetooth capability for remote communication, however, the development time required to configure the Bluetooth was much higher than expected, and was discarded in favor of a simpler system using the Arduino modules. The Android application connects to the system by 802.11n WiFi, as planned in the original design documents.

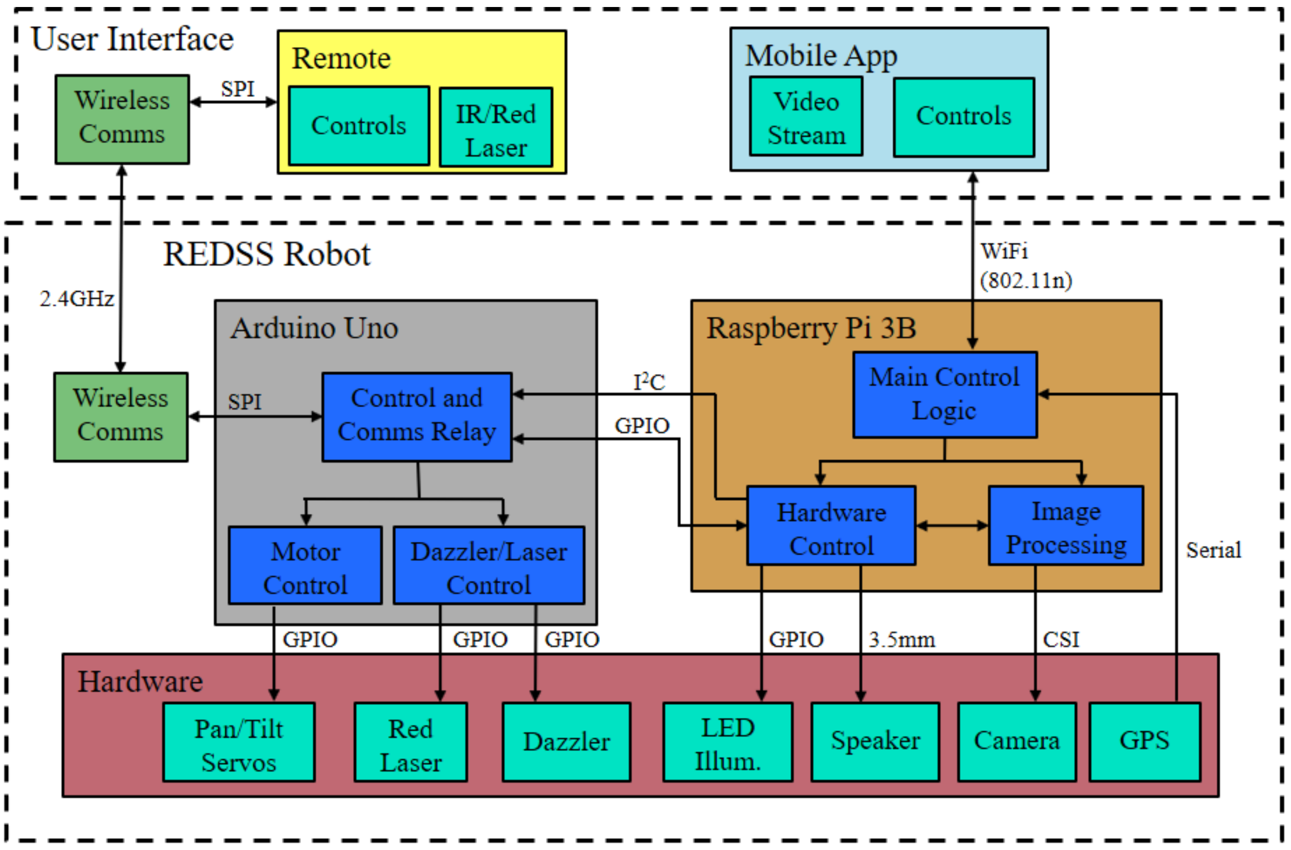


Figure 4. Detailed block diagram of the REDSS architecture.

Robot Segment

The primary processing unit, the Raspberry Pi 3B, is not well suited for hardware control. To overcome this challenge, the robot uses a front-seat/back-seat architecture, assigning any hardware control tasks to an Arduino microcontroller while the Raspberry Pi handles higher-level functions such as image processing. A detailed description of the REDSS robot design is shown below in Figure 5. Power is supplied by a 55 Wh 3-cell lithium ion battery with a nominal voltage of 11.1V.

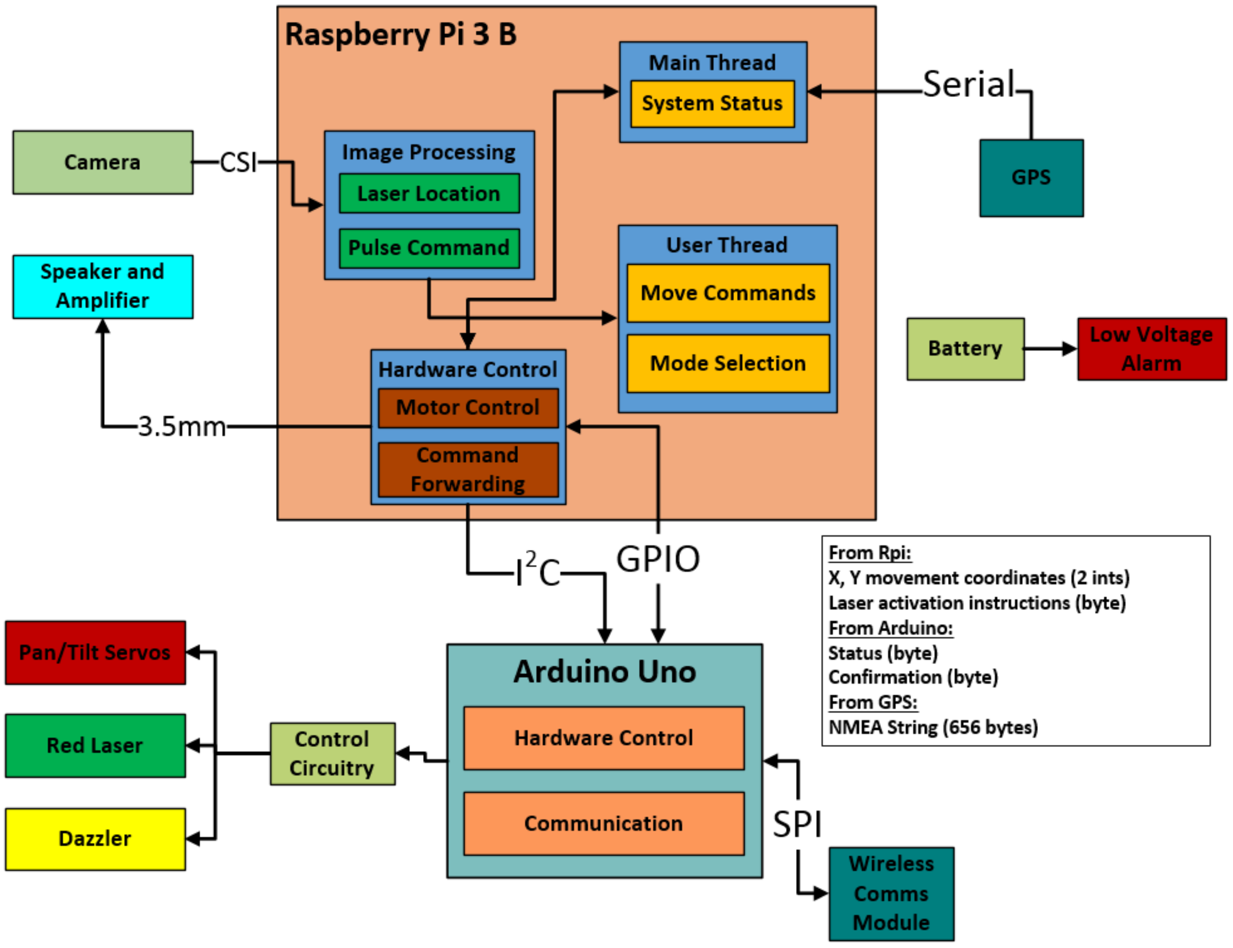


Figure 5. Detailed block diagram of the REDSS Robot subsystem.

REDSS Peripherals are controlled by the Raspberry Pi or Arduino Uno using simple circuitry. Lasers are regulated using constant current linear power supplies, with other peripherals controlled using NMOS circuits. Power is supplied by the unregulated 11.1V nominal battery voltage wherever possible, an effort to lower current demands on the regulator. Devices which require a regulated voltage are fed by a 6A, 5V switching regulator. Pan and tilt are accomplished using 180 degree servos, with the tilt servo flipping over to permit a full 360 degree pan range. Details of peripheral configuration are shown in Figure 6.

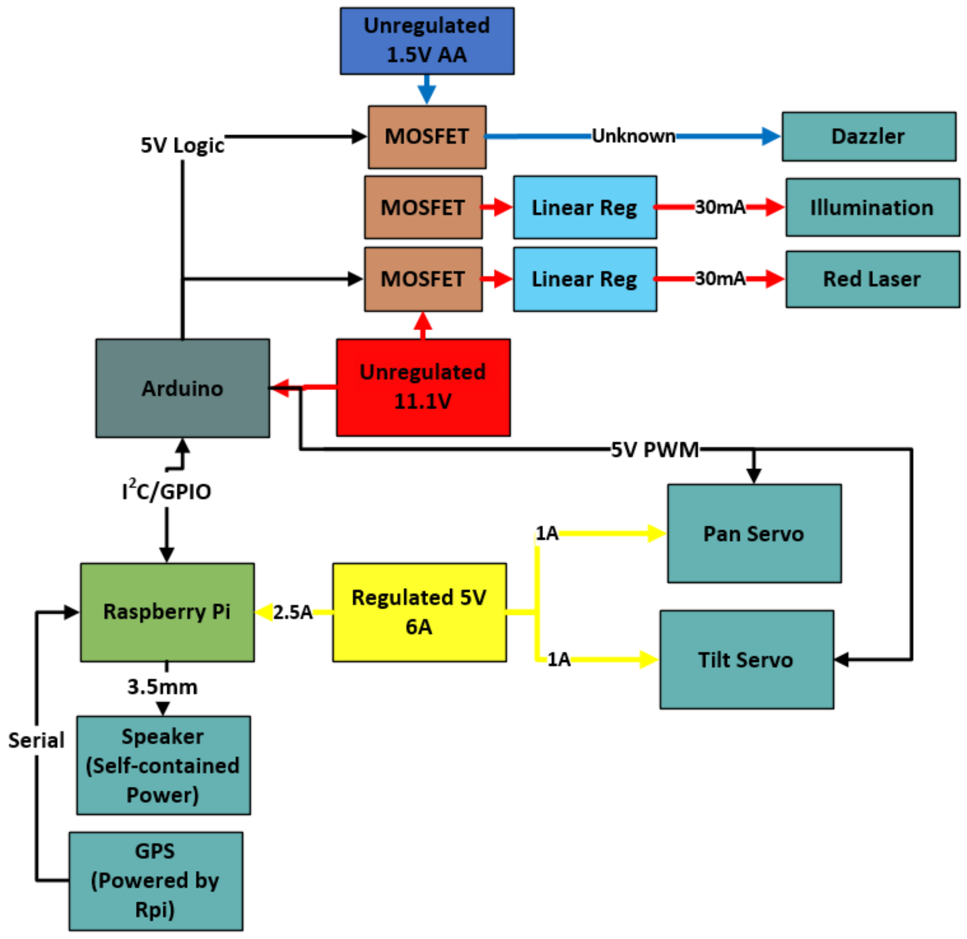


Figure 6. Peripheral and power configuration on the REDSS robot segment.

Image Processing

The REDSS robot uses a camera to track the laser, stream a video feed to the user, and provide feedback to the motors. Images are processed on the Raspberry Pi 3 using OpenCV software. Laser detection and tracking uses a simple, on-demand pulse detection algorithm, which increases resistance to noise and outperforms intensity detection algorithms [1], [2]. The image processing algorithm is shown in Figure 7. Processing complexity was limited by the low performance of the Raspberry Pi, and even the simplified algorithm achieved only 5 fps with jitter of 3 fps. Any vibration within the system corrupts the reference frame, and the on-demand pulse scheme allows the algorithm to capture a new reference frame whenever the previous one is suspected to be corrupt.

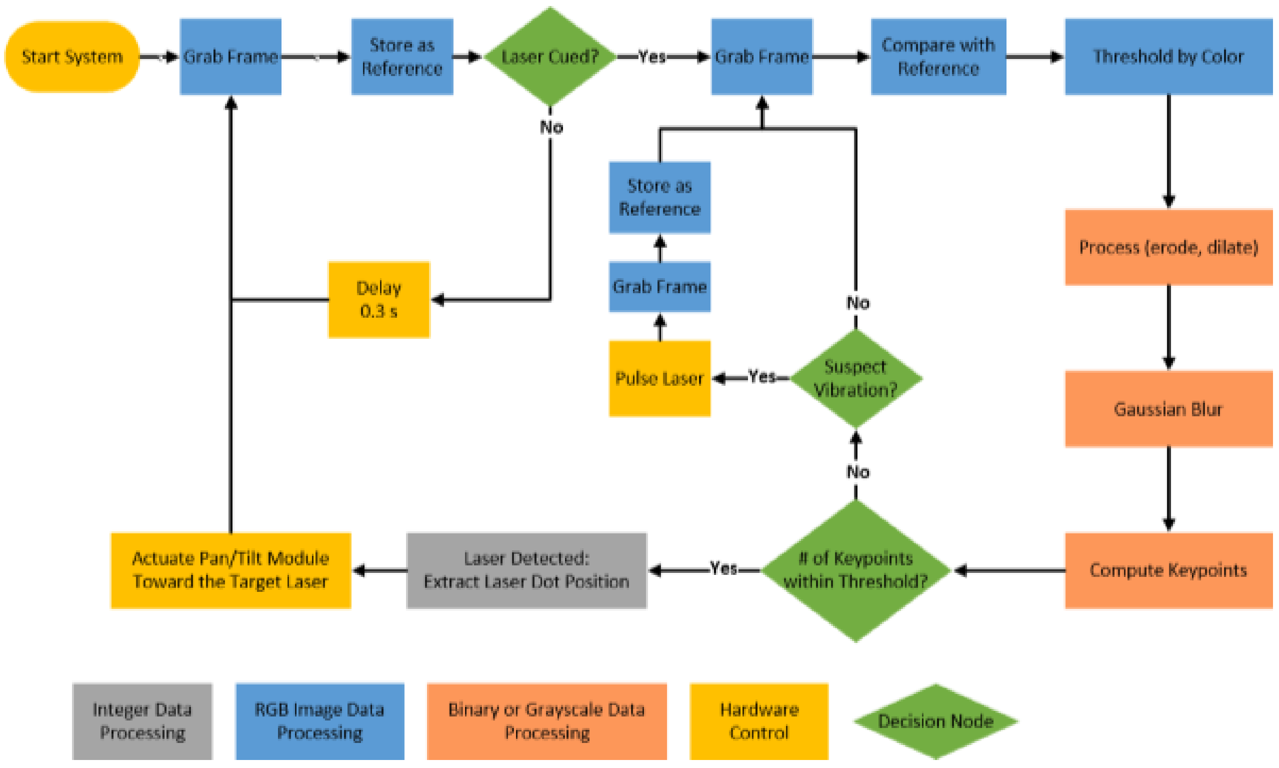


Figure 7. Flowchart of the laser tracking algorithm.

The design specified a motion detection feature for surveillance mode, a feature meant to reduce operator workload. Motion tracking is not present in the final design due to time constraints.

User Interface

The operator interacts with REDSS through an “Eyes Up” hardware remote, and accesses the video stream through an Android mobile application. The hardware remote uses an off-the-shelf 2.4GHz module for communication, while the Android application uses WiFi 802.11n.

The hardware remote consists of a red laser, IR laser, and buttons. The operator designates a target using the IR laser, which the robot then detects and uses to aim its payload. If the robot cannot find the laser, it will de-activate the remote laser using the wireless link, grab a new reference frame, and reactivate the laser. The REDSS robot confirms aim for the operator by activating a red laser, which the operator can adjust using the joystick. When commanded, the REDSS robot deploys the dazzler at the target in high brightness or pulse modes. Minor aim corrections are made using the remote’s buttons. A disassembled remote is shown below in Figure 8, with an example or rifle attachment and use in Figure 9.

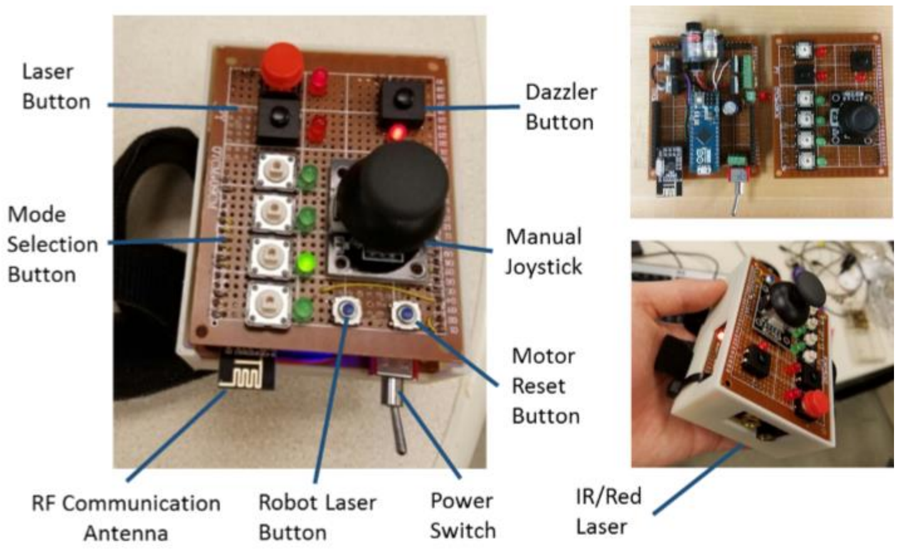


Figure 8. Disassembled REDSS hardware remote with component labels.

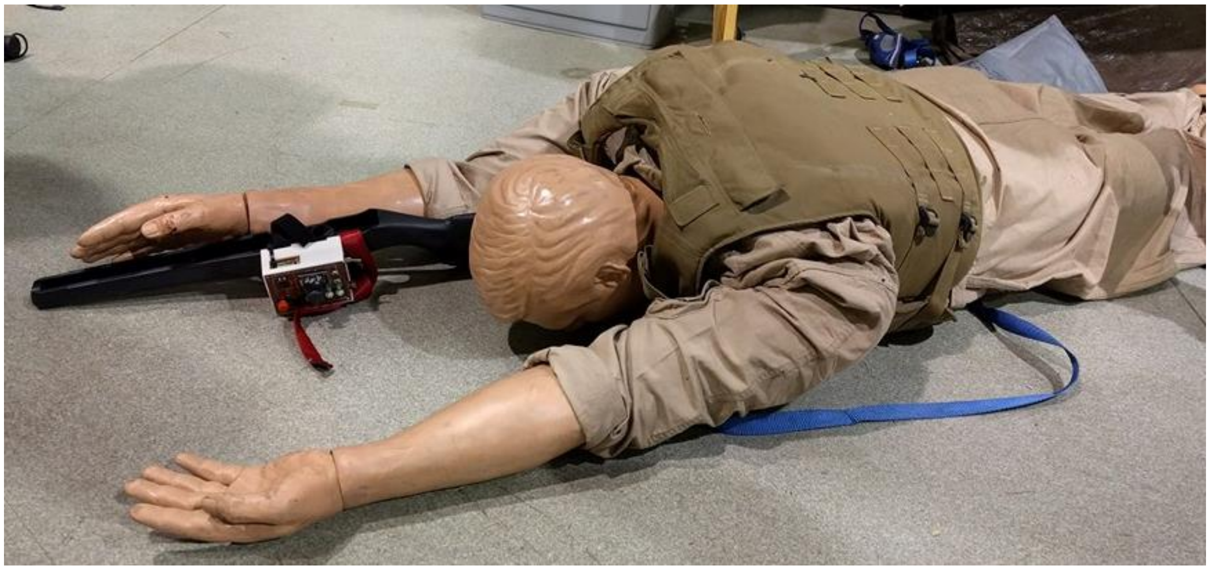


Figure 9. An operator holding a rifle stock equipped with the REDSS remote. The operator fell asleep and will not pass his patrol.

The mobile application provides access to the video stream and control over system mode. The original design called for advanced functionality within the application, but time limits prevented the development of any advanced utilities. Figure 10 shows the video feed, used for both surveillance and escalation of force procedures. The mobile application is targeted toward generic Android devices to ensure compatibility with products such as the Harris RF-3590-RT Ruggedized Tablet [8].

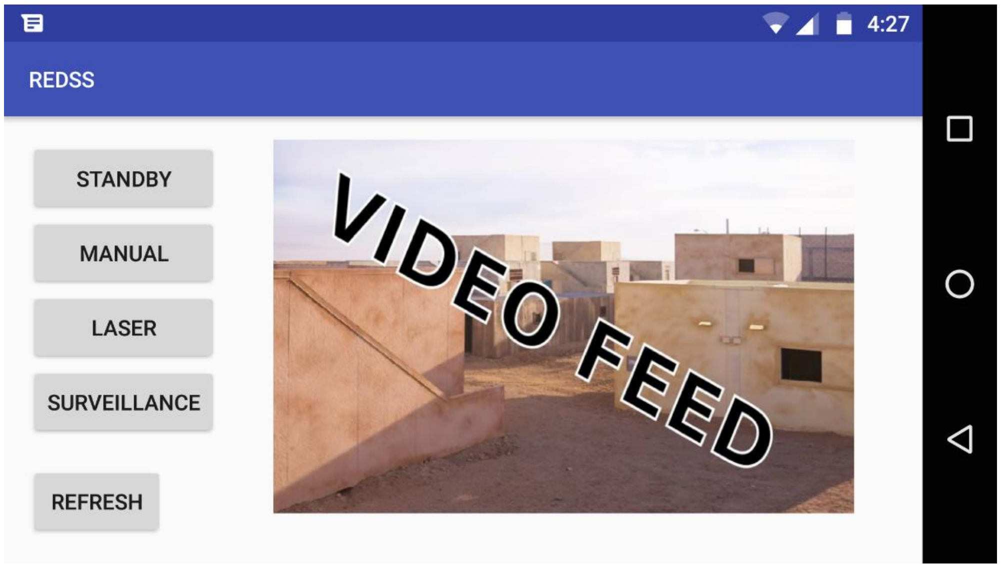


Figure 10. Video feed display on the REDSS mobile application.

Future Work

The system is limited by the Raspberry Pi's processor speed and any future work should begin with redesigning the back-seat architecture to use an NVIDIA Jetson. The parallel hardware on the Jetson will increase the framerate considerably and permit the development of a more complex, noise and vibration resistant laser tracking algorithm. Currently the system fails under even minor vibration, and the image processing must pause until vibrations have ceased.

A higher framerate laser tracking system will expose the impulse movement of the servos as a major system limitation. The servos should be replaced with DC stepper motors and a proper feedback and control system implemented to assure rapid but critically damped movement.

The robot can be physically compressed with the replacement of off-the-shelf modules with custom built electronics and PCBs. Once sufficiently compressed, a mechanical team can repackage the system into a truly cargo pocket-sized housing and harden it against impact, thereby making the robot portable and throw-able.

4.2 Codes and Standards

Raspberry Pi 3

- Open source hardware and software design developed by the Raspberry Pi Foundation with a large open source community
- Low-cost hardware as the foundation for a computer running the Raspbian operating system based on Debian
- Containing the Broadcom BCM2837 64bit ARMv7 Quad Core Processor powered Single Board Computer running at 1.2GHz
- GPU of Broadcom VideoCore IV
- 4 USB ports, 40 GPIO pins, which provide I^2C and SPI functionality
- 5V device with 3.3V outputs

Arduino Micro

- Features USART (Universal synchronous and Asynchronous Serial Receiver and Transmitter), I^2C, and SPI
- Based on ATmega32U4 board
- Allowing 20 digital input/outputs (7 PWM outputs), 12 analog inputs, and USB connection
- Compiled by C/C++
- Large open source community

I^2C (Inter-Integrated Circuit)

- Low-cost network to interconnect peripheral devices
- Master-slave relationship of Raspberry Pi and Arduino Micro
- Conflict-free connection of varied devices [9]
- 100kbps in standard and 400kbps in fast mode [10]

SPI (Serial Peripheral Interface)

- 4-wire serial bus to allow for short distance communication, including two input signals, one output, and a “chip select” signal [11]
- Communication between Raspberry Pi and Arduino

GPS

- System to report position to operator for alarm protocol or tamper-evident features
- Built around MTK3339 chipset to track up to 22 satellites on 66 channels with receiver sensitivity of -165dBm [12]

Wi-Fi

- Built-in Broadcom BCM43438 module in Raspberry Pi 3
- 2.4GHz 802.11n wireless LAN [13]

CSI (Camera Serial Interface)

- Low power 2.5 Gbps, 15-pin, processor to Pi NoIR camera interface

4.3 Constraints, Alternatives, and Tradeoffs

The most significant constraint for the prototype REDSS development was time, while the design constraints were size, weight, power, and image processing capability. The prototype was built as a proof-of-concept, and ease of use and reliability rather than absolute performance was the priority when selecting parts

The main processing unit required a powerful image processing capability, low power requirement, and peripheral connectivity. The two most capable commercial options available were the TI Beaglebone and the Raspberry Pi. The Raspberry Pi has less capable IO mechanisms compared to the Beaglebone, but contains native support for high-speed camera interfaces in the form of Camera Serial Interface (CSI) and an on-board Graphics Processing Unit. The insufficient IO of the Raspberry Pi was augmented by a Front-seat/Back-seat architecture with a microcontroller. There are four Raspberry Pi models available which contain a GPU and the CSI interface. Table IV shows the decision process, which ultimately chose the Raspberry Pi 3B. The original design plans allotted time to map the image processing algorithm onto the GPU and the decision process emphasized processors with an onboard GPU. However, time constraints prevented GPU integration.

TABLE IV DECISION MATRIX FOR THE RASPBERRY PI MODELS

Features	**RPi 1A+**	**RPi 1B+**	**RPi 2B**	**RPi 3B**
Power Consumption	**4**	3	1	2
Processor Speed Margin	1	1	3	**4**
GPU Capability	1	1	1	**4**
Memory Size	1	2	**4**	**4**
Price	**4**	3	2	3
Total	**11**	**10**	**11**	**17**

Hardware control is handled by the “Front-seat” microcontroller. The Arduino Micro was initially chosen due to its low power consumption and large user community, however the Arduino Uno proved to be much easier to use and represented significant time savings during development. The Arduino Uno was chosen for the REDSS robot segment, as size constraints were less critical and overall microcontroller demands were much higher, while the remote, which had more significant size constraints but lower processing demands, used an Arduino Micro

The system is intended for the military, and parts were chosen for military supply chain compatibility wherever possible. A production model would use standard military batteries, such as the Harris Falcon III radio battery. However, the Falcon III radio battery was beyond the budget constraints of the project, and the prototype simulates the Falcon III battery at a much lower cost with an off-the-shelf 55Wh battery.

Military lasers operate at 25mW, beyond eye-safe levels. This level was far too high for safe development and demonstration, and the lasers were limited to 5mW using constant current regulators. This limited system range, and specifications were modified accordingly.

5. Schedule, Tasks, and Milestones

The timeline for the project is attached as a GANTT chart in Appendix A and B. Per our sponsor's request, we conducted a Top Level Design Review, Midpoint Design Review, and Test Readiness Review.

Table V contains a breakdown of significant tasks, assigned engineers, and risk level. Image processing was the riskiest task due to the team inexperience and processing requirements, and required three engineers to complete.

TABLE V TASK DIVISION AND RISK LEVELS

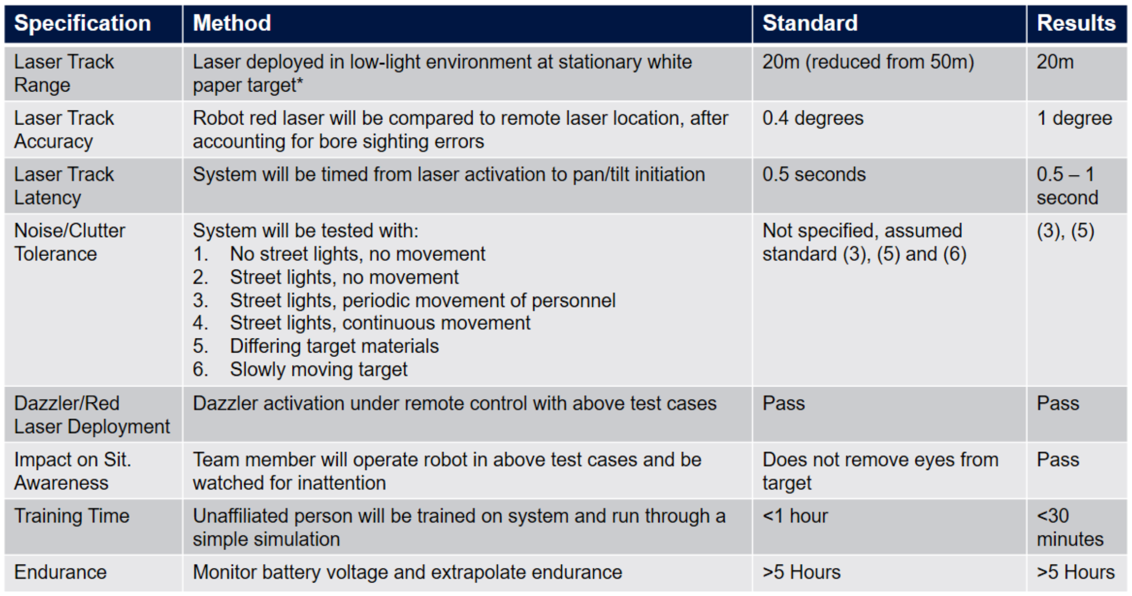
Task Name	**Task Lead**	**Risk Level**
Documentation and Presentations	All (Lead by Mike)	Low
User Interface Design/Implementation	Josh	High
Hardware Integration	Gyu Cheol, Mike, Satoshi, Seong Ho	Medium
Image Processing	Gyu Cheol, Seong Ho Mike	Highest
Peripherals (Motors, Laser, Dazzler) Controls	Gyuc Cheol, Seong Ho	Medium
Mechanical Design	Satoshi	Low
Power System	Satoshi	Low

6. Project Demonstration

6.1 Prototype Tests

The completed prototype was tested in conditions meant to approximate the target employment environment. The system was tested at night near street lights and during a rain storm, which approximates the dust endemic to most of the Middle East. The original specifications call for a laser tracking range of 50m, however, the specification targeted a 25mW laser, 5 times more powerful than the REDSS eye-safe laser. Therefore, the laser tracking range standard was reduced to 20m. Physical design specifications were not targeted and the system was not drop-tested. Test methods, standards, and results are shown below in Table VI, VII and VIII.

TABLE VI DE-ESCALATION (LASER TRACKING) MODE TEST PLAN AND RESULTS



Specification	Method	Standard	Results
Laser Track Range	Laser deployed in low-light environment at stationary white paper target*	20m (reduced from 50m)	20m
Laser Track Accuracy	Robot red laser will be compared to remote laser location, after accounting for bore sighting errors	0.4 degrees	1 degree
Laser Track Latency	System will be timed from laser activation to pan/tilt initiation	0.5 seconds	0.5 – 1 second
Noise/Clutter Tolerance	System will be tested with: 1. No street lights, no movement 2. Street lights, no movement 3. Street lights, periodic movement of personnel 4. Street lights, continuous movement 5. Differing target materials 6. Slowly moving target	Not specified, assumed standard (3), (5) and (6)	(3), (5)
Dazzler/Red Laser Deployment	Dazzler activation under remote control with above test cases	Pass	Pass
Impact on Sit. Awareness	Team member will operate robot in above test cases and be watched for inattention	Does not remove eyes from target	Pass
Training Time	Unaffiliated person will be trained on system and run through a simple simulation	<1 hour	<30 minutes
Endurance	Monitor battery voltage and extrapolate endurance	>5 Hours	>5 Hours

TABLE VII ESCALATION (MANUAL CONTROL) MODE TEST PLAN AND RESULTS

Specification	Method	Standard	Results
Manual Control Range	System will be tested at varying ranges between remote and robot	15m	20m
Speaker Range	Robot will announce instructions and team members will listen at varying ranges	25m	25m
Dazzler Deployment	Dazzler activation under remote control with above test cases	Pass	Pass
Remote Control Through Brick Wall	System will be tested with a 10m range and brick wall between remote and robot	Pass	Pass
Movement Latency	Time to pan/tilt robot will be tested	40 degrees per second	60 degrees per second
Operator Cover	Team member will operate robot without line of sight and use mobile app video stream for feedback	Does not break cover	Did not break cover
Training Time	Unaffiliated person will be trained on system and run through a simple simulation	<1 hour	<30 minutes

TABLE VIII SURVEILLANCE MODE TEST PLAN AND RESULTS

Specification	Method	Standard	Results
Video Stream Range	System will be tested at varying ranges between receiving platform and robot	50m	50m
Motion Detection*	System will be view increasing levels of motion and monitored for alarm signal	Man-sized object	Not Implemented
Operator Cover	Team member will operate robot without line of sight and use mobile app stream for feedback	Does not break cover	Did not break cover
Training Time	Unaffiliated person will be trained on system and run through a simple simulation	<1 hour	<30 minutes
Endurance	Monitor battery voltage and extrapolate endurance	>8 hours	>8 hours

6.2 Live Demonstration

All of the REDSS operational modes were demonstrated at the expo with an emphasis on the de-escalation (laser tracking) mode. The robot was placed on a table with a tether to a monitor and the remote was mounted on a rifle buttstock. Patrons at the expo were encouraged to aim the rifle stock at a wall and activate the IR laser and robot dazzler. The robot flawlessly tracked the laser at 1 – 2 fps, and responded to all user commands. The attached monitor displayed the image processing steps with scaled down output images. Surveillance mode was configured for an automatic pan, and patrons could view the camera feed on an Android phone.

The demonstrated functions were:

- Laser tracking
- Dazzler deployment
- Manual Control
- Video Stream
- Surveillance mode
- Image processing algorithm

More elaborate employment environments were shown on a looping video [19]. Longer ranges, rainy and dark environments, and changing targets were displayed to provide a better understanding of system capabilities.

7. Marketing and Cost Analysis

7.1 <u>Marketing Analysis</u>

The target market is the government, specifically for military personnel in combat environments. Although no products with identical functionality exist, the surveillance function of the system is comparable to a standard surveillance system. A commercial surveillance system with wireless communication and video feed can cost approximately $200, while advanced designs with recording capabilities cost nearly $500 [14]. Similarly, a motion tracking security light that follows movement in a 220 degree cone costs roughly $100 without a camera and $200 with a camera installed [15]. A surveillance trailer can cost upwards of $42,000 [16]. Though there are several products available that fulfill parts of the proposed device, none have the combined functionality or a laser tracking capability which creates a unique market for the proposed product.

7.2 <u>Cost Analysis</u>

The total cost materials for developing a prototype REDSS system was $1000 as shown in Appendix C. The total cost of the parts on the prototype was $592.49, with the parts list shown in Table IX. Many parts were not used in the final product due to low performance, complexity, or failure. Spares of critical items were ordered whenever budget allowed.

TABLE IX DEVELOPMENT EQUIPMENT COSTS

Part	Number	Cost Each	Cost Total
Raspberry Pi 3B	1	$39.95	$39.95
Arduino Micro	1	$24.95	$24.95
Arduino Uno	1	$24.95	$24.95
Pan tilt system	1	$45.99	$45.99
Servo (HS-485HB)	2	$16.99	$33.98
NOIR Camera	1	$29.95	$29.95
Li-ion Battery (5.2 AH)	1	$34.80	$34.80
1/8" x 12" x 12" aluminum sheet metal	1	$28.56	$28.56
Surface Mount Hinge	8	$4.83	$38.64
Stainless Steel Hex Nut pack 100	1	$3.83	$3.83
Stainless Steel Flat Head Phillips Machine Screw	1	$5.80	$5.80
IR Laser Diode	1	$72.24	$144.48
Red Laser Diode	2	$23.71	$47.42
Arduino Protoboard	1	$9.99	$9.99
LED Torch	1	$15.00	$15.00
Level Shifters	3	$3.95	$11.85
CSI Cable	1	$9.95	$9.95
Voltage Regulator	1	$19.95	$19.95
MOSFETS	20	$0.50	$10
LM317 Linear Regulators	5	$0.50	$2.50
Speaker	1	$9.95	$9.95
Parts Total			**$592.49**

The development costs were estimated using the average annual pay of Electrical Engineers as $57,030 [17]. This converts to roughly $30 an hour which was used to estimate development costs. The total labor cost is $17,280.00 and is summarized in Table X below.

TABLE X COST ANALYSIS OF LABOR

Project Component	Labor Hours	Labor cost
User interface	160	$4,800.00
Pan tilt control mechanism	144	$4,320.00
Battery monitoring	40	$1,200.00
Image processing	128	$3,840.00
Mechanical construction	56	$1,680.00
Documentation	48	$1,440.00
Labor Total	**576**	**$17,280.00**
Parts Total		**$592.49**
Grand Total		**$17,872.49**

Several assumptions are made in the production calculation: the components of the REDSS device can be obtained with a 20% discount due to bulk ordering and custom manufacturing, labor costs remain consistent, fringe costs are 30% of total labor, overhead costs are 120% of the total material and labor costs, and sales costs are only 3% due to a highly focused market. With those assumptions the REDSS can be sold for approximately $937.06 earning a $35 profit per unit if 5000 units are sold. Details are shown in Table XI below.

TABLE XI PRICE PER UNIT BREAKDOWN

Parts 20% discount	$2,962,450.00
Labor	$17,280.00
Fringe	$5,184.00
Subtotal	**$2,984,914.00**
Overhead	$1,443,759.00
Production Total	$4,428,673.00
Sales	$76,630.30
Amortized Development Costs	$5,000.00
Profit	$175,000.00
Total Production, Sales and Profit	$4,685,303.00
Selling Price	$937.06

Since the REDSS device will be used in a military setting, a more ruggedized version will need to be developed to meet environmental demands. When comparing ruggedized laptop and tablet devices, ruggedized versions cost roughly 300% the cost of an equivalent non ruggedized unit [15],

[18]. Using those estimates a ruggedized version could be developed and sold for $2,818.18 if maintaining a $35 profit per unit.

8. Conclusion

The project ended with a functional REDSS prototype, and can be declared a success. The robot successfully tracked a laser in a degraded environment without any calibration, and performed all the desired operations under remote control without excessive training time. Dozens of people at the expo operated the system effectively with very little instruction, and proved laser tracking and "Eyes Up" control as a valid command scheme. The rifle-mounted remote proved to be a practical method for robot control, and ultimately incorporated more functionality than originally planned. Mode selection and additional peripheral controls on the remote allowed it to replace the mobile application, which limited the impact of the mobile application's delayed development.

The system's utility as a de-escalation and escalation tool in combat environments is questionable. A light infantryman would likely be unable to justify the additional weight of the robot against its small utility. However, the control concept and laser tracking software can be applied to other military robots, and the "Eyes Up" control method could replace or supplement traditional remotes in most military applications.

Despite these successes, there are some significant issues with the system. The software is unstable and frequently crashes. The likely cause is the communication between the Arduino and the Raspberry Pi, but a definitive solution was never found. Additionally, the laser tracking is severely limited by the low processing capability of the Raspberry Pi, and cannot be further refined without upgrading to more robust and capable hardware. Neither of these issues disprove the concepts or in any way indicate the system or any of its underlying concepts are invalid, and further refinement may not be necessary unless a production model is desired.

As a prototype, REDSS is missing some key functions that are required on a production model. The mobile application is not fully developed, and was all but abandoned until the final days of development. Functions that rely on the mobile app, such as the speaker and location reporting, were never fully implemented and exist on the robot only as peripherals and basic software. The mechanical side of REDSS is also incomplete, but this was a deliberate choice during the design process to spare time and resources.

Excluding time as a factor, the most significant impacts on system progress resulted from early and inexperienced design decisions. The team made well-researched but out-of-context decisions during the initial design, which led to the regrettable use of the Raspberry Pi and servo motors. In retrospect, the team should not have taken hobbyist reviews of open-source systems as an indicator of capability or quality, and should have used professional grade material wherever possible. The Raspberry Pi performs well for hobbyist applications where development time or reliability are less critical factors, and the team should've evaluated the NVIDIA Jetson more closely. The additional time required to learn CUDA was known initially and assumed to be greater than the time to learn the Raspberry Pi's nuances. This was not the case, as Raspberry Pi documentation is not conclusive. The servos were chosen as an assumed development shortcut, by eliminating the need for a proper feedback loop. The servos did not have any tuning ability, and when the design required adjustment the servos could not be modified.

Future designs should use adjustable, even if more expensive, parts wherever possible and focus more on proving the critical concepts rather than minor capabilities. Regardless of some inexperienced decisions, the end result was an effective proof-of-concept and closely met most of the design specifications. Further refinement of the system is possible but should focus on the core concept of “Eyes Up” control and not add features.

9. References

[1] B.E. Meyers Advanced Photonics, "Glare Mout," [Online]. Available: http://bemeyers.com/product/glare-mout/. [Accessed 14 April 2016].

[2] M. Weisgerber, "SOCOM Puts Laser Spotlight on AC-130 Gunships," Intercepts, 20 March 2014. [Online]. Available: http://intercepts.defensenews.com/2014/03/socom-puts-laser-spotlight-on-ac-130-gunships/. [Accessed 14 April 2016].

[3] SkyCop, "SkyCop Platform," [Online]. Available: http://skycopvideo.com/skycopplatform. [Accessed 14 April 2016].

[4] Axon, [Online]. Available: http://www.axon.io/products/body2. [Accessed 14 April 2016].

[5] "Coded Spot Tracker (CST)," Northrop Grumman, 2016. [Online]. Available: http://www.northropgrumman.com/Capabilities/CodedSpotTracker/Pages/default.aspx. [Accessed 8 March 2016].

[6] C. Kirstein and H. Muller, "Interaction with a projection screen using a camera-tracked laser pointer," Lausanne, 1998.

[7] J. English, R. White, J. Springer and R. Schneider, "Semi-active laser last pulse logic seeker utilizing a focal plane array". US Patent 6111241 A, 2000.

[8] Harris, "RF-3590-RT Ruggedized Tablet," [Online]. Available: http://rf.harris.com/capabilities/c4isr/ruggedizedtablet.asp. [Accessed 14 April 2016].

[9] "Inter-Integrated Circuit Framework," [Online]. Available: http://www.qnx.com/developers/docs/6.3.2/neutrino/technotes/i2c_framework.html. [Accessed 14 April 2016].

[10] J. Byron, "Raspberry Pi SPI and I2C Turtorial," [Online]. Available: https://learn.sparkfun.com/tutorials/raspberry-pi-spi-and-i2c-tutorial. [Accessed 14 April 2016].

[11] Gordon, "Understanding SPI on the Raspberry Pi," [Online]. Available: https://projects.drogon.net/understanding-spi-on-the-raspberry-pi/. [Accessed 14 April 2016].

[12] "Adafruit Ultimate GPS on a Raspberry Pi," 2012. [Online]. Available: https://blog.adafruit.com/2012/08/28/adafruit-ultimate-gps-on-a-raspberry-pi/. [Accessed 14 April 2016].

[13] Magpi, "Raspberry Pi 3 Specs, Benchmarks," 2016. [Online]. Available: https://www.raspberrypi.org/magpi/raspberry-pi-3-specs-benchmarks/. [Accessed 14 April 2016].

[14] The Home Depot, "Wireless Outdoor iSecurity Camera with 8GB SD Recorder and Night Vision with Remote Viewing," [Online]. Available: www.homedepot.com. [Accessed 14 April 2016].

[15] Amazon.com, "NightWatcher Motion - tracking Motorized LED Flood Light with Color Camera," [Online]. Available: Amazon.com. [Accessed 14 April 2016].

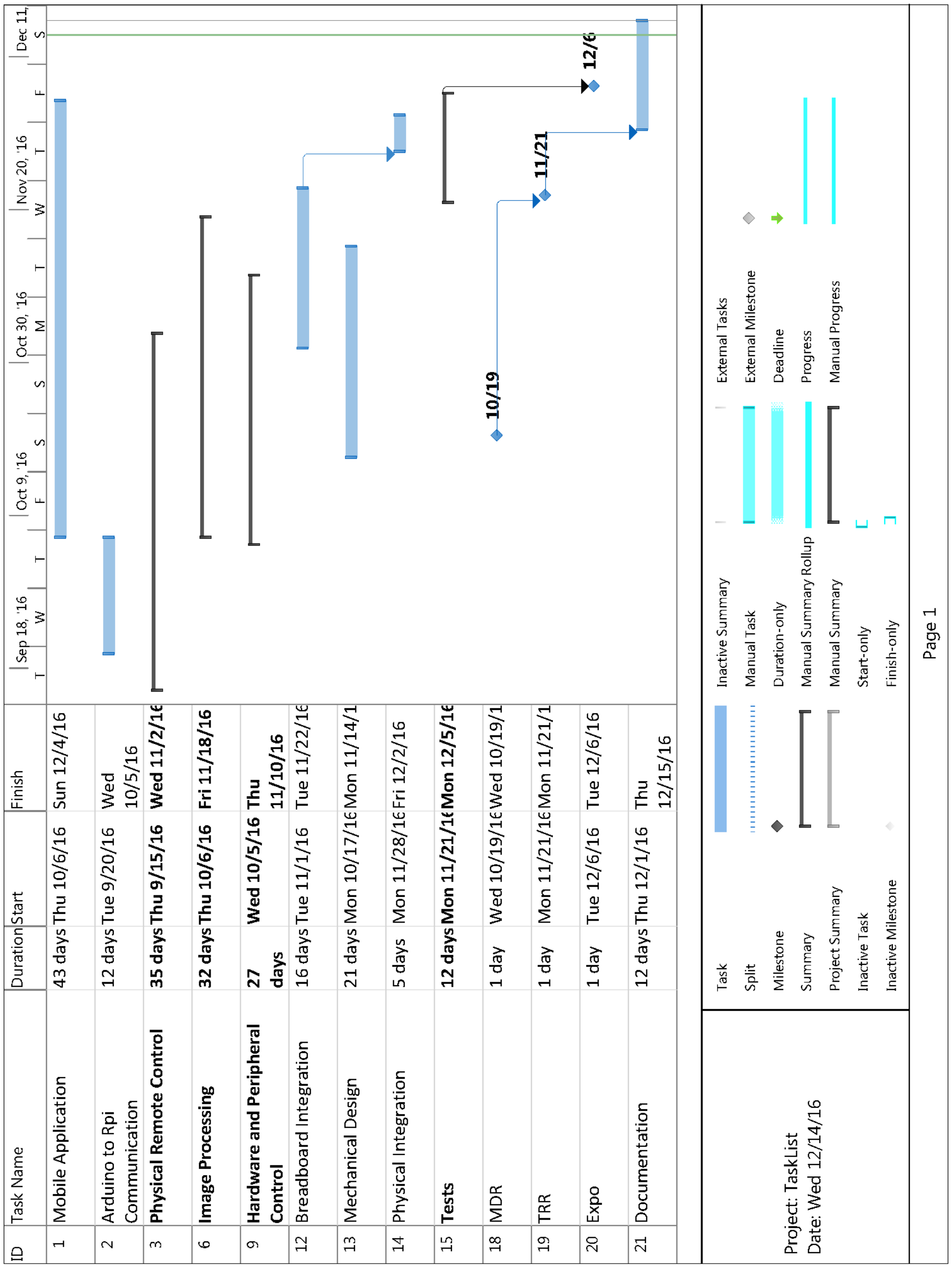
[16] K. Shackleford, Interviewee, Skycop Sales Associate. [Interview]. 14 April 2016.

[17] S. Adams, "The College Degrees With The Highest Starting Salaries in 2015," Forbes, 19 November 2014. [Online]. Available: http://www.forbes.com/sites/susanadams/2014/11/19/the-college-degrees-with-the-highest-starting-salaries-in-2015/#2e8cd52696a2. [Accessed 14 April 2016].

[18] Nextwarehouse, "www.nextwarehouse.com," [Online]. Available: http://www.nextwarehouse.com/item/?2237054_g10e. [Accessed 14 April 2016].

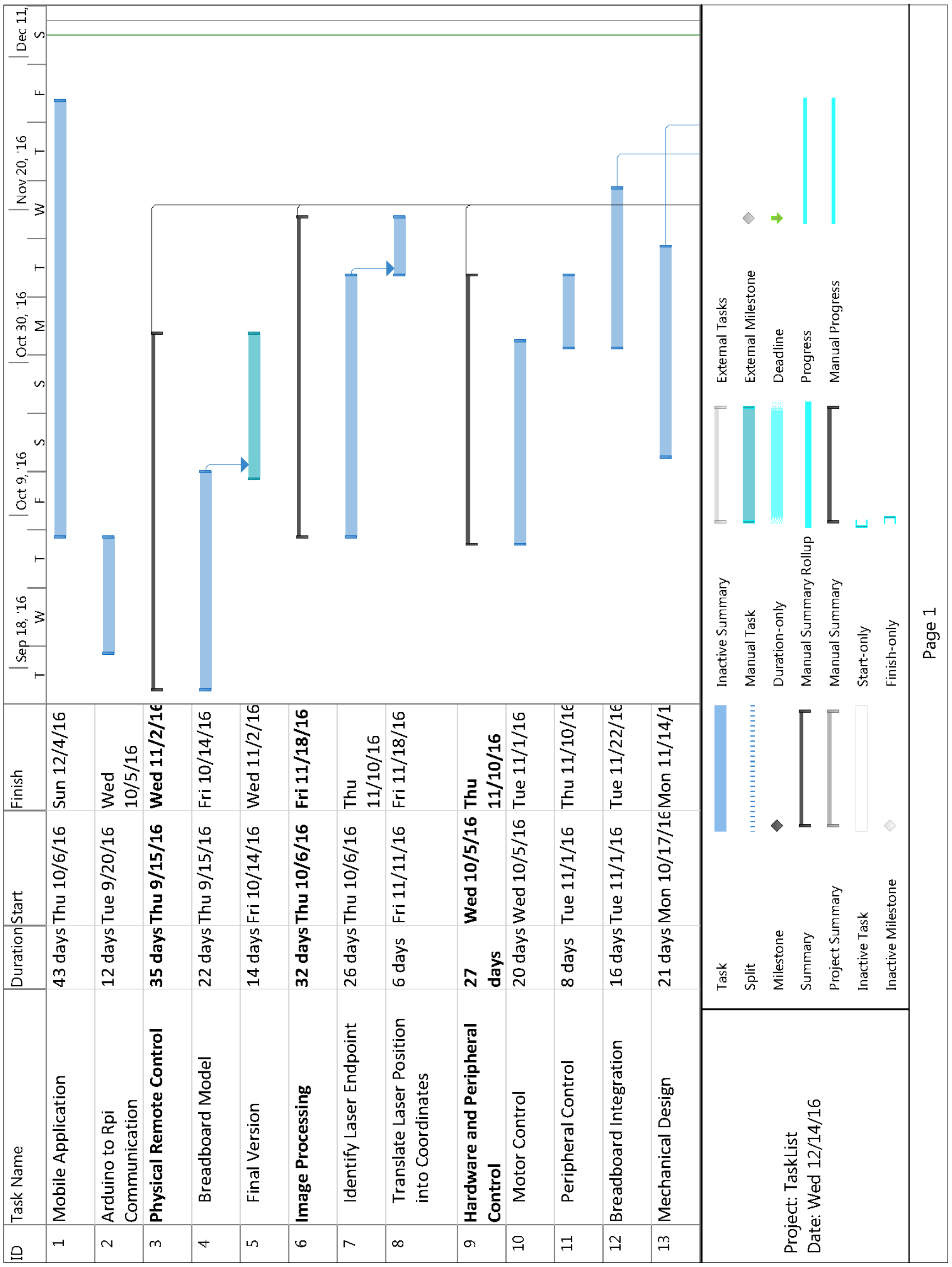
[19] PewPewPew, "REDSS Demonstration Video," Youtube, 11 December 2016. [Online]. Available: https://www.youtube.com/watch?v=Q2EtdLW4Lws. [Accessed 14 December 2016].

Appendix A. Compressed GANTT Chart



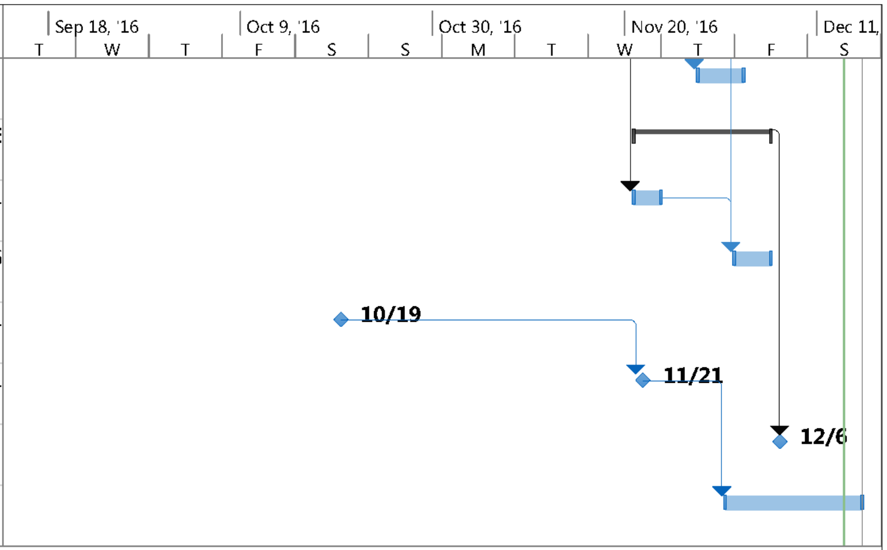
ID	Task Name	Duration	Start	Finish
1	Mobile Application	43 days	Thu 10/6/16	Sun 12/4/16
2	Arduino to Rpi Communication	12 days	Tue 9/20/16	Wed 10/5/16
3	**Physical Remote Control**	**35 days**	**Thu 9/15/16**	**Wed 11/2/1**
6	**Image Processing**	**32 days**	**Thu 10/6/16**	**Fri 11/18/16**
9	**Hardware and Peripheral Control**	**27 days**	**Wed 10/5/16**	**Thu 11/10/16**
12	Breadboard Integration	16 days	Tue 11/1/16	Tue 11/22/1
13	Mechanical Design	21 days	Mon 10/17/1	Mon 11/14/1
14	Physical Integration	5 days	Mon 11/28/1	Fri 12/2/16
15	**Tests**	**12 days**	**Mon 11/21/1**	**Mon 12/5/1**
18	MDR	1 day	Wed 10/19/1	Wed 10/19/1
19	TRR	1 day	Mon 11/21/1	Mon 11/21/1
20	Expo	1 day	Tue 12/6/16	Tue 12/6/16
21	Documentation	12 days	Thu 12/1/16	Thu 12/15/16

Appendix B. Comprehensive GANTT Chart



ID	Task Name	Duration	Start	Finish
1	Mobile Application	43 days	Thu 10/6/16	Sun 12/4/16
2	Arduino to Rpi Communication	12 days	Tue 9/20/16	Wed 10/5/16
3	**Physical Remote Control**	**35 days**	**Thu 9/15/16**	**Wed 11/2/16**
4	Breadboard Model	22 days	Thu 9/15/16	Fri 10/14/16
5	Final Version	14 days	Fri 10/14/16	Wed 11/2/16
6	**Image Processing**	**32 days**	**Thu 10/6/16**	**Fri 11/18/16**
7	Identify Laser Endpoint	26 days	Thu 10/6/16	Thu 11/10/16
8	Translate Laser Position into Coordinates	6 days	Fri 11/11/16	Fri 11/18/16
9	**Hardware and Peripheral Control**	**27 days**	**Wed 10/5/16**	**Thu 11/10/16**
10	Motor Control	20 days	Wed 10/5/16	Tue 11/1/16
11	Peripheral Control	8 days	Tue 11/1/16	Thu 11/10/16
12	Breadboard Integration	16 days	Tue 11/1/16	Tue 11/22/16
13	Mechanical Design	21 days	Mon 10/17/16	Mon 11/14/1

ID	Task Name	Duration	Start	Finish
14	Physical Integration	5 days	Mon 11/28/16	Fri 12/2/16
15	**Tests**	**12 days**	**Mon 11/21/16**	**Mon 12/5/16**
16	Breadboard Tests	3 days	Mon 11/21/16	Wed 11/23/1
17	Final Tests	3 days	Fri 12/2/16	Mon 12/5/16
18	MDR	1 day	Wed 10/19/16	Wed 10/19/1
19	TRR	1 day	Mon 11/21/16	Mon 11/21/1
20	Expo	1 day	Tue 12/6/16	Tue 12/6/16
21	Documentation	12 days	Thu 12/1/16	Thu 12/15/16



Project: TaskList
Date: Wed 12/14/16

Task	▬	Inactive Summary	▭	External Tasks	▬
Split	┈	Manual Task	▬	External Milestone	◆
Milestone	◆	Duration-only	▨	Deadline	⬇
Summary	┏━┓	Manual Summary Rollup	▬	Progress	─
Project Summary	┏━┓	Manual Summary	┏━┓	Manual Progress	─
Inactive Task	▭	Start-only	[		
Inactive Milestone	◇	Finish-only	]		

Page 2

Appendix C. Total Cost Spent in developing REDSS

	Supplier	Part Number	Manufacturer Number	Quant.	Unit Price	Total
1	Amazon.com	B00V7T1YRQ	V1 Pro	1	$13.95	**$13.95**
2	Hobbyking.com	Z50003S-25	Z50003S-25	1	$27.62	**$27.62**
3	Hobbyking.com	9070000044-0	9070000044-0	1	$21.61	**$21.61**
4	Hobbyking.com	9052000023-3	9052000023-3	1	$15.05	**$15.05**
5	Adafruit	3006	3006	1	$5.95	**$5.95**
6	Adafruit	1314	1314	1	$1.95	**$1.95**
7	Sparkfun	BOB-12009	BOB-12009	3	$2.95	**$8.85**
8	Robotshop.com	RB-Hit-53	HS-785HB	1	$44.99	**$44.99**
9	Robotshop.com	RB-Sct-208	SPT-200	1	$45.99	**$45.99**
10	Amazon	MB-MP32DA/AM	MB-MP32DA/AM	1	$9.99	**$9.99**
11	Adafruit	2479	2479	2	$17.50	**$35.00**
12	Adafruit	50	Arduino Uno R3	2	$24.95	**$49.90**
13	Robotshop.com	RB-Lyn-77	RB-Lyn-77	1	$9.95	**$9.95**
14	Pololu	2865	D24V60F5	1	$19.95	**$19.95**
15	Adafruit	(adafruit) 1995	C688-9392	1	$7.95	**$7.95**
16	Adafruit	(Adafruit) 2029	2029	1	$6.95	**$6.95**
17	Adafruit	(Adafruit) 3055	(Adafruit) 3055	2	$39.99	**$79.98**
18	Adafruit	(Adafruit) 2258	(Adafruit) 2258	1	$7.95	**$7.95**
19	Adafruit	(Adafruit) 1086	1086	2	$24.95	**$49.90**
20	Emerging Tech Sales (amazon)	PiNoIR	(ASIN) B00KX3HS4K	1	$23.99	**$23.99**
21	HobbyKing.com	SW5513-6MA	N/A	1	$13.20	**$13.20**
22	Robotshop	RB-Hit-87	HS-485HB	1	$14.99	**$14.99**
23	Digikey (US-Lasers Inc.)	38-1030-ND	D8505I	2	$36.12	**$72.24**
24	Digikey (Adafruit Industries)	1528-1391-ND	1054	2	$5.95	**$11.90**
25	Samsung (amazon)	MB-MP32DA/AM	MB-MP32DA/AM	1	$9.99	**$9.99**
26	Amazon.com	B01E9HM7NC	B01E9HM7NC	1	$6.99	**$6.99**
27	Adafruit	2077	2077	1	$9.95	**$9.95**
28	Adafruit	2310	2310	1	$4.95	**$4.95**
29	Digikey	38-1019-ND	M8505I	2	$62.48	**$124.96**
30	Adafruit	1731	1731	1	$2.95	**$2.95**
				total		**$759.59**

Appendix D. Image Processing Software Code

```
// REDSS Image Processing Code
// Team Pew^3
// 6 Dec 2016
// Version 3.4,
// Mike Bossi, Seong Ho Yeon, Gyu Cheol Lim, Josh Dixon, Satoshi Yuki

#include <ctime>
#include <iostream>
#include <raspicam/raspicam_cv.h>

// Include Rpi-hw headers
#include <rpi-hw.hpp>
#include <rpi-hw/time.hpp>
#include <rpi-hw/gpio.hpp>

#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/core.hpp"
#include "opencv2/features2d.hpp"
#include "opencv2/xfeatures2d.hpp"

#include <pthread.h>
#include <string.h>
#include <vector>

#include <unistd.h>
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <linux/i2c-dev.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <unistd.h>

#include <math.h>
#define  SCALE 0.5

using namespace std;
using namespace cv;
using namespace cv::xfeatures2d;
using namespace rpihw;

enum functions { manual, esc, de_esc, surv, stby };
functions mode = stby;

bool cue_laser = false;

//Laser location
int x = 0;
int y = 0;

//Controlled by user
bool shutdown2 = false;
bool moveFlag = false;
bool motorFlip = false;
raspicam::RaspiCam_Cv cam;
```

```
#define KEYPOINT_THRESHOLD 7
#define AFTER_MOVE_DELAY 50000

// The PiWeather board i2c address
#define I2C_ADDRESS 0x08

// The I2C bus: This is for V2 pi's. For V1 Model B you need i2c-0
static const char *i2cDevName = "/dev/i2c-1";

//I2C Communication protocols
#define I2C_MODE_ZERO 200 // Manual
#define I2C_MODE_ONE  201 // Speaker
#define I2C_MODE_TWO 202 // LaserTracking
#define I2C_MODE_THREE 203 //Surveillance

#define I2C_IR_LASER_ON 210
#define I2C_IR_LASER_OFF 211
#define I2C_RED_LASER_ON 213
#define I2C_RED_LASER_OFF 214
#define I2C_DAZZLER_ON 218
#define I2C_DAZZLER_OFF 219

#define I2C_MOTOR_PAN_POSITIVE_STEP 220
#define I2C_MOTOR_PAN_NEGATIVE_STEP 221
#define I2C_MOTOR_TILT_POSITIVE_STEP 222
#define I2C_MOTOR_TILT_NEGATIVE_STEP 223
#define I2C_MOTOR_PAN_SETPOS 225
#define I2C_MOTOR_TILT_SETPOS 226
#define I2C_MOTOR_FLIP 228

#define GPIO_MODE_PIN1 5
#define GPIO_MODE_PIN2 6
#define GPIO_LASER_PIN 13 // IR_Laser
#define GPIO_MOTOR_PIN 16
#define GPIO_FLIP_PIN 19
#define GPIO_DAZZLER_PIN 26

#define GPIO_ROBOTLED_PIN 17
//io.write(GPIO_LASER_CONTROL_PIN, LOW);

#define GPIO_LASER_CONTROL_PIN 12

#define WIFI_MODE_STBY 11
#define WIFI_MODE_MANUAL

int m1, m2;
#define SOUND_ENG_1 51
#define SOUND_ENG_2 52
#define SOUND_ENG_3 53
#define SOUND_ENG_4 54
#define SOUND_ENG_5 55
#define SOUND_ENG_6 56
#define SOUND_ENG_7 57
#define SOUND_ENG_8 58

#define SOUND_JPN_1 61
#define SOUND_JPN_2 62
#define SOUND_JPN_3 63
#define SOUND_JPN_4 64
```

```
#define SOUND_JPN_5 65
#define SOUND_JPN_6 66
#define SOUND_JPN_7 67
#define SOUND_JPN_8 68

#define SOUND_KOR_1 71
#define SOUND_KOR_2 72
#define SOUND_KOR_3 73
#define SOUND_KOR_4 74
#define SOUND_KOR_5 75
#define SOUND_KOR_6 76
#define SOUND_KOR_7 77
#define SOUND_KOR_8 78

//Networking
#include <netdb.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#define UDP_BUFFER 1

//Ports to open
#define RECEIVE_PORT_MODE 5001
#define RECEIVE_PORT_DIRECTION 5002
#define RECEIVE_PORT_AUDIO 5003

//Buffers for Receiving data from Android App
unsigned char modeBuffer[UDP_BUFFER];

pthread_mutex_t wifi_mutex;
struct sockaddr_in myAddr;
struct sockaddr_in remAddr;
struct sockaddr_in myAddr1;
struct sockaddr_in remAddr1;

void processLaserPos(int x, int y);
void imageStreamingOn(gpio &io);
void checkImageStreaming(gpio &io, functions original_mode);
void* Mode_Manual_Thread(void *v);
void HW_i2c_write(int val);

int HW_GPIO_check_motor(gpio &io);
int HW_GPIO_check_dazzler(gpio &io);
int HW_GPIO_check_flip(gpio &io);
int HW_GPIO_check_laser(gpio &io);
int HW_GPIO_check_mode(gpio &io);

void laserTrackingOp(gpio &io);
void surveillanceOp(gpio &io);
void escallationOp(gpio &io);
void manualOp(gpio &io);
void streamSound(int soundCode);

void* udpReceive(void* v)
{
	int modeSocket = socket(AF_INET, SOCK_DGRAM, 0);
	int receiveLength;
	socklen_t addrlen = sizeof(remAddr1);
	memset((char*)&myAddr1, 0, sizeof(myAddr1));

	myAddr1.sin_family = AF_INET;
```

```
        myAddr1.sin_addr.s_addr = htonl(INADDR_ANY);
        myAddr1.sin_port = htons(RECEIVE_PORT_MODE);

        bind(modeSocket, (struct sockaddr *)&myAddr1, sizeof(myAddr1));

        while (1)
        {
                cout << "In the udpModeReceive" << endl;
                receiveLength = recvfrom(modeSocket, modeBuffer, UDP_BUFFER, 0, (struct 
sockaddr *)&remAddr1, &addrlen);
                if (receiveLength > 0)
                {
                        modeBuffer[receiveLength] = 0;
                        cout << "Receive buffer from WIFI, changing mode" << modeBuffer[0] << 
endl;
                }

                switch (modeBuffer[0])
                {
                case 'A': //stdby
                        pthread_mutex_lock(&wifi_mutex);
                        m1 = 0;
                        m2 = 0;

                        pthread_mutex_unlock(&wifi_mutex);
                        break;
                case 'B': //manual

                        pthread_mutex_lock(&wifi_mutex);
                        m1 = 0;
                        m2 = 1;
                        pthread_mutex_unlock(&wifi_mutex);
                        //HW_i2c_write(I2C_MODE_ONE);
                        break;
                case 'C':
                        pthread_mutex_lock(&wifi_mutex);
                        m1 = 1;
                        m2 = 0;
                        pthread_mutex_unlock(&wifi_mutex);
                        //HW_i2c_write(I2C_MODE_TWO);
                        break;

                case 'D':

                        pthread_mutex_lock(&wifi_mutex);
                        m1 = 1;
                        m2 = 1;
                        pthread_mutex_unlock(&wifi_mutex);
                        //HW_i2c_write(I2C_MODE_THREE);
                        break;
                default:
                        break;
                }

                sleep(1);
        }

        return NULL;
}
```

```
int file; //for I2C
int sound = 0;

void streamSound(int soundCode)
{
	switch (soundCode)
	{
	case 0:
		break;
	case SOUND_ENG_1:
		system("omxplayer -o local ~/Rpi-hw/examples/sound_file/eng1.mp3");
		break;
	case SOUND_ENG_2:
		system("omxplayer -o local ~/Rpi-hw/examples/sound_file/eng2.mp3");
		break;
	case SOUND_ENG_3:
		system("omxplayer -o local ~/Rpi-hw/examples/sound_file/eng3.mp3");

		break;
	case SOUND_ENG_4:
		system("omxplayer -o local ~/Rpi-hw/examples/sound_file/eng4.mp3");
		break;
	case SOUND_ENG_5:
		system("omxplayer -o local ~/Rpi-hw/examples/sound_file/eng5.mp3");
		break;
	case SOUND_ENG_6:
		system("omxplayer -o local ~/Rpi-hw/examples/sound_file/eng6.mp3");
		break;
	case SOUND_ENG_7:
		system("omxplayer -o local ~/Rpi-hw/examples/sound_file/eng7.mp3");
		break;
	case SOUND_ENG_8:
		system("omxplayer -o local ~/Rpi-hw/examples/sound_file/eng8.mp3");
		break;

	case SOUND_JPN_1:
		system("omxplayer -o local ~/Rpi-hw/examples/sound_file/jpn1.mp3");
		break;
	case SOUND_JPN_2:
		system("omxplayer -o local ~/Rpi-hw/examples/sound_file/jpn2.mp3");
		break;
	case SOUND_JPN_3:
		system("omxplayer -o local ~/Rpi-hw/examples/sound_file/jpn3.mp3");
		break;
	case SOUND_JPN_4:
		system("omxplayer -o local ~/Rpi-hw/examples/sound_file/jpn4.mp3");
		break;
	case SOUND_JPN_5:
		system("omxplayer -o local ~/Rpi-hw/examples/sound_file/jpn5.mp3");
		break;
	case SOUND_JPN_6:
		system("omxplayer -o local ~/Rpi-hw/examples/sound_file/jpn6.mp3");
		break;
	case SOUND_JPN_7:
		system("omxplayer -o local ~/Rpi-hw/examples/sound_file/jpn7.mp3");
		break;
	case SOUND_JPN_8:
		system("omxplayer -o local ~/Rpi-hw/examples/sound_file/jpn8.mp3");
		break;
```

```
        case SOUND_KOR_1:
                system("omxplayer -o local ~/Rpi-hw/examples/sound_file/kor1.mp3");
                break;
        case SOUND_KOR_2:
                system("omxplayer -o local ~/Rpi-hw/examples/sound_file/kor2.mp3");
                break;
        case SOUND_KOR_3:
                system("omxplayer -o local ~/Rpi-hw/examples/sound_file/kor3.mp3");
                break;
        case SOUND_KOR_4:
                system("omxplayer -o local ~/Rpi-hw/examples/sound_file/kor4.mp3");
                break;
        case SOUND_KOR_5:
                system("omxplayer -o local ~/Rpi-hw/examples/sound_file/kor5.mp3");
                break;
        case SOUND_KOR_6:
                system("omxplayer -o local ~/Rpi-hw/examples/sound_file/kor6.mp3");
                break;
        case SOUND_KOR_7:
                system("omxplayer -o local ~/Rpi-hw/examples/sound_file/kor7.mp3");
                break;
        case SOUND_KOR_8:
                system("omxplayer -o local ~/Rpi-hw/examples/sound_file/kor8.mp3");
                break;
        }

        sound = 0;
        return;
}

int HW_GPIO_check_mode(gpio &io)
{
        int checkWifi = 1;

        if (m1 == 2 && m2 == 2){
                pthread_mutex_lock(&wifi_mutex);

                m1 = io.read(GPIO_MODE_PIN1);
                m2 = io.read(GPIO_MODE_PIN2);
                checkWifi = 0;
                pthread_mutex_unlock(&wifi_mutex);
        }

        if (m1)
        {
                if (m2)
                {
                        m1 = 2;
                        m2 = 2;
                        if (checkWifi)
                        {
                                HW_i2c_write(I2C_MODE_THREE);

                        }
                        return mode = surv;
                }
                else
                {
                        m1 = 2;
                        m2 = 2;
```

```
                    if (checkWifi)
                    {
                          HW_i2c_write(I2C_MODE_TWO);

                    }
                    return mode = de_esc;
              }
       }
       else
       {
              if (m2)
              {
                    m1 = 2;
                    m2 = 2;
                    if (checkWifi)
                    {
                          HW_i2c_write(I2C_MODE_ONE);
                    }
                    return mode = esc;
              }
              else
              {
                    m1 = 2;
                    m2 = 2;

                    if (checkWifi)
                    {
                          HW_i2c_write(I2C_MODE_ZERO);
                    }
                    return mode = stby;
              }
       }
       return -1;
}
int HW_GPIO_check_laser(gpio &io)
{
       return cue_laser = io.read(GPIO_LASER_PIN);
}

int HW_GPIO_check_flip(gpio &io)
{
       return io.read(GPIO_FLIP_PIN);
}

int HW_GPIO_check_dazzler(gpio &io)
{
       return io.read(GPIO_DAZZLER_PIN);
}

int HW_GPIO_check_motor(gpio &io)
{
       return io.read(GPIO_MOTOR_PIN);
}

void HW_i2c_write(int val)
{
       cout << "Writing value to Arduino " << val << endl;

       unsigned char cmd[1];
```

```
	cmd[0] = val;
	write(file, cmd, 1);
	usleep(1000);

	return;
}


void* Mode_Manual_Thread(void *v)
{
	return NULL;
}

#define STREAMING // activate this when stream on. In development stage, it is commented
void imageStreamingOn(gpio &io)
{
	cout << "void imageStreamingOn(gpio &io)" << endl;

#ifdef STREAMING
	system("uv4l --driver raspicam --auto-video_nr --width 640 --height 480 --encoding
mjpeg --enable-server on \
				--server-option '--port=8090' ");
	cout << "video stream on" << endl;

	usleep(100000);
#endif
	return;
}

void checkImageStreaming(gpio &io, functions original_mode)
{
	cout << "void checkImageStreaming(gpio &io, functions original_mode )" << endl;

#ifdef STREAMING
	HW_GPIO_check_mode(io);
	if (mode != original_mode)
	{
		cout << "killing video stream" << endl;
		system("sudo pkill uv4l");

		usleep(100000);
	}
#endif

	return;
}

void surveillanceOp(gpio &io)
{
	cout << "void surveillanceOp(gpio &io)" << endl;

	imageStreamingOn(io);
	io.write(GPIO_ROBOTLED_PIN, HIGH);

	while (HW_GPIO_check_mode(io) == surv)
	{
		motorFlip = HW_GPIO_check_flip(io);
		if (motorFlip)
		{
			//let Smartphone know that it is flipped.
			}
			sleep(1);

		}

		checkImageStreaming(io, surv); //kill the image streaming
		io.write(GPIO_ROBOTLED_PIN, LOW); //turn off the LED

		return;
}

void escallationOp(gpio &io)
{
		cout << "void escallationOp(gpio &io)" << endl;
		imageStreamingOn(io);

		io.write(GPIO_ROBOTLED_PIN, HIGH);

		while (HW_GPIO_check_mode(io) == esc)
		{
			//sound input should be processed by other thread which communicating UDP.
			streamSound(sound);

			motorFlip = HW_GPIO_check_flip(io);
			if (motorFlip)
			{
				//let Smartphone know that it is flipped.
			}
			sleep(1);

		}

		checkImageStreaming(io, esc); //kill the image streaming
		io.write(GPIO_ROBOTLED_PIN, LOW); //turn off the LED

		return;
}

// Initialize image processing variables
int iLowH = 25;//40; //70; //115
int iHighH = 150; //Works well to remove people

int iLowS = 0;   //127
int iHighS = 239; //22

int iLowV = 15;   //160
int iHighV = 218; //69

int minRadius = 500;
int failed_detect = 0;
int good_detect = 1;
int temp_x = 0;
int temp_y = 0;
int prev_x = 0;
int prev_y = 0;

void laserTrackingOp(gpio &io)
{
		cout << "void laserTrackingOp(gpio &io)" << endl;
```

```
        static std::vector<KeyPoint> keypoints_1;
        static Mat reference;
        static Mat grabbed;
        static Mat diff;
        static Mat imgThresh;
        static Mat blur;
        static Mat img_keypoints_1;
        static Mat temp;
        io.write(GPIO_LASER_CONTROL_PIN, LOW);
        cue_laser = false;

        bool finding_laser = false; //Used for internal control. Keeps from repeatedly trying
to wake IM2

        while (mode == de_esc)
        {
                //Laser is NOT on, grab a frame every 1/2 second and store as the reference
frame until the laser is cued
                HW_GPIO_check_mode(io);
                motorFlip = HW_GPIO_check_flip(io);

                while (!cue_laser && !finding_laser) { //No laser, grab reference frames
                        cam.grab();
                        cam.retrieve(temp);
                        if (motorFlip)
                                flip(temp, temp, 0);

                        resize(temp, reference, Size(), SCALE, SCALE);
                        cout << "MotorFlip " << motorFlip << " and Reference frame grabbed" <<
endl;
                        //imshow("reference", reference);
                        usleep(7000);
                        //cout << "the frame size" << reference.size()<<endl;
                        failed_detect = 0;
                        temp_x = 0;
                        temp_y = 0;
                        good_detect = 1;
                        prev_x = 0;
                        prev_y = 0;

                        HW_GPIO_check_laser(io);
                        HW_GPIO_check_mode(io);

                        if (mode != de_esc) {

                                break; //get Out if mode is changed
                        }//Do nothing, grab another frame at top of loop
                }//end no laser while

                if (mode != de_esc) break; // get Out if mode is changed;

                if (!finding_laser) {
                        finding_laser = true; //Reset to false once laser is centered in screen
                }

                //Image processing------------------------------------------------------------
--------------------------------------------
                //Grab and resize
                cam.grab();
                cam.retrieve(temp);
```

```
		if (motorFlip)
			flip(temp, temp, 0);

		resize(temp, grabbed, Size(), SCALE, SCALE);  //imshow("Grabbed", grabbed );


		//Find differences
		absdiff(reference, grabbed, diff);  //imshow("Diff", diff);

		//Threshold
		inRange(diff, Scalar(iLowH, iLowS, iLowV), Scalar(iHighH, iHighS, iHighV),
imgThresh);

		//Erode and dilate
		erode(imgThresh, imgThresh, getStructuringElement(MORPH_ELLIPSE, Size(3, 3)));
		dilate(imgThresh, imgThresh, getStructuringElement(MORPH_ELLIPSE, Size(3, 3)));

		//Blur
		GaussianBlur(imgThresh, blur, Size(5, 5), 0, 0);
		//imshow("Blur", blur );
		//Find keypoints
		Ptr<SURF> detector = SURF::create(minRadius);
		detector->detect(blur, keypoints_1);

		cout << "Keypoints found: " << keypoints_1.size() << endl;
		//Find laser location-----------------------------------------------------
------------------------------------------
		//If there are only 1 or 2 detected keypoints
		if (keypoints_1.size() > 0 && keypoints_1.size() < KEYPOINT_THRESHOLD) //
ORIGNALLY <3 for longer range. Changed to 5 for short range work
		{
			cout << "X: " << keypoints_1[0].pt.x << "Y: " << keypoints_1[0].pt.y <<
endl; //Print location for debugging
			temp_x += keypoints_1[0].pt.x; //Store in temp variables
			temp_y += keypoints_1[0].pt.y;
			if (prev_x == 0) { //If first find
				prev_x = temp_x; //Last found x
				prev_y = temp_y;
			}//End fresh find if

			if (abs(prev_x - temp_x / good_detect) < 5) { //If the current detect is
near the last detect
				prev_x = temp_x / good_detect; //Set last location to current
average location
				prev_y = temp_y / good_detect;
				good_detect++;				//Increment good detect
				cout << "Good detect" << endl;
				failed_detect--;			//Decrement failed detect
			}//End valid detect

			else {							//If the
keypoints are too far from the last ones, call it a failed detect
				failed_detect++;
				cout << "failed detect123123" << endl;
				temp_x -= keypoints_1[0].pt.x;		//Erase last find
				temp_y -= keypoints_1[0].pt.y;
			}//End failed detect
```

```
                    if (good_detect >= 2) {                                 //If I've had N-1 
good detects, declare laser found and output x,y. 2 is one find, 3 is two finds
                            x = temp_x / (good_detect - 1);
                            y = temp_y / (good_detect - 1);
                            finding_laser = false;                          //No longer 
looking for laser
                            failed_detect = 3;                              //Reset variables
                            //imwrite("whatISaw.png", img_keypoints_1);
                            moveFlag = true;
                            while (HW_GPIO_check_motor(io))
                            {
                                   cout << "Waiting for Motor" << endl;
                                   moveFlag = true;
                                   usleep(200);
                            }

                            processLaserPos(x, y); // send the position to the motor

                    } //End found laser if

                    //failed_detect = 2;

             }
             else if (keypoints_1.size() > 25) {   //15
                    usleep(AFTER_MOVE_DELAY); // I think this delay needed to be optimized

                    //cout << "Vibration. chill." << endl;
                    //imshow("Failed vibration", img_keypoints_1 );
                    //imwrite("Failed Vibration.png", blur);
                    failed_detect = 3;
             } //End major error if

             else {
                    failed_detect += 1;                              //Too many or too few 
keypoints, increment failed detect
                    usleep(5);
             }


             if (failed_detect > 2) {                                //3 or more failures, 
tell remote to turn off laser and find a new reference frame

                    usleep(5);
                    HW_GPIO_check_laser(io);
                    if (!cue_laser) {}

                    else {

                            //HW_i2c_write(I2C_IR_LASER_OFF);
                            HW_GPIO_check_mode(io);
                            if (mode != de_esc) break; // get Out if mode is changed;


                            io.write(GPIO_LASER_CONTROL_PIN, LOW);

                            while (HW_GPIO_check_laser(io)) {

                                   usleep(200); //wait until laser turned off   Reduced in V 
2.3
                            }
                            failed_detect = 0;                              //Reset variables
                            temp_x = 0;  temp_y = 0;
                            good_detect = 1;
                            prev_x = 0;  prev_y = 0;

                            while (HW_GPIO_check_motor(io)) {
                                   cout << "Wait for Motor" << endl;
                                   usleep(100);          //Reduced V 2.3
                                   moveFlag = true;
                                   motorFlip = HW_GPIO_check_flip(io);
                            } //end motor movement while

                            if (moveFlag) {              //Motor was moving, need to let
vibration die down
                                   usleep(350000); //NReduced to .25s from .9s

                                   moveFlag = false;
                            }//End vibration Delay while

                            cam.grab();                                            //Find new
reference frame
                            cam.retrieve(temp);

                            if (motorFlip)
                                   flip(temp, temp, 0);

                            resize(temp, reference, Size(), SCALE, SCALE);
                            cout << "MotorFlip : ," << motorFlip << "New reference" << endl;

                            io.write(GPIO_LASER_CONTROL_PIN, HIGH);
                            HW_GPIO_check_mode(io);
                            if (mode != de_esc) break; // get Out if mode is changed;


                            while (!(HW_GPIO_check_laser(io))) {
                                   usleep(200); //wait until laser turned on
                            }//End laser check while

                     }//End Else
                     //usleep(500);                                         //Wait for
latency
              }
              //back to top, grab another frame and preprocess or stop if laser is no longer
on.
              HW_GPIO_check_mode(io);
       }//end de_esc mode if

}

double X_WIDTH = 640;
double Y_WIDTH = 480;

//needed to be calibrated and changed
double  X_MID_ANGLE = 17; //15
double  X_EDGE_ANGLE = 30; //28
double  Y_MID_ANGLE = 12; // 11
double  Y_EDGE_ANGLE = 23; //2

double IMG_TOLERANCE = 15;
```

```
void standbyOp(gpio &io)
{
	cout << "void sstandbyOp(gpio &io)" << endl;

	imageStreamingOn(io);

	while (HW_GPIO_check_mode(io) == stby)
	{
		sleep(1);
	}
	checkImageStreaming(io, esc);
	return;
}

void processLaserPos(int img_x, int img_y)
{

	double deltaPan, deltaTilt, deltaX, deltaY;

	double temp;
	cout << "Laser position processor" << endl;
	cout << "Laser x : " << img_x << ", Laser y : " << img_y << endl;

	deltaX = -img_x + X_WIDTH / 2;
	deltaY = -img_y + Y_WIDTH / 2;

	if (abs(deltaX) < IMG_TOLERANCE)
	{
		deltaPan = 0;
		HW_i2c_write(I2C_MOTOR_PAN_POSITIVE_STEP);
		usleep(1000);
		HW_i2c_write((int)deltaPan);
		cout << "Write delta Pan to Arduino " << deltaPan << endl;
	}
	else
	{
		if (abs(deltaX)< X_WIDTH / 4)
		{
			deltaPan = deltaX / (X_WIDTH / 4)*X_MID_ANGLE;
		}

		else
		{
			if (deltaX>0)
			{
				temp = X_MID_ANGLE + (abs(deltaX) - X_WIDTH / 4) / (X_WIDTH / 
4)*(X_EDGE_ANGLE - X_MID_ANGLE);
				deltaPan = temp;
			}
			else
			{
				temp = X_MID_ANGLE + (abs(deltaX) - X_WIDTH / 4) / (X_WIDTH / 
4)*(X_EDGE_ANGLE - X_MID_ANGLE);
				deltaPan = -temp;
			}
		}
	}

	if (deltaPan > 0)
	{
		HW_i2c_write(I2C_MOTOR_PAN_POSITIVE_STEP);
		usleep(1000);
		HW_i2c_write((int)deltaPan);
		cout << "Write delta Pan to Arduino " << deltaPan << endl;
	}

	else if (deltaPan < 0)
	{
		HW_i2c_write(I2C_MOTOR_PAN_NEGATIVE_STEP);
		usleep(1000);
		HW_i2c_write((int)(-deltaPan));
		cout << "Write Negative delta Pan to Arduino " << deltaPan << endl;
	}

	if (abs(deltaY) < IMG_TOLERANCE)
		deltaTilt = 0;
	else
		deltaTilt = deltaY * Y_EDGE_ANGLE / (Y_WIDTH / 2);


	if (deltaTilt >= 0)
	{
		HW_i2c_write(I2C_MOTOR_TILT_POSITIVE_STEP);
		usleep(1000);
		HW_i2c_write((int)(deltaTilt));
		cout << "Write delta Tilt  to Arduino " << deltaTilt << endl;
	}

	else if (deltaTilt < 0)
	{
		HW_i2c_write(I2C_MOTOR_TILT_NEGATIVE_STEP);
		usleep(1000);
		HW_i2c_write((int)(-deltaTilt));
		cout << "Write Negative delta Tilt to Arduino " << deltaTilt << endl;
	}

	usleep(10000); //0.2s wait to move Motor.
}

void* escKeyLoop(void* v) {
	while (1) {
		if (waitKey(30) == 27) //wait for 'esc' key press for 30ms. If 'esc' key is
pressed, break loop
		{
			cout << "esc key is pressed by user" << endl;
			break;
		}
		sleep(1);
	}
	return NULL;
}

int main(int argc, char** argv)
{
	/* GPIO Setting up */
	gpio &io = gpio::get();
	io.setup(GPIO_LASER_CONTROL_PIN, OUTPUT);
	io.write(GPIO_LASER_CONTROL_PIN, HIGH);
	io.setup(GPIO_ROBOTLED_PIN, OUTPUT);
	io.write(GPIO_ROBOTLED_PIN, LOW);
	/* Initialize I2C */
	if ((file = open(i2cDevName, O_RDWR)) < 0) {
		fprintf(stderr, "I2C: Failed to access %p\n", i2cDevName);
		exit(1);
	}

	if (ioctl(file, I2C_SLAVE, I2C_ADDRESS) < 0) {
		fprintf(stderr, "I2C: Failed to acquire bus access/talk to slave 0x%x\n",
I2C_ADDRESS);
		exit(1);
	}

	/* Initialize Networking */
	pthread_mutex_init(&wifi_mutex, 0);
	pthread_t modeThread1;
	pthread_create(&modeThread1, NULL, udpReceive, (void *)3);//maybe (void*)t

	mode = manual;
	while (!shutdown2) {
		//cout << "REDSSmain" << endl;

		pthread_t pt_escKey;
		pthread_create(&pt_escKey, 0, escKeyLoop, (void*)2);

		// We only need esckey when we use imshow
		// it is not efficient to operate escKey as a thread
		// Rather it is better to use it right after imshow operation when we do that.

		switch (mode)
		{
		case stby:
			standbyOp(io);
			//usleep(50000); //sleep 0.05s
			break;

		case esc:
			escallationOp(io);
			break;

		case de_esc:

			/* Initialize Camera */
			cam.set(CV_CAP_PROP_FORMAT, CV_8UC3); //set to 8 bit 3 channel
			if (!cam.open())
			{
				cout << "Cannot open the picamera" << endl;
				sleep(1);
			}
			laserTrackingOp(io);


			cam.release();
			usleep(100000);
			break;
		case surv:
			surveillanceOp(io);
			break;
		default:
			break;
```

```
                }
                HW_GPIO_check_mode(io);

                usleep(50000); //wait .05s

        }
        cout << "Exit " << endl;
}
```

Appendix E. Main Robot Hardware Control Code

```
// ECE 4012
// Team Pew^3
// 6 Dec 2016
// Robot V12.0

#include <Servo.h>
#include "RF24.h"
#include <String.h>
#include <Wire.h>

// Variable for RF Comm - 0:Master / 1 : Slave
const bool radioNumber = 1; //Robot is Slave
RF24 radio(9, 10);
byte addresses[][6] = { "1Node", "2Node" };

//RF Communication protocols
#define RF_MODE_ZERO 100 // Manual
#define RF_MODE_ONE  101 // Speaker
#define RF_MODE_TWO 102 // LaserTracking
#define RF_MODE_THREE 103 //Surveillance
#define RF_IR_LASER_ON 200
#define RF_IR_LASER_OFF 201
#define RF_RED_LASER_ON 210
#define RF_RED_LASER_OFF 211
#define RF_DAZZLER_ON 220
#define RF_DAZZLER_OFF 221
#define RF_ROBOT_LASER_ONOFF 226

#define RF_MOTOR_PAN_POSITIVE_SMALL_STEP 230
#define RF_MOTOR_PAN_POSITIVE_LARGE_STEP 231
#define RF_MOTOR_PAN_NEGATIVE_SMALL_STEP 232
#define RF_MOTOR_PAN_NEGATIVE_LARGE_STEP 233

#define RF_MOTOR_TILT_POSITIVE_SMALL_STEP 240
#define RF_MOTOR_TILT_POSITIVE_LARGE_STEP 241
#define RF_MOTOR_TILT_NEGATIVE_SMALL_STEP 242
#define RF_MOTOR_TILT_NEGATIVE_LARGE_STEP 243

#define RF_MOTOR_FLIP 250

#define RF_MOTOR_CENTER 251

//I2C Communication protocols
#define I2C_MODE_ZERO 200 // Manual
#define I2C_MODE_ONE  201 // Speaker
#define I2C_MODE_TWO 202 // LaserTracking
#define I2C_MODE_THREE 203 //Surveillance

#define I2C_IR_LASER_ON 210
#define I2C_IR_LASER_OFF 211
#define I2C_RED_LASER_ON 213
#define I2C_RED_LASER_OFF 214s
#define I2C_DAZZLER_ON 218
#define I2C_DAZZLER_OFF 219

#define I2C_MOTOR_PAN_POSITIVE_STEP 220
```

```
#define I2C_MOTOR_PAN_NEGATIVE_STEP 221
#define I2C_MOTOR_TILT_POSITIVE_STEP 222
#define I2C_MOTOR_TILT_NEGATIVE_STEP 223
#define I2C_MOTOR_PAN_SETPOS 225
#define I2C_MOTOR_TILT_SETPOS 226
#define I2C_MOTOR_FLIP 228

#define RF_CONFIRM_IR_LASER_OFF 252
#define RF_CONFIRM_IR_LASER_ON 253

Servo tiltMotor;  // create servo object to control a servo
Servo panMotor;  // create servo object to control a servo
const int laserInterruptPin = 2;
const int panMotorPin = 3;
const int tiltMotorPin = 5;
const int dazzlerPin = 6;
const int laserPin = 4;
const int RpiGpioComMode1Pin = 7;
const int RpiGpioComMode2Pin = 8;
const int RpiGpioComLaserPin = A0;
const int RpiGpioComMotorPin = A1;
const int RpiGpioComFlipPin = A2;
const int RpiGpioComDazzlerPin = A3;

const int motorMaxAngle = 2310; // 180deg at 2310 us
const int motorMinAngle = 570; // 0 deg at 540 us
int panAngle = 90;
int tiltAngle = 10;
int panBuf = 0;
int tiltBuf = 0;
int mode = 0;
int modeChange = 0;
int robotLaser = 0;
int IRLaser = 0;
int RedLaser = 0;
int Dazzler = 0;
int motorFlip = 0;
int motorStep1 = 3;
int motorStep2 = 6;
int motorStepDelay = 150;

void I2C_init()
{
       Wire.begin(8);                // join i2c bus with address #8
       Wire.onReceive(receiveEvent); // register event
}
void Pin_init()
{
       pinMode(dazzlerPin, OUTPUT);
       pinMode(RpiGpioComMode1Pin, OUTPUT);
       pinMode(RpiGpioComMode2Pin, OUTPUT);
       pinMode(RpiGpioComLaserPin, OUTPUT);
       pinMode(RpiGpioComMotorPin, OUTPUT);
       pinMode(RpiGpioComFlipPin, OUTPUT);
       pinMode(RpiGpioComDazzlerPin, OUTPUT);
       pinMode(laserPin, OUTPUT);

       digitalWrite(dazzlerPin, LOW);
       digitalWrite(RpiGpioComMode1Pin, LOW);
       digitalWrite(RpiGpioComMode2Pin, LOW);
```

```
        digitalWrite(RpiGpioComLaserPin, LOW);
        digitalWrite(RpiGpioComMotorPin, LOW);
        digitalWrite(RpiGpioComFlipPin, LOW);
        digitalWrite(RpiGpioComDazzlerPin, LOW);
        digitalWrite(laserPin, LOW);

        pinMode(laserInterruptPin, INPUT_PULLUP);
        attachInterrupt(digitalPinToInterrupt(laserInterruptPin), laserInterrupt, CHANGE);
}
void Motor_init()
{
        panMotor.attach(panMotorPin, motorMinAngle, motorMaxAngle);
        tiltMotor.attach(tiltMotorPin, motorMinAngle, motorMaxAngle);

        tiltAngle = 10;
        panAngle = 90;

        delay(100);
        panMotor.write(panAngle);

        tiltMotor.write(tiltAngle);
        delay(10);
}

void RF_init()
{
        radio.begin(); // RF communication begin

        // Set the PA Level low to prevent power supply related issues since this is a
        // getting_started sketch, and the likelihood of close proximity of the devices.
RF24_PA_MAX is default.
        radio.setPALevel(RF24_PA_MAX);

        // Open a writing and reading pipe on each radio, with opposite addresses
        if (radioNumber){
               radio.openWritingPipe(addresses[1]);
               radio.openReadingPipe(1, addresses[0]);
        }
        else{
               radio.openWritingPipe(addresses[0]);
               radio.openReadingPipe(1, addresses[1]);
        }

        // Start the radio listening for data
        radio.startListening();
}


int updatePanMotor(int buf = 0)
{
        Serial.print("Update Pan motor with buffer");
        Serial.println(buf);
        //  digitalWrite(RpiGpioComMotorPin , 1);
        panAngle = panAngle + buf;
        if (panAngle >= 180)
               panAngle = 180;

        else if (panAngle < 0) panAngle = 0;

        panMotor.write(panAngle);
```

```
	Serial.print("update panAngle ");
	Serial.println(panAngle);

	delay(motorStepDelay);
	//  digitalWrite(RpiGpioComMotorPin , 0);
}

void updateTiltMotor(int buf = 0)
{
	Serial.print("Update Tilt motor with buffer");
	Serial.println(buf);
	int flipBuf;
	tiltAngle = tiltAngle + buf;



	if (tiltAngle > 180) tiltAngle = 180;
	else if (tiltAngle<0) tiltAngle = 0;
	//digitalWrite(RpiGpioComMotorPin , 1);
	tiltMotor.write(tiltAngle);
	Serial.print("update tiltAngle ");
	Serial.println(tiltAngle);

	delay(motorStepDelay);

	if (tiltAngle>90)
		flipBuf = 1;
	else
		flipBuf = 0;

	if (flipBuf != motorFlip)
	{
		motorFlip = flipBuf;
		digitalWrite(RpiGpioComFlipPin, motorFlip);
	}

	//digitalWrite(RpiGpioComMotorPin , 0);
}

void flipMotor()
{
	/*
	tiltAngle = 180-tiltAngle;
	updateTiltMotor(0);*/
	if (tiltAngle > 90)
	{
		tiltAngle = 16;
		motorFlip = 0;
	}
	else
	{
		motorFlip = 1;
		tiltAngle = 174;
	}
	updateTiltMotor(0);
	digitalWrite(RpiGpioComFlipPin, motorFlip);
	delay(100);
}
```

```
void RpiGpioModeWrite()
{
	/*
	int b1,b2;
	b1 = mode/2;
	b2 = mode%2;
	*/
	digitalWrite(RpiGpioComMode1Pin, mode / 2);
	digitalWrite(RpiGpioComMode2Pin, mode % 2);

	return;
}

int readFromRemote()
{
	unsigned long buf;
	int i = 0;

	if (radio.available())
	{
		radio.read(&buf, sizeof(unsigned long));		// Get the payload

		Serial.print("Receive data from remote : ");
		Serial.println(buf);

		switch (buf)
		{
		case RF_MODE_ZERO://
			mode = 0;
			RpiGpioModeWrite();
			panAngle = 90;
			tiltAngle = 10;
			updatePanMotor(0);
			updateTiltMotor(0);
			break;

		case RF_MODE_ONE: // Speaker
			mode = 1;
			RpiGpioModeWrite();
			break;

		case RF_MODE_TWO: // LaserTracking
			mode = 2;
			RpiGpioModeWrite();
			break;

		case RF_MODE_THREE://Surveillance

			if (mode != 3){

				panAngle = 1;
				updatePanMotor(0);
				delay(200);

				tiltAngle = 16;
				updateTiltMotor(0);
				delay(500);
			}
			mode = 3;
			RpiGpioModeWrite();
```

```
                break;

            case RF_IR_LASER_ON:
                IRLaser = 1;
                digitalWrite(RpiGpioComLaserPin, IRLaser);
                break;

            case RF_IR_LASER_OFF:
                IRLaser = 0;
                digitalWrite(RpiGpioComLaserPin, IRLaser);
                break;

            case RF_RED_LASER_ON:
                RedLaser = 1;
                break;

            case RF_RED_LASER_OFF:
                RedLaser = 0;
                break;

            case RF_CONFIRM_IR_LASER_OFF:
                IRLaser = 0;
                digitalWrite(RpiGpioComLaserPin, IRLaser);
                break;

            case RF_CONFIRM_IR_LASER_ON:
                IRLaser = 1;
                digitalWrite(RpiGpioComLaserPin, IRLaser);
                break;
            case RF_DAZZLER_ON:
                Dazzler = 1;
                digitalWrite(dazzlerPin, Dazzler);
                digitalWrite(RpiGpioComDazzlerPin, Dazzler);
                break;
            case RF_DAZZLER_OFF:
                Dazzler = 0;
                digitalWrite(dazzlerPin, Dazzler);
                digitalWrite(RpiGpioComDazzlerPin, Dazzler);

                break;

            case RF_MOTOR_PAN_POSITIVE_SMALL_STEP:
                updatePanMotor(motorStep1);
                break;
            case RF_MOTOR_PAN_POSITIVE_LARGE_STEP:
                updatePanMotor(motorStep2);
                break;
            case RF_MOTOR_PAN_NEGATIVE_SMALL_STEP:
                updatePanMotor(-motorStep1);
                break;
            case RF_MOTOR_PAN_NEGATIVE_LARGE_STEP:
                updatePanMotor(-motorStep2);
                break;


            case RF_MOTOR_TILT_POSITIVE_SMALL_STEP:
                updateTiltMotor(motorStep1);
                break;
            case RF_MOTOR_TILT_POSITIVE_LARGE_STEP:
                updateTiltMotor(motorStep2);
```

```
                        break;
                case RF_MOTOR_TILT_NEGATIVE_SMALL_STEP:
                        updateTiltMotor(-motorStep1);
                        break;
                case RF_MOTOR_TILT_NEGATIVE_LARGE_STEP:
                        updateTiltMotor(-motorStep2);
                        break;
                case RF_ROBOT_LASER_ONOFF:
                        Serial.println("Laser onoff");
                        if (robotLaser == 1)
                        {
                                robotLaser = 0;
                                Serial.print("Robot Laser ");
                                Serial.println(robotLaser);
                                digitalWrite(laserPin, robotLaser);
                        }
                        else if (robotLaser == 0)
                        {

                                robotLaser = 1;
                                Serial.print("Robot Laser ");
                                Serial.println(robotLaser);
                                digitalWrite(laserPin, robotLaser);
                        }
                        break;
                case RF_MOTOR_CENTER:
                        panAngle = 90;
                        tiltAngle = 10;
                        updatePanMotor(0);
                        updateTiltMotor(0);
                        break;
                case RF_MOTOR_FLIP:
                        flipMotor();
                        break;
                }
                return 0;
        }

        else
                return 0;
}

void sendToRemote(int data)
{
        // delay(100);
        unsigned long buf;
        buf = (unsigned long)data;
        Serial.print("Send Remote to data :");
        Serial.println(buf);

        radio.stopListening();                                          // First, stop listening
so we can talk
        radio.write(&buf, sizeof(unsigned long));               // Send the final one back.
        radio.startListening();                                         // Now, resume listening
so we catch the next packets.
}

int surveillanceDelay = 300;
int flipDelay = 700;
```

```
void surveillanceMovement()
{
       if (panAngle >= 180)
       {
              flipMotor();
              delay(surveillanceDelay);
              panAngle = 0;
              updatePanMotor(0);
              delay(flipDelay);
       }

       else{
              updatePanMotor(motorStep2);
              delay(surveillanceDelay);
       }
}

void changeDazzler()
{
       digitalWrite(dazzlerPin, Dazzler);
}

void setup()
{
       Pin_init();
       delay(10);
       RF_init();
       delay(10);
       I2C_init();
       delay(10);
       Motor_init();
       delay(10);
       Serial.begin(115200);           // start serial for output
}

void loop() {
       readFromRemote();
       switch (mode)
       {
       case 0: //Manual Movement --> Move in the processChange()
              break;
       case 1: // Laser Tracking
              delay(200);
              break;
       case 2: // Speaker - don't know what
              delay(200);
              break;
       case 3:
              surveillanceMovement();
              break;
       default:
              break;
       }

       delay(100);
}

int movBuf = 0;
#define MAXEVENT 20
void receiveEvent(int howMany) {
        int receive_Buf[MAXEVENT];
        int I2C_buf = 0;

        if (howMany > MAXEVENT) howMany = MAXEVENT;

        for (int i = 0; i < howMany; i++)
        {
                receive_Buf[i] = Wire.read();
        }

        for (int i = 0; i < howMany; i++)
        {
                I2C_buf = receive_Buf[i];
                switch (I2C_buf)
                {
                case I2C_MODE_ZERO:
                        mode = 0;
                        RpiGpioModeWrite();
                        panAngle = 90;
                        tiltAngle = 10;
                        updatePanMotor(0);
                        updateTiltMotor(0);
                        sendToRemote(RF_MODE_ZERO);
                        break;
                case I2C_MODE_ONE:
                        mode = 1;
                        RpiGpioModeWrite();
                        sendToRemote(RF_MODE_ONE);
                        break;
                case I2C_MODE_TWO:
                        mode = 2;
                        RpiGpioModeWrite();
                        sendToRemote(RF_MODE_TWO);
                        break;
                case I2C_MODE_THREE:
                        mode = 3;
                        RpiGpioModeWrite();
                        sendToRemote(RF_MODE_THREE);
                        break;
                case I2C_MOTOR_PAN_POSITIVE_STEP:
                        //while( !(I2C_buf = Wire.read()));
                        movBuf = 1; //positive Step
                        digitalWrite(RpiGpioComMotorPin, 1);
                        //updatePanMotor(I2C_buf);
                        break;

                case I2C_MOTOR_PAN_NEGATIVE_STEP:
                        //while( !(I2C_buf = Wire.read()));
                        movBuf = 2;
                        digitalWrite(RpiGpioComMotorPin, 1);
                        //updatePanMotor(-I2C_buf);
                        break;
                case I2C_MOTOR_TILT_POSITIVE_STEP:
                        //while( !(I2C_buf = Wire.read()));
                        digitalWrite(RpiGpioComMotorPin, 1);
                        movBuf = 3;
                        //updateTiltMotor(I2C_buf);
                        break;
                case I2C_MOTOR_TILT_NEGATIVE_STEP:
                        //  while( !(I2C_buf = Wire.read()));
                        movBuf = 4;
                        digitalWrite(RpiGpioComMotorPin, 1);
                        //updateTiltMotor(-I2C_buf);
                        break;
                case I2C_MOTOR_FLIP:
                        break;
                default:
                        switch (movBuf)
                        {
                        case 1:
                                if (I2C_buf != 0)
                                        updatePanMotor(I2C_buf);
                                break;
                        case 2:
                                if (I2C_buf != 0)
                                        updatePanMotor(-I2C_buf);
                                break;
                        case 3:
                                if (I2C_buf != 0)
                                        updateTiltMotor(I2C_buf);
                                digitalWrite(RpiGpioComMotorPin, 0);
                                break;
                        case 4:
                                if (I2C_buf != 0)
                                        updateTiltMotor(-I2C_buf);
                                digitalWrite(RpiGpioComMotorPin, 0);
                                break;
                        default:
                                break;
                        }
                        break;
                }

        }
}

void laserInterrupt()
{
        int input = digitalRead(laserInterruptPin);

        if (mode == 2)
        {
                switch (input)
                {
                case 0:
                        //  IRLaser = 0;
                        // digitalWrite(RpiGpioComLaserPin , IRLaser);
                        sendToRemote(RF_IR_LASER_OFF);
                        break;
                case 1:
                        // IRLaser = 1;
                        // digitalWrite(RpiGpioComLaserPin , IRLaser);
                        sendToRemote(RF_IR_LASER_ON);
                        break;
                }
        }

}
```

Appendix F. Remote Controller Code

```
// ECE 4012
// Team Pew^3
// 6 Dec 2016
// Remote V8.0

#include <SPI.h>
#include "RF24.h"
#include <String.h>

// Variable for RF Comm - 0:Master / 1 : Slave
const bool radioNumber = 0;

// set pin numbers:
const int led1 = 4;     // the number of the pushbutton pin
const int led2 = 6;     // the number of the pushbutton pin
const int led3 = 10;     // the number of the pushbutton pin
const int led4 = 12;     // the number of the pushbutton pin
const int toggle3 = A3;     // the dazzler
const int toggle1 = A4; // IR Button
const int toggle2 = A5; // Red
const int IRLaser = 1;
const int RedLaser = 0;
const int joyButton = 2;

const int joyX = A2;
const int joyY = A1;

const int button1 = 3;
const int button2 = 5;
const int button3 = 7;
const int button4 = 11;

const int button5 = 13; //cetner the Motor
const int button6 = A0; //robot laser

int data_toggle1 = LOW;
int data_toggle2 = LOW;
int data_button1 = LOW;
int data_button2 = LOW;
int data_button3 = LOW;
int data_button4 = LOW;

int opMode = 0; // 0 - Manual, 1 - laserTracking, 2 - Speaker, 3 - Surveillance
int data_IRLaser = 0;
int data_RedLaser = 0;
int data_dazzler1 = 0;
int data_dazzler2 = 0;

int data_motorFlip = 0;
int intDetached = 0;

//RF Communication protocols
#define RF_MODE_ZERO 100 // Manual
#define RF_MODE_ONE  101 // Speaker
#define RF_MODE_TWO 102 // LaserTracking
#define RF_MODE_THREE 103 //Surveillance
```

```
#define RF_IR_LASER_ON 200
#define RF_IR_LASER_OFF 201
#define RF_RED_LASER_ON 210
#define RF_RED_LASER_OFF 211
#define RF_DAZZLER_ON 220
#define RF_DAZZLER_OFF 221
#define RF_CONFIRM_IR_LASER_OFF 252
#define RF_CONFIRM_IR_LASER_ON 253
#define RF_ROBOT_LASER_ONOFF 226
#define RF_MOTOR_PAN_POSITIVE_SMALL_STEP 230
#define RF_MOTOR_PAN_POSITIVE_LARGE_STEP 231
#define RF_MOTOR_PAN_NEGATIVE_SMALL_STEP 232
#define RF_MOTOR_PAN_NEGATIVE_LARGE_STEP 233
#define RF_MOTOR_TILT_POSITIVE_SMALL_STEP 240
#define RF_MOTOR_TILT_POSITIVE_LARGE_STEP 241
#define RF_MOTOR_TILT_NEGATIVE_SMALL_STEP 242
#define RF_MOTOR_TILT_NEGATIVE_LARGE_STEP 243
#define RF_MOTOR_FLIP 250
#define RF_MOTOR_CENTER 251

RF24 radio(8, 9);
/***********************************************************/

byte addresses[][6] = { "1Node", "2Node" };

// Used to control whether this node is sending or receiving
void Pin_init()
{
       pinMode(led1, OUTPUT);
       pinMode(led2, OUTPUT);
       pinMode(led3, OUTPUT);
       pinMode(led4, OUTPUT);


       digitalWrite(led1, HIGH);
       opMode = 0;

       digitalWrite(led2, LOW);
       digitalWrite(led3, LOW);
       digitalWrite(led4, LOW);
       //  digitalWrite(led5, LOW);

       pinMode(IRLaser, OUTPUT);
       pinMode(RedLaser, OUTPUT);
       digitalWrite(IRLaser, LOW);
       digitalWrite(RedLaser, LOW);

       pinMode(toggle1, INPUT_PULLUP); // toggle1 - IR switch
       pinMode(toggle2, INPUT_PULLUP); // toggle2 - Red switch
       pinMode(toggle3, INPUT_PULLUP); // Dazzler Switch

       pinMode(joyButton, INPUT_PULLUP); // joyButton
       attachInterrupt(digitalPinToInterrupt(joyButton), flipMotor, LOW);

       pinMode(button1, INPUT_PULLUP); // mode1 : Manual
       pinMode(button2, INPUT_PULLUP); // mode2 : laser
       pinMode(button3, INPUT_PULLUP); // mode3 : speaker
       pinMode(button4, INPUT_PULLUP); // mode4 : surveillance

       pinMode(button5, INPUT_PULLUP); // center the motor
```

```
        pinMode(button6, INPUT_PULLUP); // turn on/off robot laser
}

void RF_init()
{
        radio.begin(); // RF communication begin
        // Set the PA Level low to prevent power supply related issues since this is a
        // getting_started sketch, and the likelihood of close proximity of the devices.
RF24_PA_MAX is default.
        radio.setPALevel(RF24_PA_MAX);

        // Open a writing and reading pipe on each radio, with opposite addresses
        if (radioNumber){
                radio.openWritingPipe(addresses[1]);
                radio.openReadingPipe(1, addresses[0]);
        }
        else{
                radio.openWritingPipe(addresses[0]);
                radio.openReadingPipe(1, addresses[1]);
        }

        // Start the radio listening for data
        radio.startListening();
}

int readFromRobot()
{
        unsigned long buf;
        if (radio.available())
        {
                radio.read(&buf, sizeof(unsigned long));              // Get the payload
                Serial.print("Read from Robot with buffer : ");
                Serial.println(buf);
                switch (buf)
                {
                case RF_IR_LASER_ON:
                        data_IRLaser = 1;
                        digitalWrite(IRLaser, data_IRLaser);
                        sendToRobot(RF_CONFIRM_IR_LASER_ON);
                        break;
                case RF_IR_LASER_OFF:
                        data_IRLaser = 0;
                        digitalWrite(IRLaser, data_IRLaser);
                        sendToRobot(RF_CONFIRM_IR_LASER_OFF);
                        break;
                default:
                        break;

                case RF_MODE_ZERO:
                        opMode = 0;
                        digitalWrite(led1, HIGH);
                        digitalWrite(led2, LOW);
                        digitalWrite(led3, LOW);
                        digitalWrite(led4, LOW);
                        Serial.println("Mode 0");
                        break;
                case RF_MODE_ONE:
                        opMode = 1;
                        digitalWrite(led1, LOW);
                        digitalWrite(led2, HIGH);
                        digitalWrite(led3, LOW);
                        digitalWrite(led4, LOW);
                        Serial.println("Mode 1");
                        break;

                case RF_MODE_TWO:
                        opMode = 2;
                        digitalWrite(led1, LOW);
                        digitalWrite(led2, LOW);
                        digitalWrite(led3, HIGH);
                        digitalWrite(led4, LOW);
                        Serial.println("Mode 2");
                        break;


                case RF_MODE_THREE:
                        opMode = 3;
                        digitalWrite(led1, LOW);
                        digitalWrite(led2, LOW);
                        digitalWrite(led3, LOW);
                        digitalWrite(led4, HIGH);
                        Serial.println("Mode 3");
                        break;

                }
        }

        return (int)buf;
}

void sendToRobot(int data)
{
        unsigned long buf;
        buf = (unsigned long)data;

        radio.stopListening();                                  // First, stop listening
so we can talk
        radio.write(&buf, sizeof(unsigned long));             // Send the final one back.
        radio.startListening();                                 // Now, resume listening
so we catch the next packets.

}

void checkSwitch()
{
        int newRed, newIR, newDazzler1, newDazzler2;

        newRed = digitalRead(toggle1);

        newDazzler1 = digitalRead(toggle3);

        if (newRed != data_RedLaser)
        {
                data_RedLaser = newRed;
                digitalWrite(RedLaser, !data_RedLaser);
                Serial.print("Red Laser output");
                Serial.println(!(data_RedLaser));

                if (data_RedLaser)
                {
                        sendToRobot(RF_RED_LASER_OFF);
                }
                else
                {
                        sendToRobot(RF_RED_LASER_ON);
                }
        }

        if (newDazzler1 != data_dazzler1)
        {
                data_dazzler1 = newDazzler1;

                Serial.print("Dazzler output1");
                Serial.println(!(data_dazzler1));

                if (data_dazzler1)
                {
                        sendToRobot(RF_DAZZLER_OFF);
                }
                else
                {
                        sendToRobot(RF_DAZZLER_ON);
                }
        }
}

void checkButton()
{

        if (!(digitalRead(toggle2)))
        {
                data_IRLaser = !data_IRLaser;
                digitalWrite(IRLaser, data_IRLaser);
                Serial.print("IR Laser output");
                Serial.println((data_IRLaser));

                if (!data_IRLaser)
                {
                        sendToRobot(RF_IR_LASER_OFF);
                }
                else
                {
                        sendToRobot(RF_IR_LASER_ON);
                }
        }


        if (!digitalRead(button1))
        {
                opMode = 0;
                digitalWrite(led1, HIGH);
                digitalWrite(led2, LOW);
                digitalWrite(led3, LOW);
                digitalWrite(led4, LOW);
                Serial.println("Mode 0");

                sendToRobot(RF_MODE_ZERO);
        }

        else if (!digitalRead(button2))
        {
                opMode = 1;
                digitalWrite(led1, LOW);
                digitalWrite(led2, HIGH);
                digitalWrite(led3, LOW);
                digitalWrite(led4, LOW);
                Serial.println("Mode 1");

                sendToRobot(RF_MODE_ONE);
        }

        else if (!digitalRead(button3))
        {
                opMode = 2;
                digitalWrite(led1, LOW);
                digitalWrite(led2, LOW);
                digitalWrite(led3, HIGH);
                digitalWrite(led4, LOW);
                Serial.println("Mode 2");

                sendToRobot(RF_MODE_TWO);
        }

        else if (!digitalRead(button4))
        {
                opMode = 3;
                digitalWrite(led1, LOW);
                digitalWrite(led2, LOW);
                digitalWrite(led3, LOW);
                digitalWrite(led4, HIGH);
                Serial.println("Mode 3");

                sendToRobot(RF_MODE_THREE);
        }

        if (!digitalRead(button5))
        {
                Serial.println("Center the Motor");

                sendToRobot(RF_MOTOR_CENTER);
        }

        if (!digitalRead(button6))
        {
                Serial.println("Robot laser on or off");

                sendToRobot(RF_ROBOT_LASER_ONOFF);
        }
}


void flipMotor()
{
        if (opMode == 1)
        {
                detachInterrupt(digitalPinToInterrupt(joyButton));
                data_motorFlip = !data_motorFlip;

                Serial.print("Motor Flip to :");
                Serial.println(data_motorFlip);
```

```
		sendToRobot(RF_MOTOR_FLIP);
		intDetached = 1;
	}
}

int X1 = 25;
int X2 = 300;
int X4 = 700;
int X5 = 1000;

int Y1 = 25;
int Y2 = 300;
int Y4 = 700;
int Y5 = 990;

void manualMovement()
{
	int panMove = 0;
	int tiltMove = 0;

	int rawX, rawY;

	if (intDetached){
		delay(200);
		intDetached = 0;
		attachInterrupt(digitalPinToInterrupt(joyButton), flipMotor, LOW);
	}

	rawX = analogRead(joyX); // disregard if 0~1023 , disregard 400-600
	if (rawX < X1)
		sendToRobot(RF_MOTOR_PAN_NEGATIVE_LARGE_STEP);
	else if (rawX <X2)
		sendToRobot(RF_MOTOR_PAN_NEGATIVE_SMALL_STEP);
	else if (rawX >X4 && rawX<X5)
		sendToRobot(RF_MOTOR_PAN_POSITIVE_SMALL_STEP);
	else if (rawX >X5)
		sendToRobot(RF_MOTOR_PAN_POSITIVE_LARGE_STEP);

	Serial.print("Pan Move");
	delay(100);

	rawY = analogRead(joyY); // disregard if 0~1023 , disregard 400-600
	if (rawY < Y1)
		sendToRobot(RF_MOTOR_TILT_NEGATIVE_LARGE_STEP);
	else if (rawY <Y2)
		sendToRobot(RF_MOTOR_TILT_NEGATIVE_SMALL_STEP);
	else if (rawY >Y4 && rawY <Y5)
		sendToRobot(RF_MOTOR_TILT_POSITIVE_SMALL_STEP);
	else  if (rawY >Y5)
		sendToRobot(RF_MOTOR_TILT_POSITIVE_LARGE_STEP);

	Serial.print("Tilt Move");
	delay(100);

}

void setup() {
	delay(500);
	Serial.begin(115200); // Serial Initialize
	delay(500);
	RF_init(); // RF initialize
	Pin_init();
	Serial.println("Initialize End \n\n");
}

void loop() {
	checkSwitch();
	checkButton();
	readFromRobot();
	delay(100);

	if (opMode == 1)
		manualMovement();
} // Loop
```