# Swarming Robots

May 1308

Douglas Feldmann
Matt Henderson
Matt Klinkefus
Zach Njus

# Table of Contents

# List of Figures

# Executive Summary

Every day in locations around over the globe, human beings place themselves in situations dangerous to their health for the benefit of others. Examples of this would be soldiers going to war, police officers performing their duties, and rescue workers trying to save the lives of those affected by disasters, natural or otherwise. Technology exists to make the lives of a population safer, but these people risk life and limb perform their duties because a replacement for the human body and mind has yet to be created.

Even though creating something with the physical and mental abilities is still beyond our capabilities, creating tools to reduce the risks inherent in such dangerous situations is not only possible, but achievable. The aim of this project is to create a number of vehicles that can not only cooperate to accomplish simple missions, but adapt to changing situations in real time.

# Problem Statement

With military zones disaster zones becoming increasingly dangerous, having autonomous robots helping to search areas without needing to put human life in danger will be a very important asset. The goal for this project was to build a swarm of mobile robots that interface with the world via a “command center” laptop. This swarm is able to receive missions from the command center and automatically determine how they are going to search the area and deploy themselves internally. The robots have GPS sensors and a webcam to be able to navigate and search the given area. Our team also needed to develop the communication lines from the swarm to the command center, the control code for the swarm, the mission logic software and the command center interface.

# Acknowledgments

Lockheed Martin supplied the necessary funding to complete this project. Lockheed Martin provided technical expertise when needed.

# Operating Environment

The testing environment for the project will not be extreme conditions. A wide open field free of any substantial obstacles will serve as the testing grounds for the hardware and network. This area was chosen because the goal of this project is not to extensively test the hardware and mobility of the vehicles; it is to test the process of the vehicles communicating over an ad-hoc network while performing a mission. Since the open field will not provide any real issues for the project virtual obstacles will be provided such as, losing network connection or vehicle malfunction, and physical objects will be provided for the vehicles to detect.

# Assumptions

### Network

1. Maximum number of nodes – 254
2. Testing area of one square mile
3. Minimal radio interference in testing area
4. All instructions and data must be sent over ad-hoc network

### Vehicle

1. Will not be tested in harsh environment
    a. Temperature will not exceed 110 degrees Fahrenheit
    b. Water will not be an issue
    c. Terrain will be easily traversable
2. *The system will only be run on the hardware selected by the team.*
3. Must be able to run for 30 minutes continuously
4. Must send and receive all commands and data over ad-hoc network

# Limitations

**Time** - Two semesters to research, plan , design, implement and test the system
**Money** - Approximately $3000 dollars in our design budget
**Experience** - The team has collective knowledge building software, working with hardware, some experience with networks and exposure to embedded systems programming.

# Functional Requirements

## High priority:

1. The network must be able to remain continuously connected via Wi-Fi
2. If a node drops out of the network; each node must calculate where to move in order to repair the network, inferred based on the previously recorded GPS coordinates of the other nodes.
3. If a node drops out of the network, all remaining nodes must be informed about the loss within 20 seconds
4. The network must be ad-hoc
5. A node must be able to join another node in an ad-hoc network within 10 seconds of coming within range of it
6. Nodes in the network must be able to work together to complete mission objective, which may include directing the network to a location or issuing commands to a node's sensor.
7. Routes must be planned and re-planned dynamically as needed
8. Use sonar sensors to detect obstacles larger than the radius of the wheels within 2 meters from the RC car
9. Ability to autonomously drive within 5 meters of a given coordinate that is reachable with the current battery life
10. Control point of view of onboard camera with a lateral range of +/- 180 degrees from front of car and vertical range of +/- 45 degrees from a plane parallel to the car
11. The command center issues a mission assignment to the vehicles, from a predefined list
12. Command center can display multiple views (GPS map, video feed, mission status, sensor values)
13. Minimum number of vehicles – 4
14. Each vehicle must have general payloads(video feed, image capture/recognition, etc)
15. Each vehicle must have a unique payload, physical or virtual (robotic arm, bucket, water bottle, etc)

## Medium Priority

1. Data must be continuously shared about each node with all other nodes in the network
2. Each node must continuously track information about all other nodes
3. The system must be able to share output from any arbitrary sensor type
4. At any point in time, the current information that a node has about each other node in the network must not be older than 20 seconds
5. Transmit all data between RC cars and Command center up to a range of 70 unobstructed meters
6. Ability to determine location within 5 meters accuracy while stopped or 8 meters while moving
7. Ability to process 240p streaming video at 15 fps minimum with 16 bit color (minimum color requirements for an android phone)
8. Camera controller should be able to rotate 180 degrees in 1 second for both vertical and horizontal rotation
9. The RC car must be able to maintain a minimum speed of 3.4 mph (standard march speed)
10. Monitoring of vehicle information at a defined interval (GPS location, battery life, etc)
11. Command center needs to log each vehicles information for mission summary report

# Non-Functional Requirements

1. RC car can weigh no more than 10 lbs
2. The RC car shall run on electric motors
3. Locations will be determined by GPS coordinates
4. System must support the 802.11n and g WiFi protocols
5. The system must support up to 254 nodes
6. The system must be able to detect when data from the network has been corrupted and notify the sender
7. The system must support distances of 75 yards between nodes in ideal conditions
8. Bots must have enough battery capacity for 30 minutes of constant operation

# Overall Platform Setup

The different system setups that were considered for the project include using a single board computer, a phone, or a micro controller and router combination as the main devices on the car. These three options need to allow for either video processing or just relaying the video feed. Each setup would also include the following modules with it:

- Servos for aiming the camera
- Speed controllers for brush-less DC motors
- Servo for control of the steering mechanism
- GPS for location information
- Electronic compass to help compensate for incorrect GPS readings and bearing while at a standstill.
- Wireless module
- Main computing system e.g. phone, single board computer, microprocessor, etc.
- Camera for streaming video, and still photos
- Sensors for object detection and avoidance

## Option 1

Option one is to use an Android phone as the main computing platform on the car, then using the IOIO or similar development platform create an interface for all other hardware to be controlled from the phone

- Pros
  - The phone would include a wireless communication module, camera, GPS, and a more than capable processor.
  - The phone's power system could be separate from the motors' power system, allowing for easier integration
- Cons

- This setup would have two separate power systems, which could be a hassle for recharging and other maintenance
- The phone would have to be rotated when the operator wants to control the camera, which could damage control lines for propulsion system
- The phone's screen and many other parts would be wasted, and therefore add unneeded weight.

## Option 2

Option two is to use a single board as the main computing platform for the RC car. This would allow for enough processing power to handle streaming video feeds. A microprocessor would then be used for all of the low level control of the car such as motors. This approach would allow for more creativity because the project would not be limited by what the IOIO board would give access to, but would require a greater effort for implementation.

- Pros
  - Could allow for better implementation of all devices, since it will not depend on what can be done from a phone, and would have full access to all unused processor I/O on the board
  - Less wasted space and weight
  - A single board computer is less expensive than buying a cell phone without a contract
  - Only one power system to worry about
- Cons
  - More implementation overhead, more time

## Option 3

Option three is to use a microprocessor as the main computation unit. This would mean that the microcontroller would not be able to process the video feeds. If this option were used, an IP based camera system would need to be used, and a router would need to be placed on the car. This would draw more power, consume more space, and be a less elegant and more awkward setup.

- Pros
  - A microprocessor would use much less power
  - A microprocessor would take up much less space
  - Setting up I/O for most microprocessors is fairly simple
- Cons
  - Less available computation power
  - A more expensive IP style camera would be needed
  - Even more overhead to implement wireless protocols
  - Less space for program memory

## Conclusion

From reviewing the preceding three options that were considered, it is apparent that the second option would be the best suit for this project. It will allow for the necessary processing power for the image recognition and for other components to be implemented easily. It will also add an expandability factor that

could be helpful for further improving on the design.

# Development Platform

Since the project requires the capability to process video taken from a mounted USB webcam, other data from the RC car, and relay other RC car's data in an Ad-Hoc manner, it requires a processor capable for this workload. A simple Arduino would be ideal if the project did not have to support streaming video. Since Arduino does not come close to the processing power needed, a few high end embedded development platforms came into consideration.
To make a good justification of what development platform should be used, three different boards were compared: Gumstix, Beagleboard, and PandaBoard. Below is a table detailing the specifications of each board.

| | Beagleboard-xM | PandaBoard | Gumstix Overo Fire |
|---|---|---|---|
| **Core Logic** | | | |
| **Processor** | TI ARM Cortex A8 | TI ARM Cortex A9 MPCore with SMP (symmetric multiprocessing) | TI ARM Cortex A8 |
| **Number of cores** | 1 | 2 | 1 |
| **Speed (freq)** | 1 GHz | 1 GHz (each) | 720 MHz |
| **Memory** | 512 MB LPDDR RAM | 1 GB DDR2 RAM | 512 MB RAM |
| **Performance** | 2000+ Dhrystone MIPS | 4000 Dhrystone MIPS | 1200 Dhrystone MIPS |
| **Connectivity** | | | |
| **Ethernet** | 10/100 | 10/100 | N/A |
| **WIFI** | N/A | 802.11 b/g/n on board (based on WiLink 6.0) | 802.11 b/g on board (W2CBW003) |
| **Bluetooth** | N/A | v2.1 on board (based on WiLink 6.0) | v2.0 on board (W2CBW003) |
| **USB** | 4 USB 2.0<br>1 USB 2.0 OTG | 2 USB 2.0<br>1 USB 2.0 OTG | 1 USB mini-B (with Gallop43 expansion)<br>1 USB mini-AB OTG (with Gallop43 expansion) |
| **UART** | Yes[1] | Yes[1] | |
| **I2C** | Yes[1] | Up to 4[1] | 1 (with Gallop43 expansion) |
| **MMC2** | Yes[1] | Yes[1] | |
| **PWM** | Yes[1] | | 6 (with Gallop43 expansion) |
| **GPIO** | Yes[1] | Yes[1] | Unknown |
| **SPI** | Yes[1] | Yes[1] | 1 bus (with Gallop43 expansion) |
| **RS232** | 1 (250 Kbit/s TX max) | 1 | |
| **A/D** | Yes | | 6 input (with Gallop43 expansion) |
| **SD Card** | 1 micro-SD | 1 full size SD | 1 micro-SD |
| **Camera** | Yes (expansion header) | Yes (expansion header) | Yes (expansion header) |
| **LCD** | Yes (expansion header) | Yes (expansion header) | Yes (pins only with Gallop 43 expansion) |
| **DVI-D** | Yes | Through HDMI | |
| **HDMI** | Through DVI-D | Yes | |
| **Additional Features** | | | |
| **GPS** | No | No | Yes (with Gallop43 expansion) |
| **Open Source Community** | Hundreds of open-source projects<br><br>Huge community support | | |
| **Other** | | Graphics core supports OpenGL ES v2.0, OpenGL ES v1.1, OpenVG v1.1, and EGLv1.3<br><br>Stream 1080p video at 30 | |

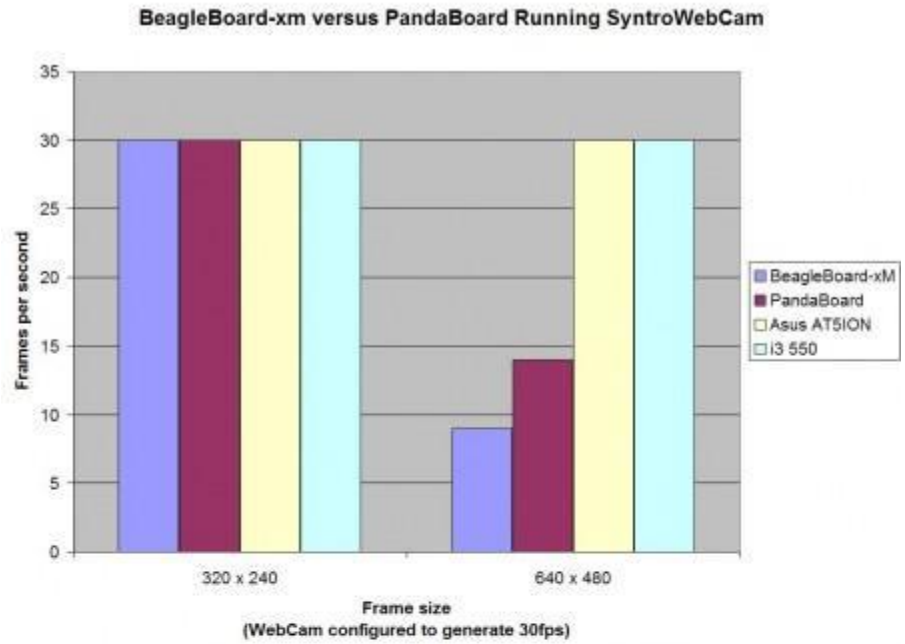
| | | | |
|---|---|---|---|
| **Power Consumption** | | | |
| | ~2 W at peak processing power | ~4 W peak with 100% CPU and WLAN | ~3 W (with Gallop43) |
| **Form Factor** | | | |
| | 3.25" x 3.25" | 4.5" x 4.0" | 0.67" x 2.28" Overo<br>4.64" x 2.64" Gallop43 |
| **Weight** | | | |
| | 1.3oz | 2.6oz | 1.47oz |
| **Price** | | | |
| | $149.99 | $174 | $219.00 (Overo)<br>$129.00 (Gallop43)<br>**$348.00 Total** |
| [1]The number of connections is dependent on what function the pin is mapped to | | | |

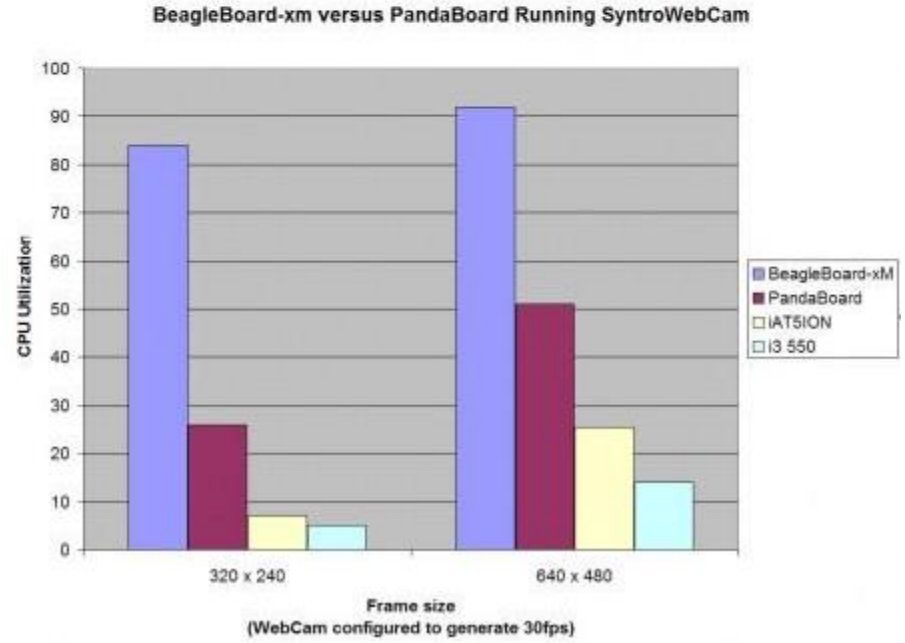
**Figure 1 - Comparison of Development Platform**

By looking at this chart, the Gumstix Fire does not offer all the features as the PandaBoard and Beagleboard offer and its processing power does not match the other two. Since video streaming does take a lot of processing power, it is an important issue to consider when selecting a board.

The Beagleboard and PandaBoard are very similar. Both offer roughly the same functionality and capabilities and are roughly the same price. However, the PandaBoard offers an on-board 802.11n module which is the exact 802.11 protocol selected for use in this project.

In addition since it has been mentioned multiple times that video streaming is a must, the PandaBoard is the clear winner for processing power. Below are two graphs that address this issue.[1] They show the fps and percent CPU utilization of two different resolutions, 320 x 240 and 640 x 480, using SyntroWebCam. SyntroWebCam is a tool used by Syntro, a generic robotic operating system. All components communicate via TCP/IP so these graphs provide valuable insight into how well the vehicle will be able to send video wirelessly from the PandaBoard.




According to these graphs, the PandaBoard will be able to meet the video processing functional requirement laid out in the Project Plan with ample CPU power to perform other tasks.

Figure 2 - Pandaboard

## Operating System

This section will focus on what operating system is going to be used on the Panda Board. The options that are in consideration are Linux, Android, and Windows CE. The operating system needs to be a widely used OS so that there will be information widely available for reference. The OS would also be preferable if it was free of charge with an open source mentality so there is direct access to the source code if need be. These two preferences will rule out Windows CE.

Since the operating system will be running on the Panda Board, there will be a constrained amount of processing power available to consume. This means that a light OS with only the minimal requirements would be preferred so that unneeded tasks are not consuming resources. Android is an OS that is directed at consumer electronics that run on batteries. At the core, Android is based off of Linux with a slightly modified kernel. The major downfall of using Android would be many extra tasks running in the background in turn wasting energy.

The remaining option would be a distribution of Linux. There is a pre-compiled version of a minimal version of Linux built specifically for use with the Panda Board. This version would not have any unnecessary tasks running in the background and would allow for full integration of all modules needed for the project. This would be much more efficient than the other options.

From the arguments above, the minimal version of Linux built exclusively for the PandaBoard will be the best option for the OS.

## Supplemental Microcontroller

Even though an Arduino microcontroller cannot perform the high-level processing as compared to the PandaBoard, it has two main features that are required for the project: PWM signal generation and reading and writing analog and digital signals.

To create a PWM signal, an interrupt within a microcontroller is triggered at a specified time defined by the microcontroller's timing counter. Current processes are suspended while the microcontroller creates a signal. The PandaBoard is capable of PWM signal generation but since it is dedicated for all high level programming that will utilize most of the CPU removing the overhead of suspending these processes for this simple function could harm overall performance of all functions.

The Arduino will be connected to the PandaBoard via SPI (Serial Peripheral Interface). This allows the PandaBoard to send commands and receive data from the Arduino. The rate of this data transfer will be determined in implementation.

The Arduino development board used could either be an Arduino Mega or an Arduino Due. The main difference between the boards is that the Mega is an 8-bit architecture while the Due has a 32-bit architecture, and the processing power available to the Due is much greater than that of the Mega.

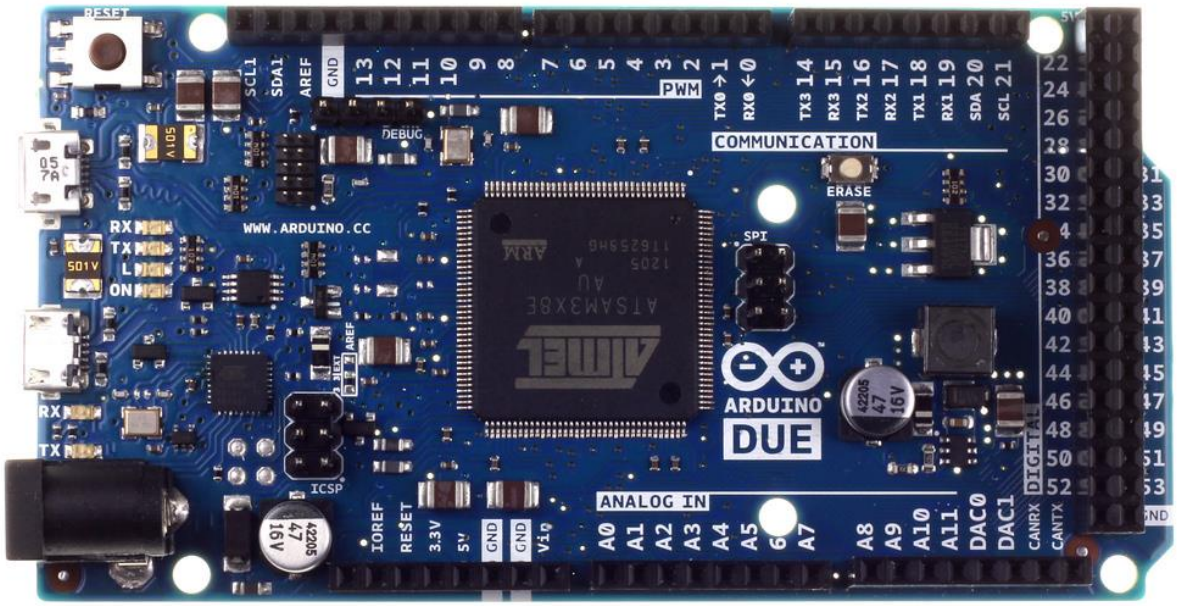



Figure 3 - Arduino Due

# RC Car

There are three main types of RC cars on the market: gasoline, nitro, and electric. Due to their high noise levels, gasoline and nitro will not be pursued for this project. Even though electric is the only type left, there are two variations of electric RC cars that need to be compared: brushed and brushless.

## Brushed DC

Brushed RC cars use electric motors with brushes in constant contact with a commutator to spin a permanent-magnet between two magnetic poles. These are very simple motors to design and operate by simply applying a positive voltage. However, their downfall comes to wasted power since the motor is always turned on, energy is lost due to the friction between the brushes and the commutator, and susceptible to mechanical wear between the brushes and the commutator.

## Brushless DC

Brushless RC cars, as the name implies, use brushless DC motors. A brushless DC motor uses permanent magnets to rotate around a fixed armature. Since a brush and commutator are no longer used to turn the motor, an electronic controller is instead used to continually switch the phase of the windings in the magnets to keep the motor turning.

Brushless offers several advantages over brushed motors including more torque per weight, more torque per watt (increased efficiency), increased reliability due to no brush and commutator erosion, reduced noise, reduction of electromagnetic interference, no windings on the rotor reduce centrifugal forces, and the entire motor can be enclosed to protect it from foreign matter.

## Type

There are many types of RC cars including but not limited to drift, rock crawler, on-road, off-road buggy, off-road monster truck, off-road truggy, off-road short course, and off-road ATV. For this project, off-road monster trucks will provide ample stability through tougher suspension, can handle more weight, and are able to drive across most outdoor terrain.

## Scale

RC cars come in multiple scales that include but not limited to 1:16, 1:12, 1:10, 1:8, and 1:6. These scales do not necessarily mean each RC car at given scale are the same size. A scale shows the size of the model compared to the actual item it is supposed to represent. Therefore, a 1:8 RC model of a tank will be much larger than a 1:8 RC model of a sports car.

For this project, there needs to be enough room for implementation and testing. A 1:16 RC car will not provide ample room for implementation of all items needed. A typical 1:16 RC monster truck will be

about 10” long by 8.5” wide. With the PandaBoard’s form factor, all peripherals, and all cabling needed will use up most of this area. In addition, all these items need to be located near the center of the RC car as much as possible to help maintain balance and reduce the effect to the RC car’s original center of gravity. Therefore 1:16 RC cars will be too small for the needs of the project and 1:10 scale with be pursued instead.

1:10 RC monster trucks generally have a length of 16” and a width of 11”. With this extra area, there should be ample room for implementation of all components while trying to preserve balance and center gravity.

### Conclusion

With all these different options to weight, the best route for this project is to use a 1:10 Brushless RC Monster Truck. Below is a picture of a model that meets all these requirements.

**Figure 4 - Exceed-RC Infinitive EP Electric Truck**

# Object Detection

The remote vehicle is required to have a functioning object detection unit onboard, to ensure the vehicle can safely travel to its future location. The three best options for doing this are stereo vision, IR, and ultrasonic sensors. This section will detail what the pros and cons are for each and provide justification for the type of sensor chosen for the vehicle.

Stereo vision is one sensor type that will not be pursued due to its advanced algorithms, need for a lot of processing power, and its need for two webcams. This will add additional costs and power usage to the project, both of which are constrained resources.

IR sensors use infrared light to measure how far away an object is. They can be very accurate but this accuracy is only limited to indoors. Using these sensors outdoors will make them obsolete due to infrared interference from the sun. Due to the functional requirement for the vehicle to be fully functional outdoors, IR sensors will not be pursued.

Ultrasonic (also known as sonar) sensors use sound waves to hear how far away an object is. These can be used very well indoors and outdoors but when used indoors, can suffer from echoes from walls that will cause undue interference.

The ultrasonic sensor will be selected for use on the vehicle, due to its superior ability to determine distance, both indoors and outdoors.

To help with the aid of obstacle detection, there is already another item on the RC car: the webcam. When an ultrasonic sensor detects an obstacle, the webcam will be able to see where the object is and make a decision as how best to avoid the object. An image analyzer takes an image and determines paths on the image that avoid objects. The only concerns about using this program are how much this program will drain CPU utilization and how it will affect real-time decision making.

# GPS Unit

When deciding what GPS unit to purchase, there are a number of areas you have to take in account. These are size, update rate, power, cost, number of channels, antenna, accuracy, and update rate. Below is a quick discussion of each of these areas.

- Power
  - The current average is around 30mA at 3.3V
  - Most of the power is used for analyzing data received from the satellites
- Cost
  - Vary greatly
- Number of Channels
  - Each channel represents a satellite there are a total of 24 GPS satellites and you will only see 12 at most at a time.
  - Therefore, anything above 12 Channels will be ample
- Antenna
  - Most common antennas are ceramic
  - Use power to amplify the GPS signal
  - Need to be pointed up
  - Need to be outdoors to use but can also work indoors. However, indoor operation is not guaranteed
- Accuracy
  - Usually accurate to +/- 10m
  - Dependent on module, time of day, clarity of reception, etc
- Update rate
  - Ranges between 1Hz to 20Hz
  - Higher the update rate, leads to better accuracy when traveling at high speeds

## Options

Below is a table that compares several GPS modules.

| | Venus GPS with SMA Connector | Copernicus DIP Module | EM-406A SiRF III Receiver w/ Antenna |
|---|---|---|---|
| **# of Channels** | 51 | 12 | 20 |
| **Max Update Rate** | 10 Hz | 1 Hz | ?? |
| **Accuracy** | < 2.5m | < 3.0m | +/- 5m |
| **Supply Voltage** | 2.7-3.3 V | 2.7-3.3 V | 4.5-6.5 V |
| **Price** | $49.95 | $74.95 | $59.95 |

**Figure 5 - Comparison of GPS Modules**

**Figure 6 - Venus GPS with SMA Connector**

## Conclusion

By looking at the table, the Venus GPS module has the best update rate, accuracy, and price of all three modules compared. Since the Venus GPS module does not come with an antenna, a ceramic antenna with a SMA connection will be purchased.

# Directional Unit

A magneto sensor, also known as a digital compass, is needed for this project so the RC car's direction can be determined from a standstill. The areas of a magneto sensor that need to be analyzed are update rate and accuracy.

## Options

Below is a table that compares several magneto modules.

| | HMC6352 | HMC5883L | MAG3110 |
|---|---|---|---|
| **Update Rate** | 1-20 Hz | 160 Hz max | 80 Hz max |
| **Accuracy** | 0.5 degrees | 1-2 degrees | ?? |
| **Supply Voltage** | 2.7-5.2 V | 2.16-3.6 V | 1.95-3.6 V |
| **Price** | $34.95 | $14.95 | $14.95 |

Figure 7 - Comparison of Magneto Sensors


Figure 8 - MHC6352

### Conclusion

Even though both the HMC5883L and MAG3110 modules are much cheaper and offer better update rates, the HMC6352 is more accurate, has internal calibration routines, and it does not require any processing of its data from a host processor. Therefore, the HMC6352 sensor will be purchased.

# Imaging Unit

To meet the requirement of having image recognition available to the command center, a camera will have to be purchased. The camera should be supported by Linux since that is the operating system that will be running on the Panda Board. The camera should also be supported by the Panda Board, and be able to meet the minimum video quality of 240p video at 15 fps minimum with 16 bit color. In the list of supported peripherals on the Panda Board's official website only one camera is listed as being supported. This camera meets our requirements of 240p and 15 fps. So for a guaranteed implementation process this is the camera that will be chosen for the project. The camera that will be used is the Logitech Webcam Pro 9000.

**Figure 9 - Logitech Webcam Pro 9000**

Specifications for the Logitech Webcam Pro 9000 are the following:

- Carl Zeiss® optics with autofocus
- Native 2-MP HD sensor
- High-definition video (up to 1600 X 1200*)
- 720p widescreen mode with recommended system
- Up to 8-megapixel photos (enhanced from native 2 MP sensor)
- Microphone with Logitech® RightSound™ technology
- Up to 30-frames-per-second video
- Hi-Speed USB 2.0 certified
- Logitech® webcam software (including Logitech® Video Effects™: fun filters, avatars, video masks, and face accessories)
- Logitech Vid™
- Universal clip fits notebooks, LCD or CRT monitors

# RC Car Hardware Implementation

Below is a diagram showing how all the components of the RC car and those to be added to the RC car will be connected. The types of connections are also stated in the diagram.

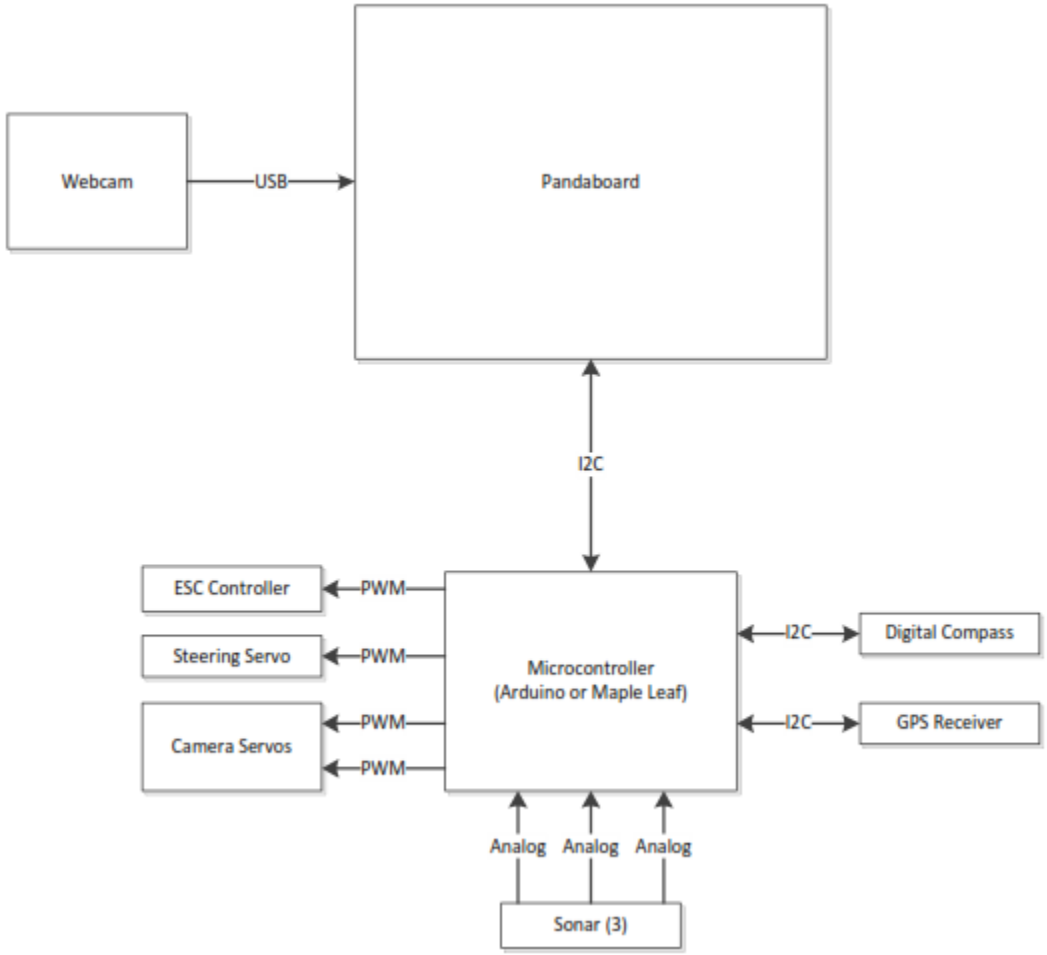



Figure 10 - System Diagram of RC Hardware

# Communication

## Zigbee

Zigbee is a new level of communication based on an IEEE 802 standard. It is best suited for applications needing to transfer data at low rates (maximum of 250kbps) and for low power consumption. Low power consumption would be ideal but the lack of data transfer will prevent the RC car to transmit any sort of

streaming video. Take into account that the maximum transfer rate of 250 kbps is an optimal transfer rate. The further data has to transfer, the lower the data transfer rate of Zigbee will be.

To help get a benchmark of how much bandwidth is needed for streaming video, look at the data provided at the *Streaming Learning Center*.[3] Derived from the information at this website a summary of some different video resolutions is provided below. These are compared at 24 fps and 15 fps and their corresponding bandwidth usage is also listed. According to research done by *Streaming Learning Center*, a reduction from 24 fps to 15 fps only reduces data rate by 20% instead of nearly 50% as many would think. This shows that running at a lower frame rate would not leave a linear relationship of available bandwidth through the connection. This will be considered when working on throttling the video stream.

| Different Video Transfer Rates | | |
|---|---|---|
| | **@ 24 fps** | **@ 15 fps** |
| **640x480** | 600-700 kbps | 480-560 kbps |
| **320x240** | 150-175 kbps | 120-140 kbps |

**Figure 11 - Different Video Transfer Rates**

## WIFI

WIFI is another option that has been looked into for the wireless communication. WIFI has a much larger bandwidth available than the Zigbee devices have and therefore is a much more viable option for the project. Also the IP protocol is natively supported in both Linux and in android, which will be the two operating systems that are used in the project. This factor would make implementation time much quicker and more reliable than creating a new software and firmware stack for the project.

| 802.11 network standards | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **802.11 protocol** | Release | Freq. (GHz) | Bandwidth (MHz) | Data rate per stream (Mbit/s) | Allowable MIMO streams | Modulation | Approximate indoor range | | Approximate outdoor range | |
| | | | | | | | (m) | (ft) | (m) | (ft) |
| **b** | Sep 1999 | 2.4 | 20 | 5.5, 11 | 1 | DSSS | 38 | 125 | 140 | 460 |
| **g** | Jun 2003 | 2.4 | 20 | 6, 9, 12, 18, 24, 36, 48, 54 | 1 | OFDM, DSSS | 38 | 125 | 140 | 460 |
| **n** | Oct 2009 | 2.4/5 | 20 | 7.2, 14.4, 21.7, 28.9, 43.3, 57.8, 65, 72.2 | 4 | OFDM | 70 | 230 | 250 | 820 |
| | | | 40 | 15, 30, 45, 60, 90, 120, 135, 150 | | | 70 | 230 | 250 | 820 |
| ***Table taken from*** [4] | | | | | | | | | | |

**Figure 12 - 802.11 Network Standards**

802.11b would not be used since it is an outdated protocol and since 802.11g and 802.11n offer better data

rates. In a perfect environment there would not be any interference in order to allow the theoretical data rate to become a reality. The 802.11g would provide ample range to meet the Project Plan's functional requirement of 70m range. However, since there is no way to know in advance what interference may be encountered, 802.11n would provide the greatest opportunity to meet the 70m operating range requirement.

Using WIFI the ability to route multiple cars' data through the network would increase the required bandwidth for each car, because each car would possibly rout other cars' data. This factor also demonstrates that 802.11n would be a more reliable option. The selected choice for the project will be WIFI using the 802.11n specifications and running on the 2.4GHz frequency band.

## Frequency Band

Since the project will be using 802.11n, there are two different frequency bands that can be chosen from. Either the 2.4GHz or the 5 GHz band would be available. However, the 2.4 GHz band has much less interference and is implemented in all Android devices with WIFI. Thus, to ensure compatibility and cut down on interference, the 2.4 GHz band would be the better option.

## Data Interchange Format

There were two data interchange formats considered for this project: XML and JSON. Both are standards and supported by all major platforms and programming languages. For this project, JSON was chosen over XML due to its smaller structure size. This means that it will take up less bandwidth to transfer across the network than XML, and will easily transfer all the information needed.

A sample JSON string for our project could look like this:

```
{ "car" : { "gps" : {
"latitude" : 42.028362 , "longitude" : -93.650959
} ,
"manual" : { "forward" : 0.47 , "right" : 0.02
} ,
"camera" : { "pan" : 0.00 , "pan_time" : 0.5 , "tilt" : 34.4 , "tilt_time" : 0.5
} ,
"sonar" : { "sonar_find" : 0
}
}
```

## Transmission and Response Rates

The cars and the command center need to be in constant communication with each other. To maintain "real-time" updates between the cars and the command center, we hope to achieve 10ms latency on average between phone and vehicle. This will require a send rate of 100 times per second. The car does not need such an instant response back to the command center, and will respond at a rate of 5 times per second.

### Conclusion

WIFI is the optimal solution for the wireless communication because it will provide the best bandwidth, be supported on all WIFI enabled netbooks, and be more readily available. This is the transport medium that will be used for the project.

# Command Center Design

## Platform

When choosing a platform to act as the command center node, the main considerations are networking capabilities and screen size. Network capabilities include which wireless protocols the device supports, and how the client implements them. Screen size is important due to the functional requirement to stream video and still images from the vehicles to the command center device.

IEEE 802.11n is an industry wireless standard that is an evolution of 802.11g, which is an evolution of 802.11b and 802.11a. All evolutions are backward compatible. Every standard uses the 2.4 GHz frequency spectrum, except for 802.11n, which also supports the 5 GHz spectrum. Chips that support both the 2.4 GHz and 5 GHz spectrum are labeled as being "dual-band".

### *Android*

Some Android manufactures have their devices listed as only supporting 802.11b/g, while others only list 802.11n networking chips in their models. Due to software limitations, no Android phone is natively able to support connectivity to an ad-hoc network.

A development called Wi-Fi Direct is a promising contender for native ad-hoc support and is implemented into the next iteration of Android, called Ice Cream Sandwich. However, this is untested and too unknown for consideration in this project.

- Pros
  - Some phones may support ad-hoc networking
  - Mobile
  - Easy to develop for
  - Larger selection of devices
  - 4 hardware buttons
- Cons
  - Most/all phones will not support ad-hoc networking without rooting

### *iPhone*

The iPhone 5 has hardware support for 802.11b/g/n. Like the Android operating system, the iPhone's

operating system does not allow for iPhone to iPhone connections or an ad-hoc network to be established. In both Android and iPhone, Bluetooth can be used to create a direct connection between devices; however, it only has a max range of 100m and a throughput of 24 Mbps, which is not acceptable for the project.

- Pros
  - Mobile
  - Low variability between versions
- Cons
  - Will not support ad-hoc networking
  - Mac OS is needed for development
  - $99 fee for developer account
  - No options for devices except for amount of memory
  - Only one hardware button

### *Netbook*

Most recent netbooks support the 802.11b/g/n protocols. Whether the netbook supports dual-band 802.11n communication depends solely on the wireless NIC card inside the netbook, but there are many on the market that support it. Furthermore, there are no software limitations on the netbook, which means that ad-hoc networks can be created between many netbooks.

- Pros
  - Supports ad-hoc networking
  - Better WIFI capabilities (range, frequency)
  - Wider range of programming languages
- Cons
  - Not as mobile
  - Higher power consumption

## Comparison

When choosing what device to use as our command center, we chose the best one that fit into our previous specifications and allowed for the most options going forward. WiFi is the wireless communications standard for our project; more specifically, IEEE 802.11n radios will be used for their superior bandwidth and range. All the choices have options available with 802.11n adapters in them.

Since one of the project's highest priorities is to connect to a mesh ad-hoc network, we researched the ad-hoc capabilities of each device. Of the three, only the netbook is able to create an ad-hoc connection to an identical device. An Android phone cannot connect over 802.11n to another Android phone directly, nor can iPhone to iPhone. For this reason, the netbook is a better solution for integration into an ad-hoc network.

| Specification | Android | iPhone 4S | Netbook | *Winner* |
|---|---|---|---|---|
| **802.11n** | Most phones, and most of those only 2.4GHz band | Only 2.4GHz | Some netbooks. | *Netbook (for having more adapter choices than Android)* |
| **Ad Hoc to identical device** | No | Over bluetooth | Yes | *Netbook* |

**Figure 13 - Comparison of Mobile Platforms**

## Conclusion

We will be using a netbook for the command center. Having better WIFI capabilities and ad-hoc support is the number one priority. Developing in Java on a netbook will enable development in many programming languages. This gives easy portability on to PC, Mac, or Linux because Java is run on a virtual machine compatible across platforms. Java also offers easy multi- threading and networking interface. Another reason for selecting a netbook is the keyboard and mouse. These extra inputs options will allow the UI to free up valuable screen space.

## Netbook Selection

The netbook selected for this project is an Acer Aspire One. The netbook runs Ubuntu Linux and will give us the greatest flexibility. Developing in Java on a Linux netbook will also allow us to more easily port the command center over to an android tablet to meet our lowest priority.

## Architecture

The software in this project will have to tackle several issues. One of the biggest will be communication over the network. We will have to send and receive messages from other cars and the command center. The cars will have to send sensor updates on demand and the command center will have to receive this data.

## Communication

The command center will use a monitoring architecture that will listen to a socket for messages. The mission commands will be sent using TCP. We chose TCP for messages because they have guaranteed delivery. It is critical that the missions are delivered. All of our sending and receiving of mission and sensor data will be done through a communication module. All of the communication methods will be housed here. This will eliminate the risk of communication changes affecting the rest of the source code.

Any sensor that is added to the car will implement the Arduino interface. This will allow for expanded functionality in the future. Any sensor will be able to be added and accessed from the device. Having this requirement allows for a low cost in adding more features.

## Command Center Interface

The command center is comprised of two major screens. The first is the Mission screen that has a Map interface that shows the route of all robots, sensor values for each robot, and a button that brings up a

screen to select mission priority levels. The second is a screen that shows only information for a specific car. This information includes current sensor values, Map of current location, and a button to turn on the video feed. At the top of the screen there is always a button for each car and a button to return to the mission screen available on every screen.

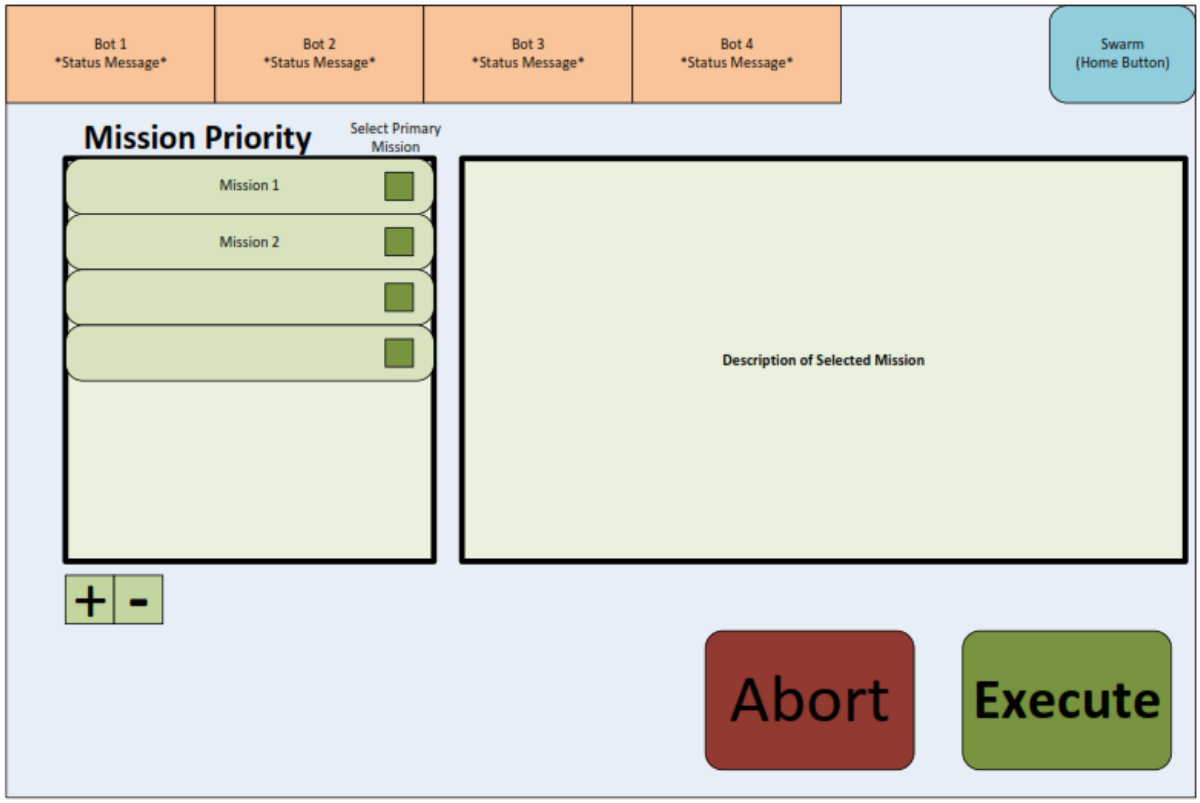



Figure 14 - Mission Priority Screen

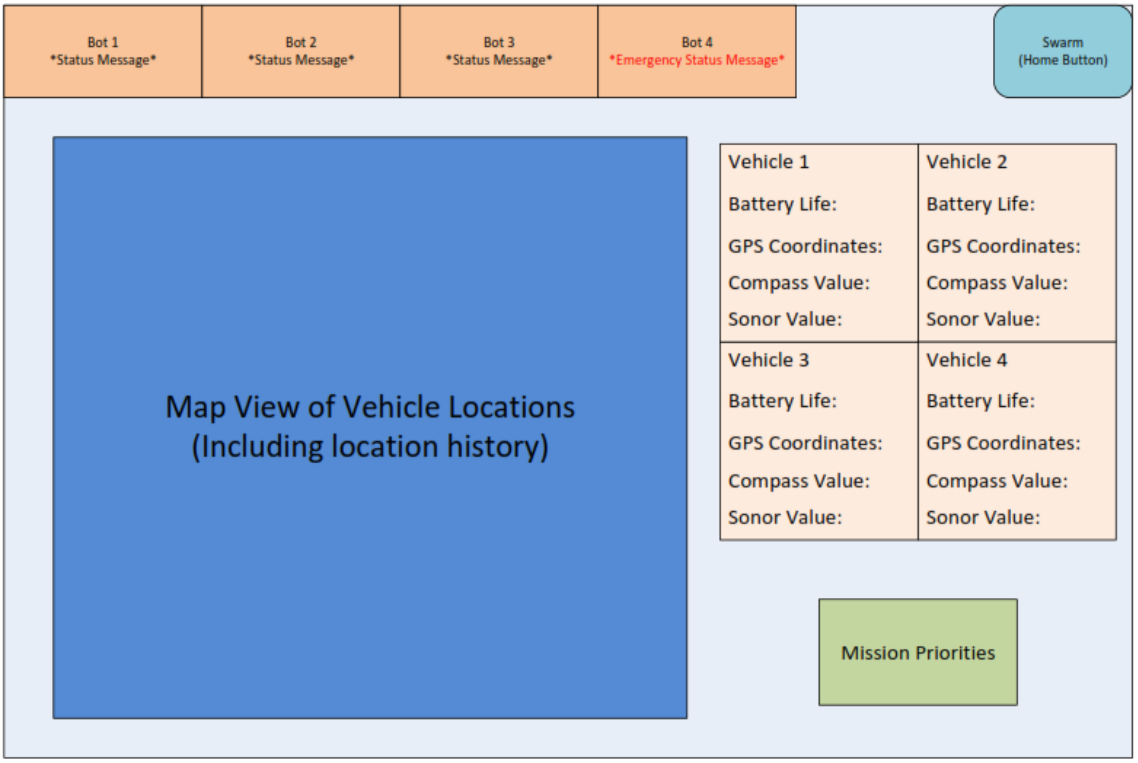



Figure 15 - Swarm Screen

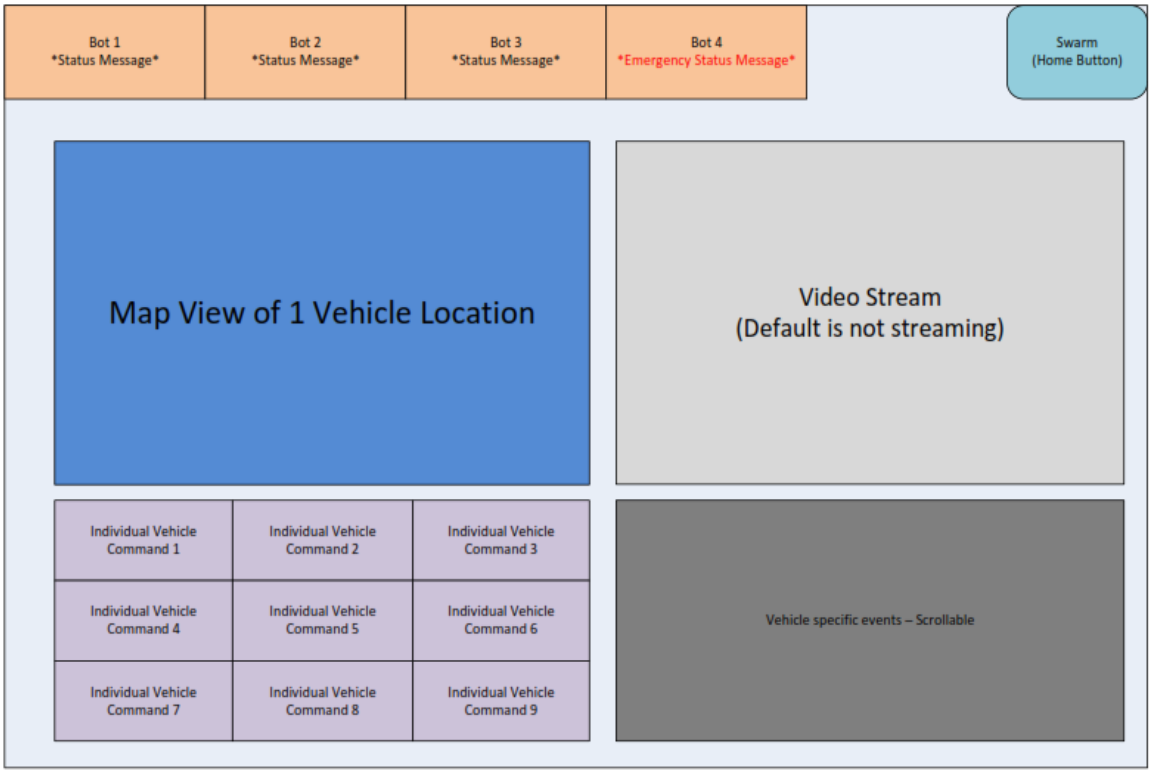



Figure 16 - Vehicle Screen

# Overall System Integration

Below is a diagram summarizing how the entire system, including the command center, the robots, and the components of the robots function.

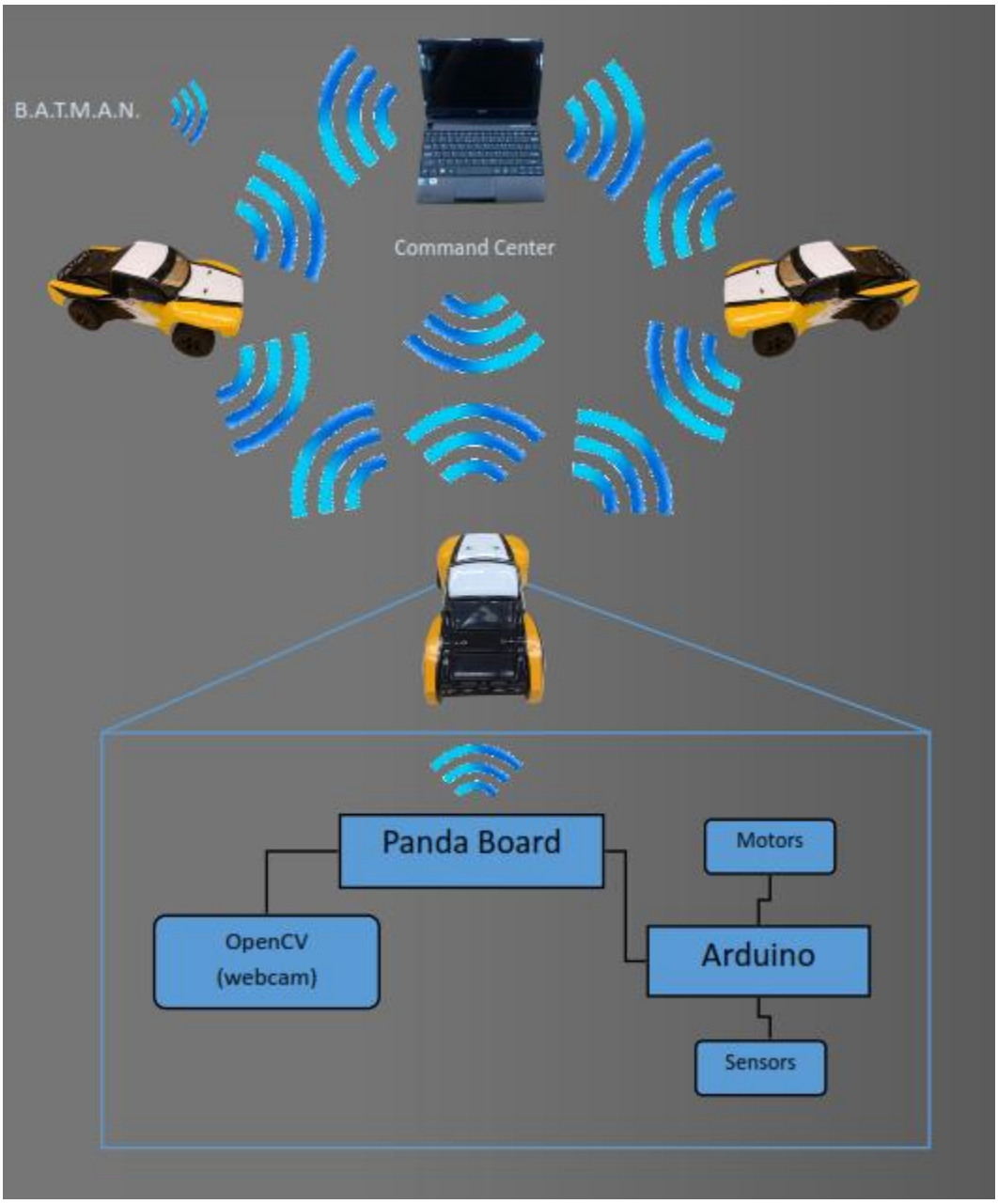



Figure 17 - System Design

# Arduino

The Arduino Due was chosen over the Mega because of the increase in processing power. At the time of initial implementation, the Due had been on the market for slightly over two months. This became an issue as the difference in architecture between the two microcontrollers prevent the third-party libraries, that are the reason the Arduino is more useful, cannot be ported directly to the more complex architecture of the Due without significant rewriting. Therefore, finding libraries that not only performed the required functions but also functioned on the new microcontroller was difficult, and will be discussed later in the sections directly relating to those components.

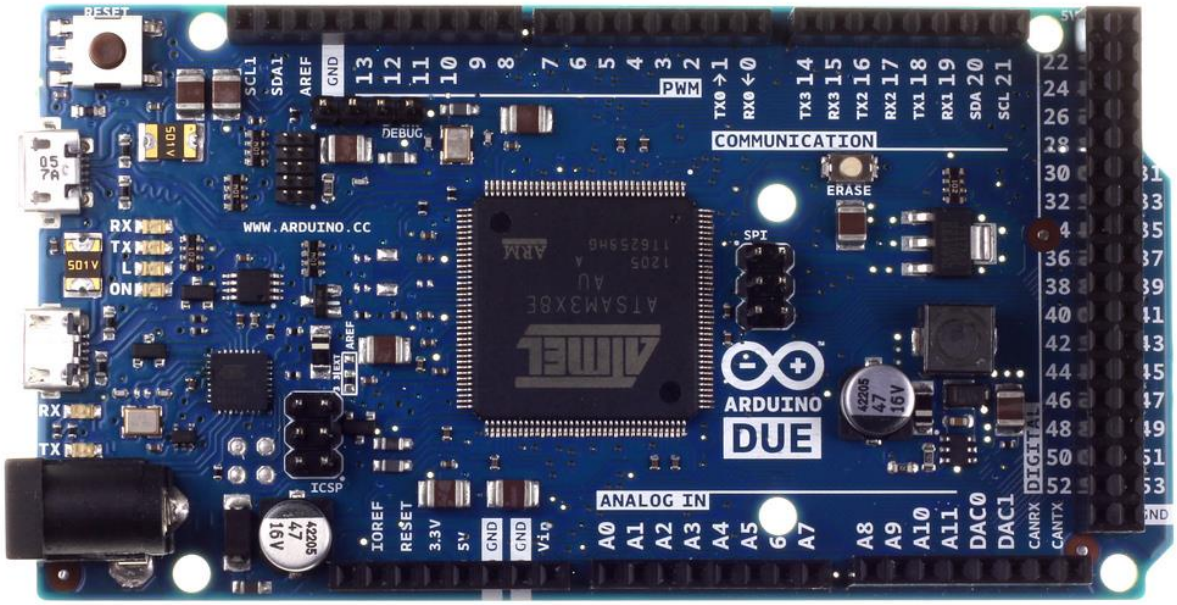



Figure 18 - Arduino Due

# Sensors

There are three sensors added to the RC car to supply the necessary functionality to meet the functional requirements. These sensors are a GPS module, digital compass, and sonar sensor.

## GPS

As was discussed before, the GPS module communicates with the Arduino via serial communication. Communication between the GPS module and the microcontroller is still an issue because none of the available libraries were able to access the updated GPS module that we were forced to select (due to the unavailability of the module used by the previous team). Limited success was achieved by accessing the module directly and processing the raw data. Below is a simple diagram of which pins of the GPS module

are connected to the pins on the Arduino Due.

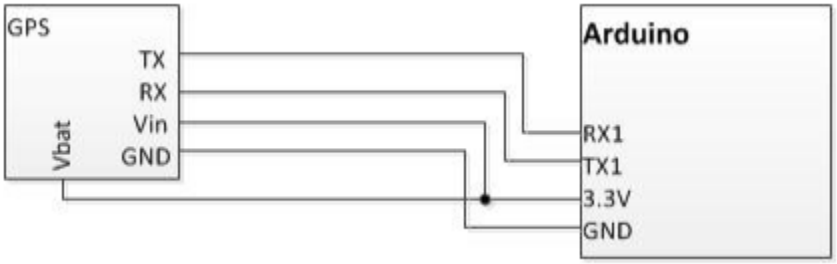



Figure 19 - Connection of GPS Sensor

## Digital Compass

The compass communicates with the Arduino Due via I2C communication. Below is the specification of how to use the digital compass (HMC6352) as given from Honeywell:

> *The HMC6352 uses a layered protocol with the interface protocol defined by the I2C bus specification, and the lower command protocol defined by Honeywell. The data rate is the standard-mode 100kbps rate as defined in the I2C Bus Specification 2.1. The bus bit format is an 8-bit Data/Address send and a 1-bit acknowledge bit. The format of the data bytes (payload) shall be case sensitive ASCII characters or binary data to the HMC6352 slave, and binary data returned. Negative binary values will be in two's complement form. The default (factory) HMC6352 7-bit slave address is 42(hex) for write operations, or 43(hex) for read operations.*

Reliable communication was achieved with the compass module, but accuracy is still an issue during operation. The module functions perfectly when the Drive systems are disengaged, but during operation the magnetic fields generated by the motor, servos, and wiring causes the readings to fluctuate as much as twenty degrees. It might be possible to reduce the fluctuations to an acceptable amount by shielding both the module and wiring. Below is a simple diagram of which pins of the digital compass are connected to the pins on the Arduino Due.

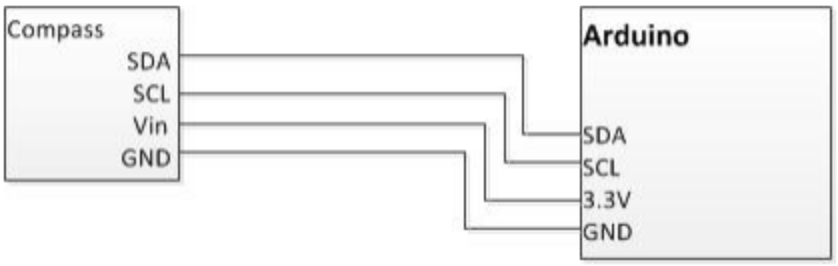



Figure 20 - Connection of Digital Compass

## Sonar Sensor

The sonar sensor can give back both PWM and analog signals to indicate the distance to the largest object in cm. When testing on the Arduino Due, the PWM signals were unable to give back a correct signal so the analog signal was used instead. In the end this was probably better for the speed of computation on the

Arduino Due because there will be no interrupts to interpret the PWM signal given by the sonar sensor. Power is supplied to the sonar sensor through the 5V switching voltage regulator to reduce the amount of current being drawn through the USB between the Arduino and Pandaboard. Below is a simple diagram of which pins on the sonar sensor are connected to the pins on the Arduino Due.

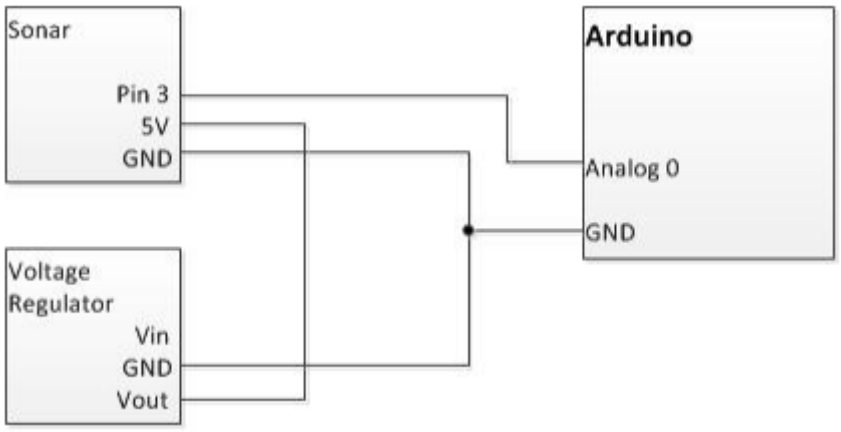



**Figure 21 - Connection of Sonar Sensor**

## Servo Control

Both the camera servos and steering servo located on the RC car are controlled via PWM signals. Power is supplied to the servos through the 5V switching voltage regulator to reduce the amount of current being drawn through the USB between the Arduino and Pandaboard. Below is a simple diagram of which pins on the servos are connected to pins on the Arduino Due.

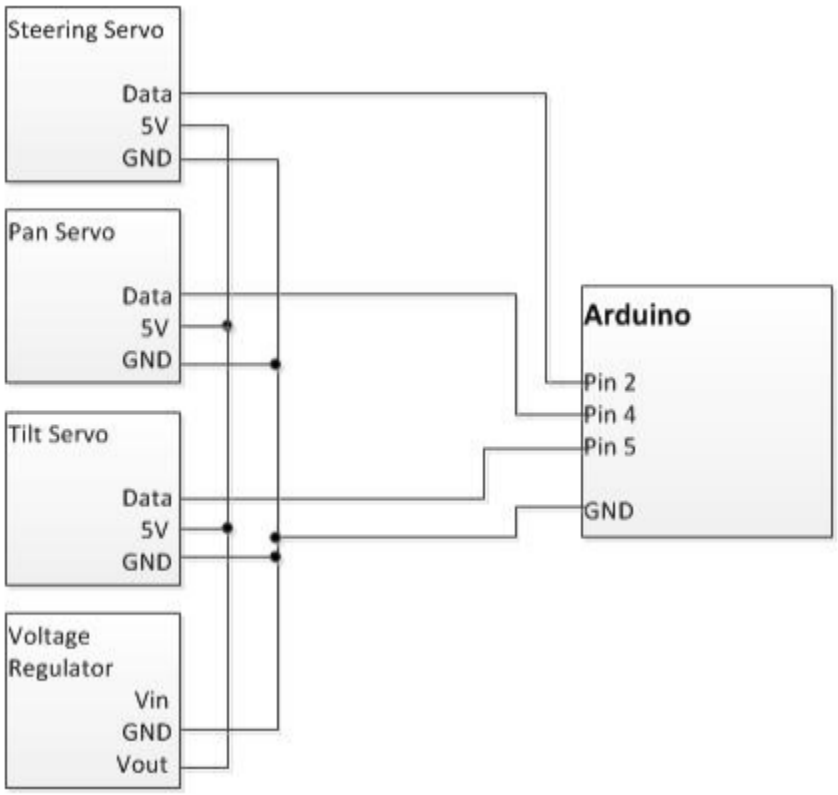



Figure 22 - Connection of Servos

## ESC Control

The ESC to control the motor on the RC car also receives a PWM signal. Note that only the PWM data signal and ground wire are supplied to the ESC because the ESC supplies its own power. If the 5V wire is connected to the Arduino Due 5V pin, the Arduino Due will actually be powered through the ESC. This can easily damage the Arduino Due because reverse current will be supplied on an output voltage pin. Below is a simple diagram of which pins on the ESC are connected to the pins on the Arduino Due.

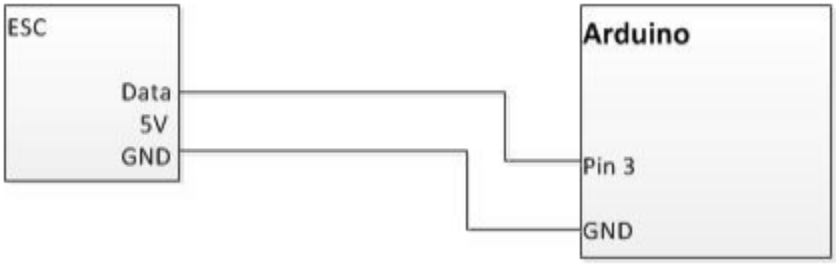



Figure 23 - Connection of ESC

## Custom Shield

When designing the shield, the location of the Arduino Due on the RC car needed to be determined first. Afterwards, placement of all connections and hardware could be determined. The switching voltage regulator, GPS sensor, digital compass, power connectors, servo connectors, and sonar connector are all

located on the shield. The digital compass was placed on the shield so that the orientation for north was facing towards the front of the RC car.

The shield was designed with Eagle, which has a freeware version for certain sized PCB boards. The main layout of the board was taken from the Arduino Due footprint so that the header pins would match up correctly. After the base layout was captured, the board could then be properly laid out. All of the 3 pin connections for the servos were placed toward the front side of the board because this was closer to the moving servos on the camera turret mount.

The voltage regulator was placed towards the back, as close to the screw terminals for the batteries as it could go. This was to keep the trace resistance lower for the higher voltage and current of the batteries path. The Pandaboard DC barrel jack was also placed nearby the voltage regulator for the same reasons. Also notice the thicker traces and extra vias that are on the 5V traces for the servos. The GPS and compass integrated circuits are powered off a 3.3V path, which is brought up from the Arduino. After all of the signal paths and voltage paths were laid out, a ground port was added to both sides of the board to allow for a less resistant path to ground.

Below is the layout of the board.

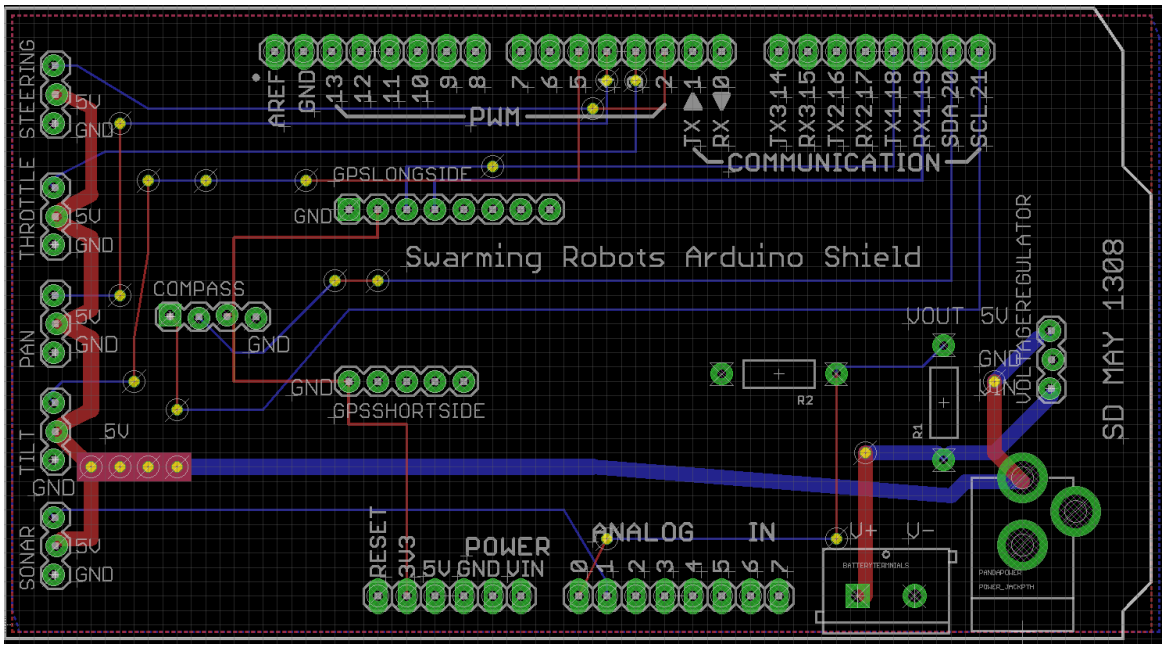



Figure 24 - Layout of the custom Arduino shield

# Pandaboard

The Pandaboard is the main processing unit of each robot. It runs a version of Ubuntu Server specially compiled for the Pandaboard. Pandaboard is responsible for calling the Arduino, running the main logic of

the project, the OpenCV image detection logic, the B.A.T.M.A.N. routing protocol and networking, as well as the search algorithm.

Figure 25 - Pandaboard ES

# OpenCV

In order for the swarm to determine where its objectives are located, color detection using OpenCV was implemented. The color detection function is on the Pandaboard and can be called directly from the mission logic code.

The OpenCV function queries an image from the webcam connected to the Pandaboard and uses a threshold to convert the image gathered from the webcam to a binary image. Thresholding looks at each pixel in the image and sets it to 0(black) if that pixel is outside a specified range or 1(white) if the pixel is inside the specified range. After a thresholded (binary) image has been created, the total number of white pixels is counted. To ensure that any noise was eliminated another weight was set, so small amounts of white would be ignored and only large enough white would be classified as an objective. Built in moment functions are also used to gather where the found object is in the image in the horizontal direction. The function puts the information of whether the bot found an object and where that object is on the camera in a structure and outputs a pointer to that structure.

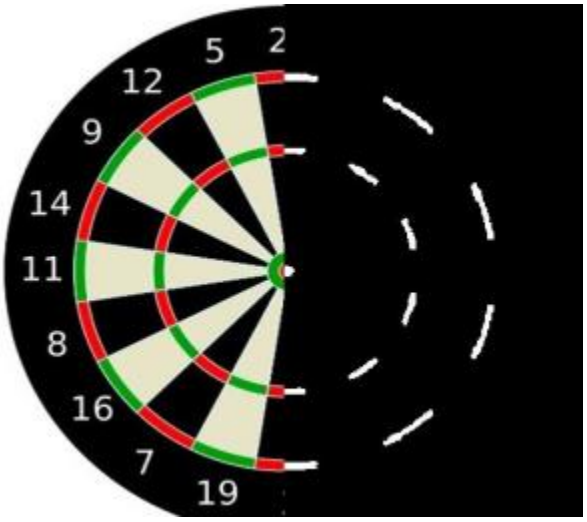



Figure 26 - Thresholding with red

# B.A.T.M.A.N.

In order for the robots to communicate amongst themselves and with the command center, we needed to create an ad-hoc network. Previous iterations of this project implemented the ad-hoc at the application layer. This approach was the simplest to implement, however it requires the most overhead. To improve the ad-hoc network we chose to create the networkwith a routing protocol. Our original design decision was to use the AODV-UU routing protocol, but we found it was not well document or updated. Through our research, we found "Better Approach to Mobile Ad-Hoc Networking" a much better alternative. B.A.T.M.A.N. is an open source kernel module for Linux that is now already baked into the Ubuntu kernel. Through the use of B.A.T.M.A.N. we were able to include some more advanced features into our network such as bridge loop avoidance, distributed ARP table, and automatic network updating. Implementing the ad-hoc network as a routing protocol allowed us to implement upper layers such as TCP sockets exactly the same as a traditional network.

Figure 27 - B.A.T.M.A.N. logo

# Search Algorithm

For the robots to efficiently search a specified area they must have a plan. The algorithm developed to coordinate the robots movements resembles a rake pattern.

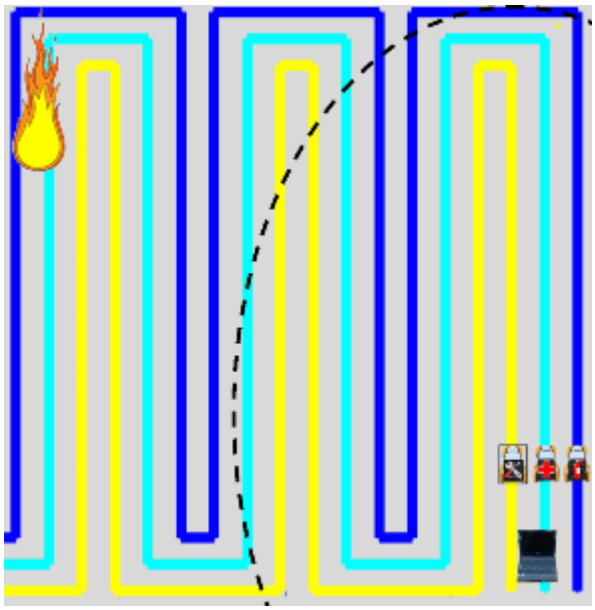


Figure 28 - Search algorithm rake

It takes in four GPS coordinates, which make up a rectangle, and then divides the area into a grid pattern based on factors involving the robots vision. Now it develops n number of paths, n being the number of robots, and assigns each grid point to one of the paths.

# Command Center

The command center is the user interface point for the swarm. The command center runs on a Linux netbook. We had two netbooks at our disposal. We decided to install Ubuntu on one to match the Pandaboards. We decided to install Fedora on the other netbook, because feature team teams may decide to move some of the command center's functionality to a server. Red Hat is the most common Linux server operating system and Fedora is the closest we can get to Red Hat without buying a support account.

# RC Car

## Motor

The stock motor that came with the RC car is rated for a top speed of up to 40 mph when tuned properly. To help increase controllability of the RC car, the ESC was programmed for its lowest speed setting. However, this was not enough to decrease the extreme top speed the motor was capable of. The only option was to decrease the number of cells in the battery effectively limiting the output voltage of the battery, resulting in much lower speeds.

## Mounting Hardware

The ultimate outcome from mounting all the additional hardware onto the RC car was to still use the original cover. This was done to have the RC car resemble its original look. Here is a list of design implementations taken to mount all the hardware:

- The Pandaboard, Arduino, pan/tilt servos, GPS antenna, sonar sensor, and on/off switch are all mounted on a 1/8 inch sheet of acrylic. This acrylic is mounted onto the shock towers.
    - The Pandaboard is mounted near the middle of the RC car. This is done due to the USB and Ethernet stack connection on the Pandaboard. The SD card slot will be facing towards the front of the car.
    - The Arduino Due had to be mounted near the back of the car to accommodate the USB connections on the Pandaboard.
    - The sonar sensor is mounted near the very front of the RC car.
    - The pan/tilt servos for the webcam are mounted between the posts of the front shock towers.

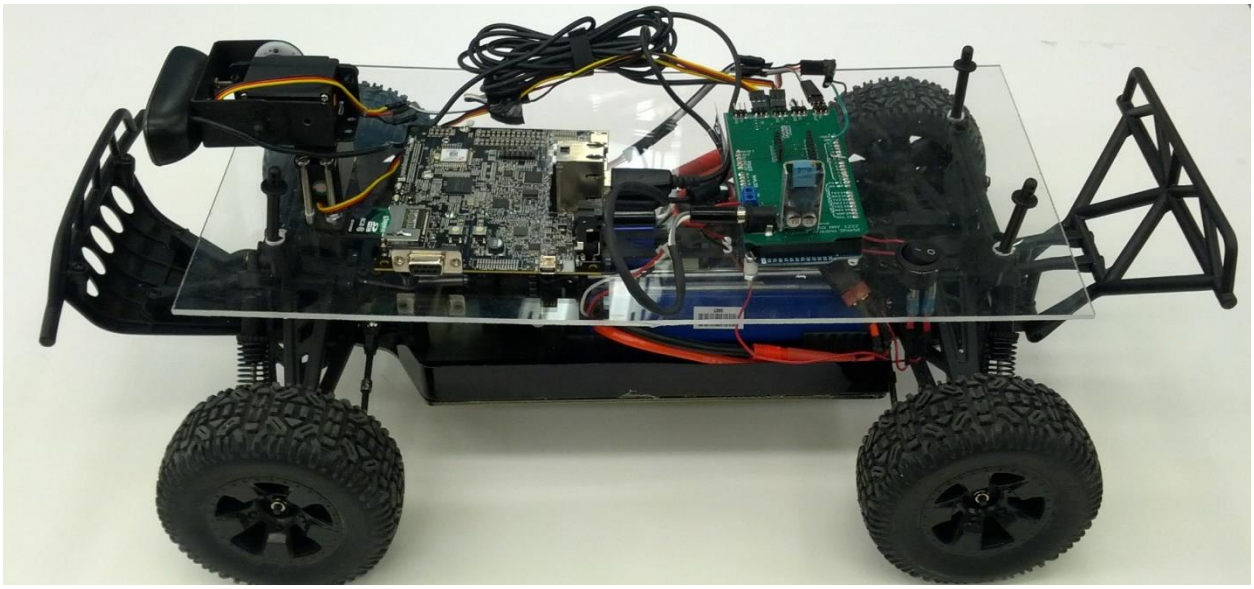


Figure 29 - Final car without cover

Figure 30 - Finished Robot

## Batteries

To meet the functional requirement of a minimum operation time of 30 minutes, one 7.4 V 5000 mAh LiPo battery was used to power each car. This is more than enough battery life to have each car operate for longer than 30 minutes.


Figure 31 - Zippy LiPo 7.4V 5000mAh

# Final Budget Breakdown

We set a budget of $3,000. We were able to finish the project under budget. The breakdown can be seen below.

| Item | Unit Cost | Quantity |
|---|---|---|
| Pandaboard | 179 | 3 |
| Arduino Due | 49.99 | 4 |
| Sonor | 39.95 | 3 |
| GPS | 49.95 | 3 |
| GPS Antenna | 18.8 | 3 |
| Compass | 34.95 | 3 |
| WebCam | 31.28 | 3 |
| Servo | 9.99 | 6 |
| Truck | 144.99 | 3 |
| Battery | 20.93 | 4 |
| SD Card | 7.99 | 4 |
| Custom PCB | 50 | 4 |
| Servo Turret | 10 | 3 |
| **Total** | 2102.34 | |

Figure 32 - Budget breakdown

# Testing

We decided to take a cumulative approach to our testing. We built and tested the individual components, and then started combining them together. After adding a component we verified its function as part of the system.

## Pandaboard & B.A.T.M.A.N.

After installing Ubuntu on the Pandaboard, we ran into many issues with the TI drivers for critical components such as the Wi-Fi card and power management. We found the default drivers compiled into Ubuntu worked better than the TI specific ones. After resolving the driver issues, we were able to install and test B.A.T.M.A.N. between a Pandaboard and two netbooks.

## OpenCV

Compiling the OpenCV program was an issue because after talking with another team it was determined that the build of OpenCV we installed did not work. The other team that we were in contact with gave us an install script that we were able to use. With the working version of OpenCV we were able to write and test OpenCV programs on a laptop with the same webcam that the robots are using. From the laptop we were able to test different ranges of color thresholds and found the value of the center of the webcam images, 319.5. The colors we mainly tested were yellow, red, and blue.

System So Far: Once the OpenCV program was tested individually on a laptop, we installed OpenCV on the Pandaboard and tested the OpenCV program while on the Pandaboard. This was achieved by viewing the thresholded image from the OpenCV program through a SSH session.

## Arduino

The testing of the Arduino platform was incorporated with every feature addition, by performing the test in a standalone implementation, and then testing the functionality after it was integrated with the main library created to simplify the interaction between the Arduino and the hardware (servos, compass, etc.). The main issue found during testing the Arduino was the inability to communicate with the GPS module. Many libraries were used, as well as the code designed by the previous team, to no avail.

System So Far: Communication between the Arduino Due and Pandaboard consists of a serial connection using a USB cable. Initial tests entailed the transmission of characters, followed by pertinent information. For example, in a preliminary build, sending the string "d-50" would have turned the wheels fifty percent to the left of center.

## Command Center

At this point we were ready to write code on the command center to control the robot as a whole. We wrote smaller programs to demo important sections of our project.

## Functional Requirements

| Requirement | How the requirement was met |
|---|---|
| High 1 | This requirement was met. The network remained connected via Wi-Fi. |
| 2 | This requirement was not met, because the GPS sensor didn't function properly. |
| 3 | This requirement was met. BATMAN was configured with a distributed ARP table. |
| 4 | This requirement was met. BATMAN was used to implement the ad-hoc network. |
| 5 | This requirement was met. BATMAN was configured with a distributed ARP table. |
| 6 | This requirement was met. Nodes in the network are able to work together. |
| 7 | This requirement was met. The searching algorithm was written and simulated. |
| 8 | This requirement was met. The sonar sensor was able to detect an obstacle the size of the diameter of the wheels from over 2 meters away. |
| 9 | This requirement was not met, because the GPS sensor didn't function properly. |
| 10 | The camera servo is not able to rotate towards the ground up to 45 degrees due to hardware limitations. Horizontal +/- 180 degrees and vertical +45 degrees were all met. |
| 11 | This requirement was met. In our demo the user picks a mission from a predefined list. |
| 12 | This requirement was not met, because the command GUI couldn't be fully implemented with the GPS sensors working correctly. |

| | |
|---|---|
| 13 | This requirement was met. We have 4 vehicles. |
| 14 | This requirement was met. Each vehicle has the same set of sensors. |
| 15 | This requirement was not met, because a multi-robot system couldn't be tested without functioning GPS sensors. |
| Medium 1 | This requirement was met. BATMAN was configured with a distributed ARP table. |
| 2 | This requirement was met. BATMAN was configured with a distributed ARP table. |
| 3 | This requirement was met. The Arduino library is used by the Pandaboard to call any sensor. |
| 4 | This requirement was met. BATMAN keeps track of known peers at less than a 1 second interval. |
| 5 | This requirement was met. Using 802.11n we far exceed 70 meters. |
| 6 | This requirement was not met, because the GPS sensor didn't function properly. |
| 7 | This requirement was not met, because video streaming was no longer used. |
| 8 | This requirement was met. The servo can rotate 180 degrees in 1 second. |
| 9 | This requirement was met. Reducing the battery voltage allowed for greater speed control. |
| 10 | This requirement was not met, because the GPS sensor didn't function properly. |
| 11 | This requirement was not met, because a multi-robot system couldn't be tested without functioning GPS sensors. |

# Future Implementation

## Search Algorithm

The current search algorithm is limited to searching rectangular areas. Many times the area to be searched would not be a rectangular shape so developing an algorithm to search irregular shaped areas would be needed. As always searching in the most efficient manner is preferred so improving the efficiency of all search algorithms should be done.

## Heterogeneous Swarm

To avoid geographical obstacles the integration of multiple vehicle types such as aerial and aquatic vehicles is a must.

## Object Recognition

Identifying mission objects through more advanced object recognition techniques would greatly expand the possibilities for the project. This would also more closely represent the real world applications that this project could be used for.

## Distributed Computing

Currently each robot requires relatively large, powerful, and expensive components. If the system could be distributed effectively, different hardware could be used to create smaller, cheaper, and more power

efficient robots.

### Operation Manual

Please refer to this document on our website for details of how to use the project.

# Standards

## Software

On both the Pandaboard and Arduino, C was chosen as the best programming language to use. C11 is the most recent standard revision of the C language, and was used to create the server application on the Pandaboard. Because C11 was used, it was possible to integrate code that captured a video stream from the webcam connected to the Pandaboard. This task would have been made more difficult if a standard, supported programming language was not used. The Arduino platform has its own standard variation of C, which was used on the Arduino board.

The initial communication design used JSON to format data between the Pandaboard and the command center application. The use of the JSON standard allowed the use of free libraries for both platforms to parse the JSON string. Writing a custom JSON parser to the level of the libraries initially used would be more work than the project required. The JSON standard was replaced with the lighter-weight CSV standard.

To actually transmit the CSV string between the Pandaboard and command center would have been a huge task, if not for standard sockets programming over a WiFi connection. C11 has the ability to create, bind, and read/write to a TCP socket. TCP is another standard itself for ensuring delivery of data over a socket. To connect the sockets, the WiFi standard IEEE 802.11n was used. Without that standard in place, the wireless communication between Pandaboard and command center could have been incompatible. The use of WiFi, TCP, and sockets programming allowed easy to set up and reliable communication between the pandaboard and the command center, a task that would have been immensely more difficult without the use of those standards.

## Hardware

The Arduino is powered by its USB connection to the Pandaboard. This requires the Pandaboard and Arduino to meet the USB standards for power. The Arduino and Pandaboard also communicate over the USB connection, which requires both boards to meet communication standards over USB as well. Furthermore, all the auxiliary devices connected to the Arduino had to be connected over a standard connection to ensure connectivity.

The compass IC selected for use in the project, the Honeywell HMC6352, uses the Inter-Integrated Circuit (I2C) standards for its communication channel. Due to this the Arduino also has to use the I2C standards to communicate with the compass module. This standard had to be used because the HMC6352 was one of the few available compass sensors that met the minimum requirements of the project while remaining inside the boundaries of the budget.

The I2C standard uses two bidirectional communication lines between each device. Each device is then connected onto an I2C network. The I2C standards allow for 7 or 10 bit device address space. The most common data rates for an I2C channel are either 10kbps (low speed) or 100kbps (standard speed). The I2C channel is usually defined to run at 3.3V or 5V depending on what the hardware requirements are.

The HMC6352 compass module uses the 100kbps standard speed for its data rate. The compass module is also powered by 3.3V, and thus is using the 3.3V standard for the line level. Also the device is using a 7 bit device address space.

The GPS IC selected for the project, the SkyTraq Venus 634LPx, uses the UART standards for its communication. Because the Venus uses a UART interface, the Arduino must also support using a UART standard. The baud rate of the IC is configurable to 4800, 9600, 19200, 38400, 57600, and 115200. This standard was used because the Venus was a GPS that was capable of meeting the minimum requirements while maintaining a low budget.

The UART standard is an asynchronous communication standard that is very common in computing. It works by disassembling data from bytes into bits, then transmitting the bits at the specified baud rate, and reassembling the bits into bytes. This process is usually accomplished by using a shift register. Each device must be set to the same baud rate for proper communication between the devices.

The SkyTraq Venus is configured to support the NEMA data standard on boot. This standard is sent over the UART connection of the device. A NEMA data stream sends a sentence code at the start of each message, and the required data for the corresponding sentence code.

The SkyTraq also uses its own binary command set for accepting commands for its master. This command set is specific to SkyTraq devices. This standard uses two header bytes to indicate the start of a message. It then uses two bytes to declare the length of the following message. It then appends the data packet which can range from 1 to 65535 bytes. After this a checksum byte and the two following end of transmission bytes are sent. This standard was used because it was the only standard for configuring the Venus GPS.

# Reference Documents

[1] http://tirokartblog.wordpress.com/2011/03/26/beagleboard-xm-v-PandaBoard-running-syntrowebcam/

[2] http://en.wikipedia.org/wiki/Brushless_DC_electric_motor

[3] http://www.streaminglearningcenter.com/articles/streaming-101-the-basics---codecs-bandwidth-data- rate-and-resolution.html

[4] http://en.wikipedia.org/wiki/IEEE_802.11

http://en.wikipedia.org/wiki/Bluetooth#Uses

http://www.iphonedevsdk.com/forum/iphone-sdk-development/7282-ad-hoc-wifi-connection-between- two-

iphones.html

http://www.apple.com/iphone/specs.html

http://www.howardforums.com/showthread.php/1716949-Android-Phones-with-5Ghz-%28Dual- Band%29-Wifi-chips

http://en.wikipedia.org/wiki/Comparison_of_Android_devices