**rro**

**EE175ABC Final Report Template**

**Autonomous Ground Vehicle**

**EE 175AB Final Report**

**Department of Electrical Engineering, UC Riverside**

|  |  |
| --- | --- |
| Project Team Member(s) | Spencer Lee slee163@ucr.edu,  Billy Xiao bxiao001@ucr.edu,  Ryan Sabik rsabi001@ucr.edu |
| Date Submitted | June 5, 2015 |
| Section  Professor | Gang Chen |
| Revision | 1.0 |
| URL of Project Wiki/Webpage | https://drive.google.com/open?id=1hyOt0s-E-9kEQT1jALnqrEfsevVe7e4liLViJ\_D8MLU&authuser=0 |

Summary

This report presents the design and engineering involved in the creation of the Unmanned Ground Vehicle. These considerations include low level design, testing, problems encountered and others. In addition there is a brief section on teamwork, and future applications of the project.

Revisions

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version | Description of Version | Author(s) | Date Completed | Approval |
| Version Number | Information about the revision. This table does not need to be filled in whenever a document is touched, only when the version is being upgraded. | Full Name | 00/00/00 |  |
| 0.1 | First draft | Spencer Lee, Billy Xiao,  Ryan Sabik | 04/20/2015 |  |
| 0.2 | Second Draft | Spencer Lee,  Billy Xiao,  Ryan Sabik | 5/22/2015 |  |
| 1.0 | Final Draft | Spencer Lee,  Billy Xiao  Ryan Sabik | 5/5/2015 |  |

Table of Contents

Revisions. 3

Table of Contents. 4

1 \* Executive Summary. 7

2 \* Introduction. 8

2.1 \* Design Objectives and System Overview.. 8

2.2 \* Backgrounds and Prior Art. 8

2.3 \* Development Environment and Tools. 9

2.4 \* Related Documents and Supporting Materials. 9

2.5 \* Definitions and Acronyms. 9

3 \* Design Considerations. 10

3.1 \* Assumptions. 10

3.2 \* Realistic Constraints. 10

3.3 System Environment and External Interfaces. 10

3.4 \* Industry Standards. 11

3.5 \* Knowledge and Skills. 11

3.6 \* Budget and Cost Analysis. 11

3.7 \* Safety. 11

3.8 Performance, Security, Quality, Reliability, Aesthetics etc. 12

3.9 \* Documentation. 12

3.10 Design Methodology. 12

3.11 Risks and Volatile Areas. 12

3.12 \* Understanding of Professional and Ethical Responsibility. 12

3.13 \* Global, Economic, Environmental and Societal Impact. 13

3.14 \* Contemporary Engineering Issues. 13

3.15 \* Recognition of the need for and an ability to engage in lifelong learning 13

3.16 \* Importance of Team Work. 13

4 Experiment Design and Feasibility Study. 15

4.1 Experiment Design. 15

4.2 Experiment Results and Feasibility. 15

5 Architecture. 16

5.1 System Architecture. 16

5.2 Rationale and Alternatives. 16

6 \* High Level Design. 17

6.1 Conceptual View.. 17

6.2 Hardware. 17

6.3 Software. 17

7 Data Structures. 18

7.1 Internal software data structure. 18

7.2 Global data structure. 18

7.3 Temporary data structure. 18

7.4 Database descriptions. 18

8 \* Low Level Design. 19

8.1 The X Module (replace x with the name of the module and repeat the section for all modules, e.g., The Wireless Module, The Sensor Module, The Control Module, etc.) 19

8.1.1 Processing narrative for the X module. 19

8.1.2 X Module interface description. 19

8.1.3 X Module processing details 19

9 User Interface Design. 20

9.1 Application Control. 20

9.2 Screen 1..m.. 20

9.3 Development system and components available. 20

10 \* Test Plan. 21

10.1 \* Design of Tests. 21

10.2 \* Bug Tracking. 21

10.3 \* Quality Control. 21

10.4 Performance bounds. 21

10.5 \* Identification of critical components. 21

10.6 \* Items Not Tested. 21

11 \* Test Report. 22

11.1 \* Test Iteration 1. 22

11.1.1 Test 1. 22

11.1.2 Test i (repeat for i = 2, …., n) 22

11.2 \* Test Iteration j (repeat for j=2,…m) 22

11.2.1 Test 1. 22

11.2.2 Test i (repeat for i = 2, …., n) 22

12 Administrative and Other Design Issues. 24

12.1 \* Project Management. 24

12.2 Requirements traceability matrix. 24

12.3 Packaging and installation issues. 24

12.4 Design metrics to be used. 24

12.5 Restrictions, limitations, and constraints. 24

13 \* Conclusion and Future Work. 25

13.1 \* Conclusion. 25

13.2 Future Work. 25

13.3 \* Acknowledgement. 25

14 \* References. 26

15 Appendices. 27

# **1 \* Executive Summary**

For our project, we designed an Unmanned Ground Vehicle with primary considerations being placed on meeting the requirements listed in the Proposed Projects document. The Unmanned Ground Vehicle had 4 required goals, must be remotely controllable, and must send status details, such as speed, back to a remote location.

In order to remotely drive the car, we decided the Unmanned Ground Vehicle must send a video feed and accept controls from a remote through a wireless connection. To that end, multiple server and client connections must be made using a combination of TCP and UDP. The video stream is sent to a laptop which acts as our central remote location and that’s also where we control our Unmanned Ground Vehicle with a Bluetooth-PlayStation Remote Controller.

To have the car accurately maneuver around without human control, we chose the method to use ultrasonic sensors. The Unmanned Ground Vehicle uses the sensors to detect if there is any obstruction to close to the vehicle and avoids traveling in that direction. As for remote control, we used a webcam that streams through TCP and UDP protocols to give the user a live video feed so we can maneuver the Unmanned Ground Vehicle without actually seeing the vehicle itself. With the webcam, we can just rely on the live feed and maneuver with the visual feedback from the laptop.

This is all being controlled by our Raspberry Pi, which acts as our main central processing unit. We chose this because of its high processing capabilities as well as its given peripherals for us to use. We also use an Arduino Uno to control our drive train. The Arduino Uno is small enough, but yet strong enough to relay commands to our driver motors as well as relay speed and other information as well.

As we were building our project, here are some of the important achievements we obtained:

* Fully configuring the Raspberry Pi
* Understand the controls of our hardware, i.e. understand how the Ultrasonic Sensors work and its capabilities as well as it limitations.
* Remote control drivability
* Wi-Fi control drivability through TCP/UDP
* Basic autonomous drivability with the aid of Ultrasonic Sensors
* More advanced autonomous drivability
* Integrated “switch-over” drivability, i.e. Autonomous Mode – Remote Control Mode
* Basic video feedback
* “Live-Video” feedback enhancement

# **2 \* Introduction**

## 2.1 \* Design Objectives and System Overview

The system’s core is two external processing units, the Arduino Uno, and the Raspberry Pi. The Arduino is used to power and read from the sensor, as well as power the 4 DC motors that turn the wheels, the servomotor that turns the camera, and it sends information about the speed of the car to the Raspberry Pi. The Raspberry Pi controls the camera, that means the video feed from the camera, as well as receiving controls from the controller, and sends the video to the computer.

We designed an Unmanned Ground Vehicle. It has the capability to self-drive, maneuver around objects, as well as send information about the vehicle back to a computer through a wireless connection. The wireless connection also allows us to control the vehicle remotely, as well as receive a video feed from the camera mounted on the vehicle.

This project has far reaching implications on the future of our society. Major technological companies, such as Google, have already developed automated driving vehicles, which while using different technology to accomplish this task, have the same basic idea, that automated vehicles are the future of driving. Our vehicle, being unmanned, is closer to the military application of drones that allow for manual override as well as patrolling and monitoring situations, while avoiding collision with any object near it.

This project consisted of a couple of technical design objectives: Raspberry Pi as our main processing unit, Ultrasonic Sensor to drive the autonomous drivability, network protocol and video streaming.

Our vehicle is powered by a 1000maH battery, which can drive the vehicle for several hours. The Raspberry Pi has four USB ports, which we use for a Bluetooth Adapter and a webcam for live video stream. The Raspberry Pi has numerous ports available for us to use, such as USART, to communicate with our Arduino Uno.

The Arduino Uno controls the Ultrasonic Sensors, and with the information provided by the Ultrasonic Sensors, it can calculate how to drive by commanding the driver motors, which control the four individual wheels. All these information are also sent to the Raspberry Pi via USART to control any necessary actions. The Ultrasonic Sensors have precision up to 13 feet, an effectual angle of less than 15**°**, measuring angle of 30**°**, and a resolution of 0.3 cm. For the Ultrasonic sensors to work effectively, it needs a power source of 5V DC, a quiescent current of no more than 2ma, and the trigger input pulse must be a width of at least 10uS.

|  |  |
| --- | --- |
| **Responsibilities:** | **Team Members Involved:** |
| Raspberry Pi | Spencer Lee |
| Arduino Uno | Billy Xiao |
| Sensors | Ryan Sabik and Billy Xiao |
| Networking | Spencer Lee and Ryan Sabik |
| Hardware | Billy Xiao |
| Video Streaming | Ryan Sabik |

## 2.2 \* Backgrounds and Prior Art

Majority of our findings were from Google searching and reading up specification sheets that were given with our hardware or searching on the web. The web was a great source to our advantage, but it was also very misleading on generic parts that we used.

Our main advantage was finding sources from the Raspberry Pi community, such as the creators of PyGame. This greatly helped us in achieving numerous functionalities such as remote control drivers and drivers that aided with our video streaming. The web helped us find resources that guided us to effectively use our hardware. There were numerous spec sheets and tutorials, and with that, we were able to integrate it with our project.

There were also some drawbacks as well. Some of our hardware did not come with any spec sheets, thus, making it harder for us to understand since there was only generic information about some of these hardware. For example, our Ultrasonic sensor was pretty generic and we found numerous sources that claim a “correct” functionality, but was sometimes misleading.

## 2.3 \* Development Environment and Tools

The design environment of our project consisted of three different environments, the Pygame Library, Pyserial, both for the Raspberry Pi, and the Arduino coding environment included when uploading code to the Arduino. We coded this project mainly on two different computers, a laptop and a desktop. We also used IDEs such as Eclipse to organize, simulate our code, and run network commands as well as other commands to directly control our vehicle.

## 2.4 \* Related Documents and Supporting Materials

TCP/UDP protocols. Bluetooth protocols.

## 2.5 \* Definitions and Acronyms

UGV – Unmanned Ground Vehicle

TCP – Transmission Control Protocol

UDP – User Datagram Protocol

UART – Universal Asynchronous Receiver/Transmitter

USART – Universal Synchronous Receiver/Transmitter

# **3 \* Design Considerations**

Some issues that we addressed prior to completing and while completing the design includes how we should utilize our Raspberry Pi as our main processing unit, how to overcome the issues with timer interrupt when using the Ultrasonic Sensors, which protocols to use for our networking, and how to make our video stream “live” with acceptable delays and quality of the video feed.

We addressed the Raspberry Pi first because we needed to utilize its capabilities as well as efficiently use its peripherals and functionality without any overhead and over complications.

Originally, we were going to use Atmega microcontrollers to use our Ultrasonic Sensors, but the complications were to great with manually configuring timer interrupts. So we opted for Arduino Uno. The Arduino Uno helped us utilize the Ultrasonic Sensors almost flawlessly. Timer interrupt is used so we can count how long it takes for an echo to be triggered after the trigger input.

TCP and UDP have both pros and cons to which protocol to use. We ended up using both to take advantage of each protocol for specific tasks. TCP requires full packets to be sent, but we do not need to manage data integrity. For UDP, we can send packets as freely as we want, but we have to manually manage data integrity.

With the help of using both network protocols, we were able to enhance video streaming to a “live” video stream with acceptable delays. We had to decrease the frames per second enough to maintain quality, but our concern was the delay. If we had any delays, we would not be able to control our UGV “live.”

## 3.1 \* Assumptions

The Unmanned Ground Vehicle requires non-sound absorbing surfaces for the ultrasonic sensors to read distance properly. The Unmanned Ground Vehicle also requires the fact that 802.11 wireless signals operate in the environment as the wireless feed uses Wi-Fi to send and receive from a computer.

## 3.2 \* Realistic Constraints

The maximum power draw of the vehicle must be under 10.2 Watts, as the battery supplies a maximum of 5V and 2.1A. The processing of the sensors, determination of the correct movement and control of the motors must be under the maximum processing speed of 16MHz provided by the Arduino Uno. The video reading from the camera and wireless connection must be under the processing speed of 700MHz provided by the Raspberry Pi. The wireless connection must be fast enough to send/receive a live video feed at a resolution of 360p at 10fps, with an operating distance of up to at least 30 feet away from the computer. The weight of the car must be ~~~~~~. The maximum budget we allocated to the project was $300.

## 3.3 \* System Environment and External Interfaces

Our project utilizes both TCP and UDP protocols. We use both these protocols to communicate our UGV with our laptop/desktop as well as sending “live video” feedback from the camera mounted on our UGV to our computer.

## 3.4 \* Industry Standards

Industry standards that we were involved with in order to complete our projects were: USB, TCP/UDP, and Bluetooth.

## 3.5 \* Knowledge and Skills

Describe the knowledge and skills required by this project. Complete the following for each team member:

List all the Electrical Engineering and other technical courses (e.g., Computer science or upper division physics) you took or are taking that are related to your project.

List any new knowledge and skills that you had to learn while working on the project in order to complete it.

Electrical Engineering and other Technical Courses taken by team members

|  |  |
| --- | --- |
| Course Number | Course Title |
| CS 100 | Software Construction |
| CS 120B | Introduction to Embedded Systems |
| CS 122A | Intermediate Embedded Systems |
| CS 161 | Computer Architecture |
| EE 1AB | Introduction to Circuit Analysis |
| EE 100A | Electronic Circuits |

New knowledge and skills obtained while working on project

|  |
| --- |
| **Knowledge/Skill** |
| Network protocols |
| Video Enhancement |
| Data Management |

## 3.6 \* Budget and Cost Analysis

Initial Budget: $300

Actual Used: ~$250

Break Down:

|  |  |
| --- | --- |
| **Parts** | **Price** |
| Raspberry Pi | $44.00 |
| Arduino Uno | $25.00 |
| Vehicle with chassis, motor, wheel | $25.00 |
| Motor Drivers | $14.00 x 4 = $56.00 |
| Webcam | Free, spare part available |
| Ultrasonic Sensor | $5.00 |
| Others - Need to verify other parts | $100 |

## 3.7 \* Safety

Majority of our project does not propose any serious safety threat, but one to consider is if this vehicle was to be used frequently, beware of touching motor drivers. The drivers will be very hot. The motor driver requirements for power are low enough that very cheap motor drivers are used to drive the wheels, but these have no heat sink, and cause the motor drivers to overheat frequently, when overheated the motor drivers are hot enough to sting on touch, but will not cause severe or permanent damage.

The vehicle, while in automatic mode, will stop or turn before hitting anything. Unless the car is in manual drive, the chance of the vehicle hitting something is minimal. The vehicle does not weigh enough, or move fast enough, to produce any injury causing impact, should it hit anything.

## 3.8 \* Documentation

As we were building our project, we kept a notebook of all the major build notes as well as any notes on improvement to the design. Our side notes helped us keep track of what was done and what problems we ran into in order to finish our report.

## 3.9 \* Understanding of Professional and Ethical Responsibility

The automation of vehicles has such obvious benefits to the public and private sectors, as well as to the general public, that Google had publicly announced in 2010 its intention to have automated cars on public roads. Along with the benefits of a self-driving vehicle, are the obvious danger risks to other drivers should the vehicle make a wrong turn or acceleration. Our autonomous ground vehicle is not made at the size and does not have the speed of a human carrying vehicle. It will not be able to generate enough force to cause bodily injury, and we have no fear of the repercussions of it impacting a human. The physical health risks are minor should our project become a commercial product at its current weight and size.

With the autonomous ground vehicle’s small size comes the concern for privacy invasion. Today’s society has large issues with privacy invasion from companies such as Facebook, and government agencies, such as the NSA. Our vehicle can monitor areas, and send a video feed from the camera mounted on the vehicle to a remote computer. A commercial version of this product would be used as a patrolling security drone. The vehicle’s ability to spy on unsuspecting people is something that could cause an obvious concern, however it was not something we placed a huge emphasis on trying to solve, or much thought at all.

An unmanned patrolling drone, used for security purposes, would not want to be seen or heard easily. If it was easily spotted and avoided, then it would not be a very good patrolling drone. The fact that this autonomous ground vehicle could be used as a device to spy on others is simply something that must be part of the product, making it unable to discretely monitor an area eliminates the drone’s primary function.

In the $250 budget, we decided to split the project’s costs among the three of us as equally as we could. Along with splitting the workload into what we believed were equal partitions, we also tried to dedicate a fairly similar amount of time on the project per person. Of course, not everything goes according to plan, and some spent more time than others on the project. Trying to maintain a proper amount of time per person, along with some things plainly being done faster by one person, or people having more time to work on aspect more than others forced us to acknowledge that a perfectly symmetrical amount of time and work on every single aspect of the project while perhaps the most ethically responsible in terms of workload, is simply not the best overall plan to efficiently work on the project. Rather, we saw that being fluid and accepting of changes or shifts in the project work was the correct way to assign responsibilities in a project with all members having a changing schedule and different specialities.

## 3.10 \* Global, Economic, Environmental and Societal Impact

An autonomous ground vehicle at the price point of our $250 design is not in the price range that would be viable for mass public use. The vehicle will likely be used by companies looking to cut costs in surveillance. Rather than having people patrol the area, they would be able to purchase a bulk of these vehicles that would replace the recurring salary costs of hired men and women for the job. Another occupation that would be shrinking as automaton replace human labor.

Having the autonomous ground vehicle work as security drones would likely cause a temporary unemployment rate increase, as a sector of jobs is slowly replaced. However, like the many times this has happened before, these workers would find work elsewhere. In most situations this would cause these displaced unemployed workers to find work at a lower pay, the average salary of a security guard working the United States is roughly $28, 738. The salary of a person working minimum wage for the year is estimated at $15,080, therefore assuming they all end up in minimum wage jobs, whatever economic effect security workers losing around half their annual pay would be roughly the same as being replaced by these vehicles, not counting the generation of sales and production of the autonomous ground vehicle, and only considering if every single worker was replaced by these vehicles.

At our cost to make point of $250, these would also be an attractive buy for the municipal police departments or their respective SWAT teams. An automated self-navigating vehicle with the ability to send a live video feed back to a computer has practical purposes for scouting out high-danger locations. However, these are likely to be destroyed if found and need to be easily replaced and bought. Considering the price point of our device is currently $250, this is surely a low enough price for lowering the danger a previously blind trek into dangerous locations would have been.

Should the autonomous ground vehicle become a commercial product, it would be another sign that the privacy concerns of people across the world are more in line with their worries than they had hoped. The cost of building this product was $250, assuming there is also some profit to be made per device, and assuming we do not reduce the cost of each vehicle, then this would not be a price point that would be easily accessible to majority of people on Earth, even if it would not be outrageous for those in the first-world countries. It’s reasonable to expect that should this be purchased by a private, non-corporate entity, like for personal use, it would mainly be purchased by those with valuables that are worth significant amounts of money, creating a further divide in the rich and poor of the world, something that people are becoming increasingly aware of, assuming enough of these sell to create the controversy.

## 3.11 \* Contemporary Engineering Issues

Our project deals with a large range of modern engineering aspects, such as real-time sensing and processing and wireless data transmission. While there are many other engineering issues and ideas that are present, such as in all projects being developed, these issues play a key part in our project, as without any of these, the project would not have been successful at all. By having all of these aspects work at the same time, we have the core and most essential aspects of our project completed.

When creating an autonomous ground vehicle, what separates this from a simple RC car, is the autonomous aspect. The vehicle must be able to navigate and move on its own. By navigate and move on it’s own, we mean that the vehicle should not crash into other objects and should not require human input in order to move around and about a space. Should it encounter something blocking the path, it should be able to move past it, whether left, right, or backwards. For the vehicle to make the decisions required to move around, we used ultrasonic sensors to detect any obstacles on the sides and front of the vehicle. These ultrasonic sensors return a number that we use to determine the distance from the sensor to the obstacle that it has detected. Using this distance data from the sensors we then have the processor determine the correct movement, which is what creates real-time decision making from real-time sensors. The whole movement processing process is done at or in real-time, and allows the vehicle to move continuously, something we wanted to emphasize was the ability the car had to move without needing to stop for any processing, whether movement or otherwise. As everything in our world moves towards faster and faster computing, algorithms made by more mobile devices should move towards this near instantaneous processing and decision making, something we have definitely tried to do.

Our vehicle has a web-cam attached to the front of the car, that scans to the left and right of the vehicle while moving. This web-cam records everything it faces, and we send this video feed to a computer that is on the same network as the processor that is communicating with the web-cam. This wireless transmission can be done through any wireless connection, as long as the computer and the processor are on the same network. In addition to sending the video feed, the processor also needs to be able to receive manual human movement commands from the same computer with a controller plugged into it. The wireless communication is bi-directional in our project, the processor sends a video feed to the computer and receives commands from the computer should the computer tell the processor controlling the car to move into manual drive mode. Nowadays it is practically impossible to walk anywhere without running into a wireless network, whether it’s broadcast by a business, like Starbucks or McDonalds or every single hotel, home networks, or even wireless carriers, you might as well have a harder time avoiding a wireless network then trying to find one. The fact that our communication is done solely through a single wireless network is something that hardly limits its potential at all, and might as well not hamper it in the least.

## 3.12 \* Recognition of the need for and an ability to engage in lifelong learning

When we began brainstorming for this project, we knew there would be many things we would need to “look up” or brush up on, as we only had limited knowledge of how to program the processors or get communication between our devices. We were proven right, as the one thing that remained consistent throughout the project, the one constant in our ever changing plans, was the fact that the Internet was invaluable as a resource.

Whether it was finding open-source libraries for web-cam interaction, or looking up server-client connections and finding out how to upload programs for the two different processors we had, we were constantly using the Internet to look up information and learn new skills we would need to solve the problems we encountered. That is ultimately what we did for the majority of our time on the project, was trying to implement something, then running into a problem we had to look up for help on, and learning how to solve the problem with the new information we found on the Internet. Our classes at university were not nearly enough to know how to solve all the problems we encountered, we always had to learn more and build upon what we already knew in order to implement anything we thought of. The true accomplishment wasn’t getting the project working, it was learning how to do it.

## 3.13 \* Importance of Team Work

Our project team consists entirely of a single major, Computer Engineering. Considering that none of us, at the start of the project year, had taken too many Technical Electives, we had thought that our knowledge wouldn’t be too far apart from each other when it came to relevant to the project information. However, it became quite apparent that each of us had more skills or tolerance in one area than another, and we each brought separate skills to the project, even though our shared university knowledge wasn’t too far apart. Working towards our individual strengths, allowed us to divvy up the responsibilities in a more reasonable manner, and working together on the important aspects of the project allowed us to see that despite us being wrong in what we assumed each would know how to do well, we were able to utilize these differences as best as we could.

# **4 Experiment Design and Feasibility Study**

## 4.1 Experiment Design

The objective of this project is to create an Unmanned Ground Vehicle that was able to be remotely controlled as well as autonomously drive on its own while avoiding obstacles. It should also relay information about its current status back to a remote location, such as a laptop.

We created a design for our project and purchased parts for it. Then we learned how to use every hardware we purchased and built the vehicle. From there, we wrote code to make the vehicle capable of achieving our goal. Through various tests, we were able to fine tune our autonomous driven vehicle.

|  |  |
| --- | --- |
| **Responsibilities:** | **Team Members Involved:** |
| Raspberry Pi | Spencer Lee |
| Arduino Uno | Billy Xiao |
| Sensors | Ryan Sabik and Billy Xiao |
| Networking | Spencer Lee and Ryan Sabik |
| Hardware | Billy Xiao |
| Video Streaming | Ryan Sabik |

## 4.2 Experiment Results and Feasibility

Under-Work

# **5 Architecture**

## 5.1 System Architecture

This section provides a high level overview of the structural and functional decomposition of the system. Focus on how and why the system is decomposed in a particular way rather than on details of the particular components. Include information on the major responsibilities and roles the system (or portions thereof) must play. A pictorial representation of the architecture should be presented, which should show the hierarchical structure of the modules; interaction and interface among modules and with databases, external software, system, and networks

State clearly who is responsible for which module/task



# **6 \* High Level Design**

## 6.1 Conceptual View

This is our high level diagram of the Unmanned Ground Vehicle



Our system comprises of three main parts: the Arduino Uno subsystem, which directly controls our drive train, and the Raspberry Pi subsystem, which is our main central processing unit, and a laptop that we use as a remote location which the user is able to see statuses about the UGV. We also have a remote control that is hooked up to the laptop so we can manually control the UGV.

The Arduino Uno subsystem comprises of the Arduino Uno itself, the motor and wheel assembly as well as the motor drivers, and the Ultrasonic sensors. The Arduino Uno controls the Ultrasonic sensor, and based off the information these sensors return, the Arduino Uno then sends drive information to the motor driver, which directly controls the motor and wheel assembly.

The Raspberry Pi subsystem comprises of the Raspberry Pi itself and a couple of peripherals attached to it. The Raspberry Pi also communicates with the Arduino Uno via UART, which enables the Raspberry Pi to obtain drive statuses of the UGV. Attached to the USB slots, we have a Wi-Fi dongle attached as well as a webcam. The webcam provides live-video feed and all of the UGV statuses as well as the video feed is transmitted to our remote laptop via Wi-Fi.

The laptop subsystem consists of the laptop itself and the remote control. The laptop relays information that the Raspberry Pi knows and displays it for the user to see. We can stream a live-video feed and control the UGV without being right next to it. The laptop also establishes a network so the Raspberry Pi can communicate back and forth with the laptop via a Wi-Fi dongle attached to the Raspberry Pi's USB.

## 6.2 Hardware

The hardware involved are the following: The main processing unit of our UGV is the Raspberry Pi. Through the Raspberry Pi's peripherals, we have a webcam and a Wi-Fi dongle attached through the USB ports.

The next subsystem of our UGV is the drive train that is controlled by the Arduino Uno. The Arduino Uno communicates with the Raspberry Pi via UART. Through the Arduino Uno's available ports, we have sever Ultrasonic Sensors and motor drivers attached. The motor drivers drive the motor and wheel assembly of the UGV.

Lastly, we have our laptop that communicates to the UGV via Wi-Fi. We have a PlayStation remote connected to our laptop to manually control the UGV.

## 6.3 Software

The UGV has four main modules. These modules consisted of the operation of the vehicle (Drone\_Controller\_Module.py), the Server\_Controller\_module.py, the operation of the webcam on the UGV (Drone\_Webcam\_Module.py) and the code that was responsible for showing the video on the laptop (Server\_Webcam\_Module.py). There was also a supplementary Test\_Bench.py as well as a Combined\_Server.py code.

## 6.4 Responsibilities

The table below shows which person(s) were responsible for which module/task in both the hardware and the software associated with it.

|  |  |
| --- | --- |
| **Responsibilities:** | **Team Members Involved:** |
| Raspberry Pi | Spencer Lee |
| Arduino Uno | Billy Xiao |
| Sensors | Ryan Sabik and Billy Xiao |
| Networking | Spencer Lee and Ryan Sabik |
| Hardware | Billy Xiao |
| Video Streaming | Ryan Sabik |

# **7 Data Structures**

n/a

# **8 \* Low Level Design**

For all the Low Level Design, the chart in the High Level Design covers the responsibilities of each module/task.

## 8.1 Raspberry Pi - The Main Processing Unit Module

This module is the Raspberry Pi. The Raspberry Pi is the core of our UGV.

### 8.1.1 Processing narrative for the Main Processing Unit module

### 8.1.2 Main Processing Unit Module interface description.

### 8.1.3 Main Processing Unit Module processing details

## 8.2 The Arduino - Drive Train Module

This module is the Arduino Uno. The Arduino Uno controls how the UGV drives.

### 8.2.1 Processing narrative for the Arduino - Drive Train module

### 8.2.2 Arduino - Drive Train Module interface description.

### 8.2.3 Arduino - Drive Train Module processing details

## 8.3 The Communication Module

This module is the Arduino Uno. The Arduino Uno controls how the UGV drives.

### 8.3.1 Processing narrative for the Communication Train module

### 8.3.2 The Communication Module interface description.

### 8.3.3 The Communication Module processing details

## 8.4 The Ultrasonic Sensor Module

### 8.4.1 Processing narrative for the Ultrasonic Sensor module

### 8.4.2 Ultrasonic Sensor Module interface description.

### 8.4.3 Ultrasonic Sensor Module processing details

## 8.5 The Live-Video Feed/Camera Module

### 8.5.1 Processing narrative for the Live-Video Feed module

### 8.5.2 Live-Video Feed Module interface description.

### 8.5.3 Live-Video Feed Module processing details

## 8.6 The Remote Control Module

### 8.6.1 Processing narrative for the Remote Control module

### 8.6.2 Remote Control Module interface description.

### 8.6.3 Remote Control Module processing details

# **9 User Interface Design**

## 9.1 Application Control

n/a

# **10 \* Test Plan**

Formal Test are still being written.

## 10.1 \* Design of Tests

10.1.1 Test Case 1: Remote Manual Commands

1. The objective of this test is to verify that the Autonomous Ground Vehicle is able to receive a motor or servo command from the remote station and act on that command within 0.5 seconds time.
2. The Autonomous Ground Vehicle shall be placed in an open area. The remote station will then connect over 802.11 wireless to the AGV.
3. Once the remote station has successfully connected to the AGV, the station operator will attempt to issue a movement command to the AGV via controller. The AVG is to perform the issued command, whether it be go, stop, turn, or rotate its camera within 0.5 seconds of the command being issued.
4. Our expected result is that when the AGV is issued the command, it will perform the specified action within 0.5 seconds.

Spencer and Ryan are responsible for testing wireless communication.

10.1.2 Test Case 2: Video Stream Over Wireless

1. The objective of this test is to verify that the Autonomous Ground Vehicle is able to send a video stream from the AGV to the remote station. The the delay on the video stream must be less than 2 seconds.
2. The AGV is placed on a rotating platform in an area with 3 distinct objects placed around the AGV with the AGV facing one of the objects.
3. Once the AGV is connected to the remote station, the remote station shall begin displaying a video stream from the webcam on the AGV. While the video stream is active we rotate the AVG using the rotating platform so it faces a different object. We then measure the time taken for the video stream on the remote station to display what is being captured by the AGV’s camera.
4. Our expected result is that the video stream on the remote station should take at most 2 seconds to update to the image being captured by the camera.

Spencer is responsible for testing wireless video streaming.

10.1.3 Test Case 3: Automatic Collision Avoidance

1. The objective of this test is to verify that the AGV is able to avoid obstacles using the ultrasonic sensors.
2. The AGV is place in an area surrounded by walls with a several blocks scattered around the area.
3. Once activated the AGV should move through the area. The AGV should stop within 15 cm of any solid block or wall directly in front of it. Then the AGV should turn such that the front of the vehicle is no longer obstructed. Next the AGV shall continue to move forward until it encounters another solid object.
4. We expect the AGV to successfully avoid obstacles in our test environment.

Billy is responsible for developing the sensors. Spencer and Ryan are responsible for developing the avoidance algorithm.

10.1.4 Test Case 4: Detection of Inclines

1. The objective of this test is to verify that the AGV’s collision avoidance algorithm is able to differentiate between ramps and inclines and walls.
2. This test assumes that the AGV has successfully completed Test Case 3 (10.1.3). The AGV is placed in a an enclosed area with three inclines of varying grade (10, 20, 30 degrees) .
3. The AGV is directed to the base of a ram and is then set to automatic navigation.
4. Our expectations are that the AGV shall be able to differentiate between incline and wall. Thus the AGV should be able to automatically navigate itself up an incline up to a maximum of 30 degrees instead of interpreting the incline as a wall and turning to avoid it.

Billy is responsible for developing the sensors. Spencer and Ryan are responsible for developing the avoidance algorithm.

## 10.2 \* Bug Tracking

|  |  |  |
| --- | --- | --- |
| Bug Number | Bug Description | Assignment to Fix/Investigate |
| 1 | AGV receives commands on a delay. | Spencer Lee |
| 2 | Video stream from AGV to remote station has low framerate. | Spencer Lee and Ryan Sabik |
| 3. | AGV fails to detect surfaces while moving on carpet. | Billy Xiao |
| 4. | AGV fails to detect walls with rubber coating. | BIlly Xiao |
| 5. | AGV unable to differentiate incline | Ryan Sabik |
| 6. | Motor drivers frequently overheat. | Ryan Sabik |

## 10.3 \* Quality Control

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Track Number | Pass/Fail | Deviations from Expected Results | Date of Failure | Date of Successful Retest |
| 10.1.1 | FAIL | AGV operated on a delayed response to the Remote Station | 11/5/14 | 11/22/14 |
| 10.1.2 | PASS | AGV successfully sends an a video stream to the remote station with a < 2 second delay. Framerate a bit low. |  | 11/29/14 |
| 10.1.3 | PASS | AGV successfully navigates area avoiding walls and other obstacles |  | 3/11/15 |
| 10.1.4 | FAIL | AGV will not automatically climb an incline over 10 degrees. | 3/16/15 |  |

## 10.4 Performance Bounds

In order for the AGV to avoid obstacles, it must be in an environment that does not absorb nor alter ultrasonic waves.

## 10.5 \* Identification of critical components

Super critical component to keep an eye on at all times are the motor drivers. The motor drivers we were using have a tendency to draw a lot of current and fry up the circuit board. It also gets very hot.

## 10.6 \* Items Not Tested

Speed sensors have not received formal testing due to incomplete implementation.

# **11 \* Test Report**

## 11.1 \* Test Iteration 1

### 11.1.1 Test 1: Remote Manual Commands

Testers: Spencer Lee

1. Upon issuing a command at the remote station, the AGV took considerable time (> 2 seconds) to respond. Upon issuing multiple commands, the AGV would still take considerable time to respond and would then execute the all the given commands in a rapid burst.
2. Test results did not align with our expectations, The AGV responded considerably slower than desired.
3. From our findings, the TCP connection we currently use for the sending commands will wait for multiple commands and attempt to ‘bundle’ the commands together. This leads to the delayed response and rapid execution of multiple commands.
4. To address this issue we will switch over to a UDP connection for the sending of commands.

**11.1.2 Test 2: Video Stream Over Wireless**

Testers: Spencer Lee

1. After successfully connecting we observed that the wireless video stream does operate within the 2 second delay range. However we noted that the framerate is a bit low causing sharp transitions.
2. In terms of delay, the test results met our expectations.
3. Despite meeting our expectations in terms of time delay, the framerate was far too low to be suitable. The sharp transitions when moving the camera would prevent the operator from accurately controlling the AGV.
4. We have two plans to address the issue of lower framerate. The first will be to reduce the time delay between images when sending the images. The second plan is to change the connection to UDP and see if this mode of communication will improve framerates.

**11.1.3 Test 3: Automatic Collision Avoidance**

Testers: Billy Xiao

1. Our observations show that the AGV can successfully navigate an area with cardboard obstacles without colliding with any walls or or blocks.
2. In this test our observed results met with our expected results.
3. We successfully tested that the AGV can avoid obstacles in this environment. However, we should modify the environment to ensure that the AGV can operate in other environments as well.
4. To expand on our testing we plan to change certain variable such as flooring and types or walls to see how the AGV will respond to differing environments.

**11.1.4 Test 4: Detection of Inclines**

Testers: Ryan Sabik

1. Our observations indicate that the AGV will on occasion detect inclines as walls and will attempt to maneuver around them rather than go up them. Our observations show that the AGV will always move up the 10 degree incline. The AGV would only occasionally move up the 20 degree incline. Repeat tests show an approximate 70/30 split in favor of avoiding the incline. Lastly the 30 degree incline would always be detected as an incline.
2. The results of the test did not fully meet our expectations.
3. Due to the forward facing ultrasonic sensors, the AGV is less likely to be able to differentiate between an incline and a wall.
4. We expect that we would either need to modify the location of the ultrasonic sensors or change the detection algorithm to address this issue.

**11.2 \* Test Iteration 2**

### 11.2.1 Test 1: Remote Manual Commands

Testers: Spencer Lee

1. Once again, considerable time (> 2 seconds) to respond. Upon issuing multiple commands, the AGV would still take considerable time to respond and would then execute the all the given commands in a rapid burst.
2. Unfortunately the test results did not meet with expectations.
3. In our debugging we have determined that that the issue was we were sending too many commands, too quickly. As a result we would overflow the receive buffer on the UGV causing commands to drop needing needing more time to process extra commands.
4. To fix this we plan on adding a delay between the sending of commands so we do not overflow the buffer.

**11.2.2 Test 2: Video Stream Over Wireless**

Testers: Spencer Lee

1. After changing the implementation to UDP we had considerable worse performance. The delays were much greater than 2 seconds and the framerate did not appear to improve at all.
2. The results did not meet our expectations.
3. Unlike TCP, UDP has no innate error checking and no guarantee that the packets sent will arrive in proper order. Due to the size of the images, which requiring over 600 packets to complete, many of the packets would be lost thus creating lost frames. These lost frames translate into longer delays and equal if not worse framerate.
4. Due to inexperience in creating an error and loss handling system for UDP packets, we have chosen to return to TCP and to simply reduce the delay between images.

**11.2.3 Test 3: Automatic Collision Avoidance**

Testers: Billy Xiao

1. In this iteration of the test we shall keep the same walls and obstacles but the flooring will be carpet instead. Our observations show that the AGV would fail to detect both walls and blocks. The AGV would neither stop nor attempt to path around the object.
2. The outcome of this test did not meet our expected result.
3. As the AGV uses ultrasonic sensors to detect objects, we believe that the carpet is absorbing the the ultrasonic pulses. Thus, the AGV is unable to detect any walls or floors due to the lack of echo from the ultrasonic pulses.
4. This is an issue with the the hardware being incompatible with the environment, so it appears that it will not be addressable.

**11.2.4 Test 4: Detection of Inclines**

Testers: Ryan Sabik

1. In this test we have made an alteration to the test environment changing the inclines to 10, 15, 20 degrees instead. Additionally we have make slight changes to the detection algorithm. From our findings, the AGV will still climb the 10 degree incline and it will also climb the 15 degree incline, but the AGV will still not climb the 20 degree incline 100% of the time.
2. Our test results showed minor improvement but did not fully meet our expectations
3. Because of the ultrasonic sensors, the AGV is less likely to be able to differentiate between an incline and a wall.
4. Continued refinement of this test could be very time consuming. We have chosen to accept that the AGV can automatically traverse at most 15 degree incline.

**11.2 \* Test Iteration 3**

### 11.3.1 Test 1: Remote Manual Commands

Testers: Spencer Lee

1. The AGV successfully responds to issued commands with no noticeable delay (no delay greater than 0.5 seconds).
2. The outcomes have met our expectations.
3. By not sending as many UDP packets, we stopped overflowing the buffer allowing the AGV to properly process the commands.
4. Our project requirements have been met no further testing of this system will be needed.

**11.3.2 Test 2: Video Stream Over Wireless**

Testers: Spencer Lee

1. By reducing the delay between between frames the overall video stream was much smoother.
2. Although not completely smooth, we can say that the framerate combined with the < 2 second delay is within an operable level. So the results of this test met our expectations.
3. The reduction of the delay has shown that the bottleneck for framerate was in how frequently we send images over the connection. By reducing the delay we are able a achieve a smoother framerate.
4. Our project requirements have been met no further testing of this system will be needed.

**11.3.3 Test 3: Automatic Collision Avoidance**

Testers: Billy Xiao

1. This iteration of the test, we shall keep linoleum flooring but we shall use rubber padded walls.Our observations show that the AGV would still detect and avoid blocks, but it would fail to detect the walls.
2. Because the AGV could not detect the walls, it did not meet our expectations.
3. As the AGV uses ultrasonic sensors to detect objects, we believe that the rubber lined wall does not reflect ultrasonic pulses properly. Thus, the AGV is unable to detect any walls lack of echo from the ultrasonic pulses.
4. This is an issue with the the hardware being incompatible with the environment, so it appears that it will not be addressable.

# **12 Administrative and Other Design Issues**

## 12.1 \* Project Management

One paragraph from each team member

How was the project managed, how was tasks distributed, how was scheduled made, what project management software/method did you use, your experience in working together, what have you learned in terms of team work, time management, project management etc., from this project.

# **13 \* Conclusion and Future Work**

## 13.1 \* Conclusion

## 13.2 \* Acknowledgement

In completing our project, we would like to acknowledge the following for lending their time and knowledge with helping us completing our project:

Dr. Gang Chen

Pavle Kirilov

Google

Developers of PyGame, PySerial, and OpenCV

Chris Dube, Benson Ninh, and Calvin Nguyen

# **14 \* References**

Ultrasonic Sensors:

<http://arduino.cc/en/Reference/pulseIn>

The User manual for Ultrasonic Sensor I found online, attached from our Google Drive:

https://drive.google.com/open?id=0B5AsbGmynAfNOUhQU3RKSHZkOWM&authuser=1

Speed Sensor:

https://sites.google.com/site/myscratchbooks/home/projects/project-11-infrared-speed-sensing-module

# **15 Appendices**

**Appendix A:** Parts List

|  |
| --- |
| **Parts** |
| Raspberry Pi |
| Arduino Uno |
| Vehicle with chassis, motor, wheel |
| Motor Drivers |
| Webcam |
| Ultrasonic Sensor |
| Servo |
| PlayStation Remote Controller |
| Speed Detector Sensor Module |
| Battery |

**Appendix B:** Equipment List

|  |
| --- |
| **Parts** |
| Laptop |
| Digital Multimeter |
| GoPro Camera |
| Soldering Iron |
| Hot Glue Gun |
| Screwdrivers |
| Dremel |
| External Router |

**Appendix C:** Software List

|  |
| --- |
| **Parts** |
| Eclipse IDE |
| Arduino Environment |
| Notepad++ |
| Python Packages |
| Pygame Library |
| Arduino Library |
|  |
|  |
|  |

Drone\_Controller\_Module.py

import pygame

import socket

import struct

import serial

import time

def joystickSetup():

pygame.joystick.init()

joystick = pygame.joystick.Joystick(0)

joystick.init()

return joystick

def UDPServerInit(host, port):

try:

sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

except socket.error as msg:

print ('Socket Create failed')

print(msg)

sock.close()

pygame.quit()

exit()

#s.setsockopt(socket.SOL\_SOCKET, socket.SO\_BROADCAST, 1)

try:

sock.bind((host,port))

except socket.error as msg:

print ('Bind failed')

print(msg)

sock.close()

pygame.quit()

exit()

return sock

class Drone\_Receiver:

def \_\_init\_\_(self, host = '', port = 1881, local\_control = False):

self.unpacker = struct.Struct('f f f f ? ? ? ?')

self.packer = struct.Struct('B')

self.local = local\_control

self.SERVERHOST = host

self.SERVERPORT = port

self.sock = UDPServerInit(self.SERVERHOST, self.SERVERPORT)

self.uart = serial.Serial("/dev/ttyAMA0",baudrate = 9600, timeout = 0.03)

self.uart.open()

self.joystick = None

if local\_control == True:

self.joystick = joystickSetup()

self.done = False

print("Remote Server is active")

print('Host Address: ', self.SERVERHOST)

print('Port Number: ', self.SERVERPORT)

def ClassExit(self):

self.sock.close()

self.uart.close()

pygame.quit()

exit()

def ClassClose(self):

self.sock.close()

self.uart.close()

def determineVelocity(self, axisx, axisy):

direction = 0

magnitude = 0

#scaling speed backward

if axisy > 0.20 and axisx < 0.20 and axisx > -0.20:

direction = 4

magnitude = 1 if axisy > 0.70 else 0

#scaling speed forward

elif axisy < -0.20 and axisx < 0.20 and axisx > -0.20:

direction = 1

magnitude = 1 if axisy < -70 else 0

#scaling rotational speed right

elif axisx > 0.20 and axisy < 0.20 and axisy > -0.20:

direction = 2

magnitude = 1 if axisx > 0.70 else 0

#scaling rotational speed left

elif axisx < -0.20 and axisy < 0.20 and axisy > -0.20:

direction = 3

magnitude = 1 if axisy < -0.70 else 0

else:

direction = 0

magnitude = 0

velocity = (direction) | (magnitude << 3)

return velocity

def determineServo(self, axis\_rx, axis\_ry, center):

servo\_action = 0

if center == True:

servo\_action = 7

elif axis\_rx < -0.3 and axis\_ry < 0.3 and axis\_ry > -0.30:

servo\_action = 2

elif axis\_rx > 0.3 and axis\_ry < 0.3 and axis\_ry > -0.30:

servo\_action = 1

elif axis\_ry > 0.3 and axis\_rx < 0.30 and axis\_rx > -0.30:

servo\_action = 4

elif axis\_ry < -0.3 and axis\_rx < 0.30 and axis\_rx > -0.30:

servo\_action = 3

else:

servo\_action = 0

return servo\_action

def testConnection(self):

try:

print('Waiting for connection...')

data = self.sock.recvfrom(1024)

message = data[0].decode('UTF-8')

address = data[1][0] + ":" + str(data[1][1])

print('Signal from: ' + address)

print('Signal: ' + message)

print('Signal Received, Connection Successful')

except KeyboardInterrupt:

self.ClassExit()

def RunSingleIteration(self):

axis\_x = axis\_y = axis\_rx = axis\_ry = 0.0

tog\_auto = tog\_man = center = status = False

for event in pygame.event.get(): # User did something

# Possible joystick actions: JOYAXISMOTION JOYBALLMOTION JOYBUTTONDOWN JOYBUTTONUP JOYHATMOTION

if event.type == pygame.QUIT: #and event.key == pygame.K\_ESCAPE:

self.done = True

if self.local == True:

axis\_x = self.joystick.get\_axis(0)

axis\_y = self.joystick.get\_axis(1)

axis\_rx = self.joystick.get\_axis(2)

axis\_ry = self.joystick.get\_axis(5)

but\_share = self.joystick.get\_button(8)

but\_option = self.joystick.get\_button(9)

center = self.joystick.get\_button(11)

tog\_auto = self.joystick.get\_button(3)

tog\_man = self.joystick.get\_button(1)

if but\_share == True and but\_option == True:

status = True

else:

status = False

else:

packet, addr = self.sock.recvfrom(self.unpacker.size)

axis\_x, axis\_y, axis\_rx, axis\_ry, center, status, tog\_auto, tog\_man = self.unpacker.unpack(packet)

if status == True:

self.done = True

velocity = self.determineVelocity(axis\_x, axis\_y)

servo\_action = self.determineServo(axis\_rx, axis\_ry, center)

state\_switch = 0

if tog\_auto == True:

uart\_set = 0x81

print("Setting Mode to Auto")

elif tog\_man == True:

uart\_set = 0x82

print("Setting Mode to Manual")

else:

uart\_set = (velocity) | (servo\_action << 4)

uart\_signal = self.packer.pack(uart\_set)

#print("UART Signal: {0:b}".format(uart\_signal))

self.uart.write(uart\_signal)

'''

serial\_recv = self.uart.read(1)

if(serial\_recv != b''):

state\_rpm = self.packer.unpack(serial\_recv)[0]

state = "Automatic" if (state\_rpm & 0xC0) == 0x80 else "Manual"

rpm = state\_rpm & 0x3F

print("Mode: {}".format(state))

print("RPM: {}".format(rpm))

'''

Combined\_Server.py

import pygame

import Server\_Controller\_Module

import Server\_Webcam\_Module

import time

import threading

server\_quit = False

def ControllerThreadLoop(Controller):

global server\_quit

print("Controller Module Now Running\n")

while Controller.done == False and server\_quit == False:

try:

Controller.RunSingleIteration()

time.sleep(0.1)

except KeyboardInterrupt:

server\_quit = True

exit()

server\_quit = True

exit()

def CameraThreadLoop(Camera):

global server\_quit

while Camera.done == False and server\_quit == False:

try:

Camera.RunSingleIteration()

time.sleep(0.03)

except KeyboardInterrupt:

server\_quit = True

exit()

server\_quit = True

exit()

pygame.init()

Controller = Server\_Controller\_Module.Drone\_Controller('192.168.0.205',1881)

Server\_Cam = Server\_Webcam\_Module.Webcam\_Receiver('',9001)

Control\_thread = threading.Thread(target=ControllerThreadLoop,args=(Controller,))

Control\_thread.start()

CameraThreadLoop(Server\_Cam)

Control\_thread.join()

Controller.ClassClose()

Server\_Cam.ClassClose()

pygame.quit()

exit()

Server\_Controller\_Module.py

import pygame

import socket

import struct

import time

def joystickSetup():

pygame.joystick.init()

joystick = pygame.joystick.Joystick(0)

joystick.init()

return joystick

def clientSetup():

try:

sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

except socket.error:

print("Failed to create socket")

sock.close()

pygame.quit()

exit()

sock.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

return sock

def UDPSend(sock, HOST, PORT, packet):

try:

sock.sendto(packet,(HOST, PORT))

except socket.error as msg:

print(msg)

class Drone\_Controller(object):

def \_\_init\_\_(self, host = '', port = 1881):

self.SERVERHOST = host

self.SERVERPORT = port

self.sock = clientSetup()

self.joystick = joystickSetup()

self.packer = struct.Struct('f f f f ? ? ? ?')

self.done = False

def ClassExit(self):

self.sock.close()

pygame.quit()

exit()

def ClassClose(self):

self.sock.close()

def TestConnection(self):

try:

print("Sending connection Signal...")

signal = 'Remote Signal'

UDPSend(self.sock, self.SERVERHOST, self.SERVERPORT, signal.encode('UTF-8'))

except socket.error as msg:

print(msg)

self.ClassExit()

except KeyboardInterrupt:

self.ClassExit()

input('Press Enter to Continue')

def RunSingleIteration(self):

for event in pygame.event.get(): # User did something

# Possible joystick actions: JOYAXISMOTION JOYBALLMOTION JOYBUTTONDOWN JOYBUTTONUP JOYHATMOTION

if event.type == pygame.KEYDOWN: #and event.key == pygame.K\_ESCAPE:

self.done = True

exit\_button = False

axis\_lx = self.joystick.get\_axis(0)

axis\_ly = self.joystick.get\_axis(1)

axis\_rx = self.joystick.get\_axis(2)

axis\_ry = self.joystick.get\_axis(3)

but\_share = self.joystick.get\_button(8)

but\_option = self.joystick.get\_button(9)

but\_cam\_stick = self.joystick.get\_button(11)

tog\_auto = self.joystick.get\_button(1)

tog\_man = self.joystick.get\_button(3)

if but\_share == True and but\_option == True:

self.done = True

exit\_button = True

print("Ending Connection")

axis\_pair = (axis\_lx, axis\_ly, axis\_rx, axis\_ry, but\_cam\_stick, exit\_button, tog\_auto, tog\_man)

axis\_pack = self.packer.pack(\*axis\_pair)

UDPSend(self.sock, self.SERVERHOST, self.SERVERPORT, axis\_pack)

Server\_Webcame\_Module.py

import socket

import pygame

pygame.init()

def serverInit(host, port):

try:

sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

except socket.error as msg:

print ('Socket Create failed')

print(msg)

sock.close()

pygame.quit()

exit()

try:

sock.bind((host,port))

except socket.error as msg:

print ('Bind failed')

print(msg)

sock.close()

pygame.quit()

exit()

return sock

class Webcam\_Receiver(object):

def \_\_init\_\_(self, host = '', port = 9001):

self.SERVERHOST = host

self.SERVERPORT = port

self.screen = pygame.display.set\_mode((640,360),0)

self.sock = serverInit(self.SERVERHOST, self.SERVERPORT)

self.sock.listen(2)

self.done = False

print("Remote Server is active")

print('Host Address: ', self.SERVERHOST)

print('Port Number: ', self.SERVERPORT)

def ClassExit(self):

self.sock.close()

pygame.quit()

exit()

def ClassClose(self):

self.sock.close()

def RunWebcamServer(self):

try:

while True:

connection, addr = self.sock.accept()

received = []

while True:

chunk = connection.recv(691200)

if not chunk:

break

else:

received.append(chunk)

frame\_string = b''.join(received)

frame\_image = pygame.image.fromstring(frame\_string,(640,360),"RGB")

self.screen.blit(frame\_image,(0,0))

pygame.display.update()

for event in pygame.event.get():

if event.type == pygame.QUIT:

self.ClassExit()

except KeyboardInterrupt:

self.ClassExit()

def RunSingleIteration(self):

for event in pygame.event.get():

if event.type == pygame.QUIT:

self.done = True

connection, addr = self.sock.accept()

received = []

while True:

chunk = connection.recv(691200)

if not chunk:

break;

else:

received.append(chunk)

frame\_string = b''.join(received)

frame\_image = pygame.image.fromstring(frame\_string,(640,360),"RGB")

self.screen.blit(frame\_image,(0,0))

pygame.display.update()

Drone\_Webcame\_Module.py

import socket

import pygame

import pygame.camera

import time

pygame.init()

pygame.camera.init()

class Drone\_Webcam(object):

def \_\_init\_\_(self, host = '', port = 9001):

self.SERVERHOST = host

self.SERVERPORT = port

self.camlist = pygame.camera.list\_cameras()

self.webcam = pygame.camera.Camera(self.camlist[0],(640,360))

self.webcam.start()

self.sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

self.sock.close()

self.done = False

def ClassExit(self):

self.sock.close()

pygame.quit()

exit()

def ClassClose(self):

self.sock.close()

def ConnectServer(self):

try:

self.sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

except socket.error as msg:

print ('Socket Create failed')

print(msg)

self.ClassExit()

try:

self.sock.connect((self.SERVERHOST, self.SERVERPORT))

except KeyboardInterrupt:

self.ClassExit()

def RunWebcam(self):

try:

while True:

image = self.webcam.get\_image()

image\_string = pygame.image.tostring(image, "RGB")

self.ConnectServer()

self.sock.sendall(image\_string)

self.sock.close()

time.sleep(0.04)

#=======================================================================

# except socket.error as msg:

# print("Socket Error")

# pass

#=======================================================================

except KeyboardInterrupt:

self.ClassExit()

def RunSingleIteration(self):

image = self.webcam.get\_image()

image\_string = pygame.image.tostring(image,"RGB")

self.ConnectServer()

self.sock.sendall(image\_string)

self.sock.close()

Test\_Bench.py

import pygame

import Drone\_Controller\_Module

import Drone\_Webcam\_Module

import time

import threading

import socket

drone\_quit = False

def ControllerThreadLoop(Controller):

global drone\_quit

while Controller.done == False and drone\_quit == False:

try:

Controller.RunSingleIteration()

time.sleep(0.05)

except KeyboardInterrupt:

drone\_quit = True

exit()

drone\_quit = True

exit()

def CameraThreadLoop(Camera):

global drone\_quit

while Camera.done == False and drone\_quit == False:

try:

Camera.RunSingleIteration()

time.sleep(0.04)

except (KeyboardInterrupt, socket.error):

drone\_quit = True

exit()

drone\_quit = True

exit()

pygame.init()

Controller = Drone\_Controller\_Module.Drone\_Receiver('',1881, False)

#ControllerThreadLoop(Controller)

Drone\_Cam = Drone\_Webcam\_Module.Drone\_Webcam('192.168.137.1',9001)

Control\_thread = threading.Thread(target=ControllerThreadLoop,args=(Controller,))

Camera\_thread = threading.Thread(target=CameraThreadLoop,args=(Drone\_Cam,))

Control\_thread.start()

input("Press Enter to start Webcam Thread")

Camera\_thread.start()

Control\_thread.join()

Camera\_thread.join()

Controller.ClassClose()

Drone\_Cam.ClassClose()

pygame.quit()

exit()