CSC 460 Project 3

Real Time Systems Final Project

Team

Curtis St. Pierre
Mark Roller
Geoffrey Gollmer
Adam Anderson
Te-yu (Allen) Chang
Mikko Sanchez
Fraser Delile
Logan Bissonnette

Group

Geoffrey Gollmer - V00730142
Adam Anderson - V00724357

Repo: https://github.com/Gomer3261/CSC460

Table of Contents

1.0 Project Introduction:

Project 3 is the culmination of the topics taught in CSC 460. The goal of the project is to get multiple iRobot Roombas to work together to play a game of your choice. For this project my group chose to play cops and robbers. The game consists of two teams of two roombas autonomously navigating the room, and attempting to hit opponents using an infrared signal. When a roomba is hit with an opponents infrared signal it dies. If a dead roomba is hit with an ally's infrared signal it is revived. A base station uses radios to keep track of the states of all roombas within the game, and is in charge of detecting when a game has ended, and starting a new game.

2.0 Primary Hardware:

There are a few core technologies that the system relies on. The first is the ATmega2560 AVR board, which runs our custom operating system and logic for each individual roomba, as well as the base station itself. The second is the iRobot Create 2, a programmable edition of the Roomba. The final primary technology is the nRF24L01 radio, which is used for all communication between the five components of the system.

2.1 ATMega2560

All of our systems run on an ATmega2560 AVR board. The board has 6 timers, 52 digital I/O pins, 16 analog I/O pins, support for the UART transfer protocol, and numerous interrupts [1]. The ATmega2560 processor is a RISK based processor that runs at 16Mhz, with 8KB of ram and 64KB of flash memory, running on 2.7V - 5.5V of electricity [2]. In this project one board is used as a primary base station for the game, and one board is used to control each iRobot Create 2. The boards are powered either via a wall plug, usb, or the Create 2's internal battery.

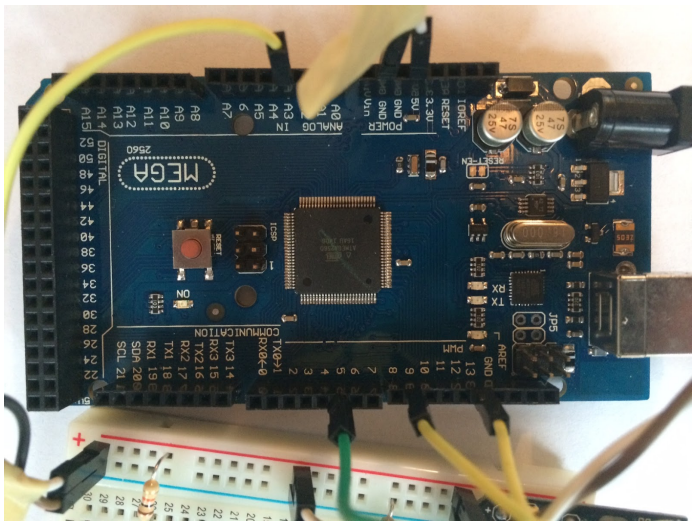

Figure 1: ATmega2560 arduino board.

2.2 iRobot Create 2

The iRobot Create 2 Roomba is a special edition of the Roomba vacuum designed for STEM research fields. The Create 2 has numerous programmable features that are not normally included in the Rooba platform, and is specifically designed to help teach and learn robotics and real time systems. The roomba can be controlled via a serial UART data bridge or a serial-to-usb port controller. The specifics of the roomba create 2's interface can be found on iRobot's website [4].

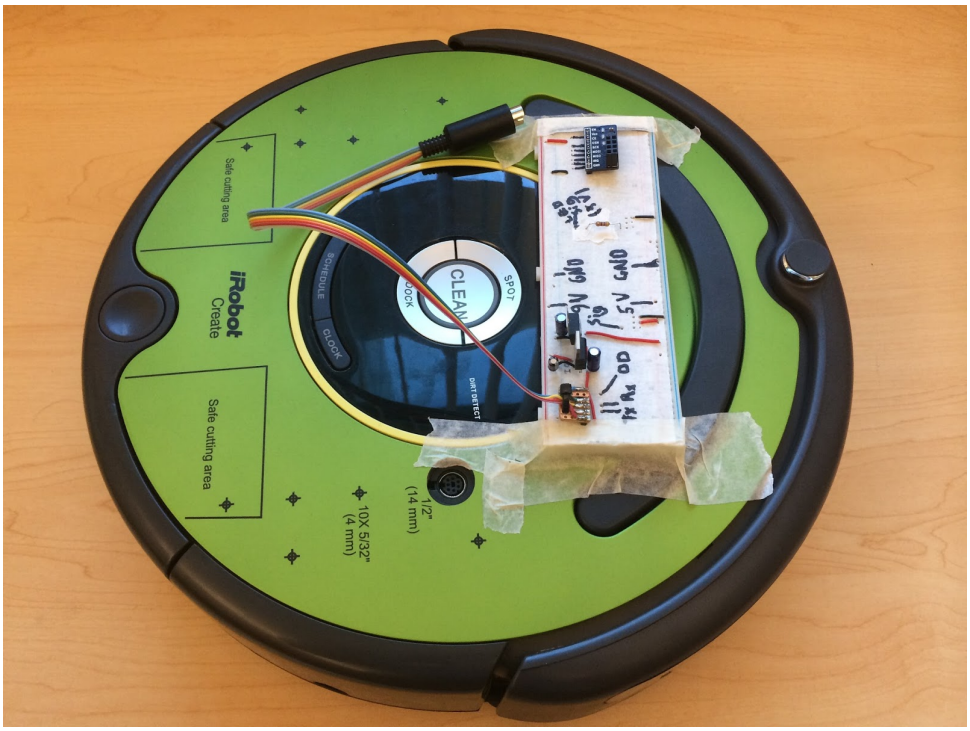

Figure 2: iRobot Create 2 robotic vacuum.

2.3 nRF24L01 wireless radio

The nRF24L01 wireless radio is a chip that allows communication between itself and other chips using radio waves. The radio can send waves on a number of different frequencies, and uses addresses to filter out messages, ensuring only the targeted radio actually processes the sent packets [3]. Incoming packets are stored in an internal buffer that can store four individual packets, but if the buffer is not emptied packets will be dropped once it's full. A hardware interrupt is fired whenever the radio receives a new packet, however if the packet is dropped the interrupt is not fired. An SPI interface is used to transfer data between the radio and its controller.

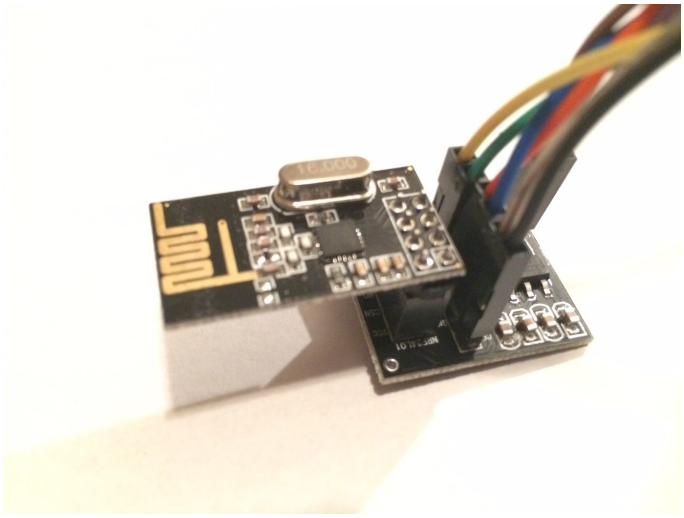

Figure 3: Radio Hardware.

The radio uses a set of internal memory to store packet data that can hold packets of exactly 32 bytes in length. This memory acts as a buffer and can hold a limited number of packets before overflowing. The memory can be wiped by toggling the power line off and back on again.

The radio runs at 3.6V, which is not provided by our hardware. To provide power to the radio without damaging its circuits we use a special base chip that drops the voltage for the radio. The base also has the nice benefit of making the wiring for the radio a little more clear.

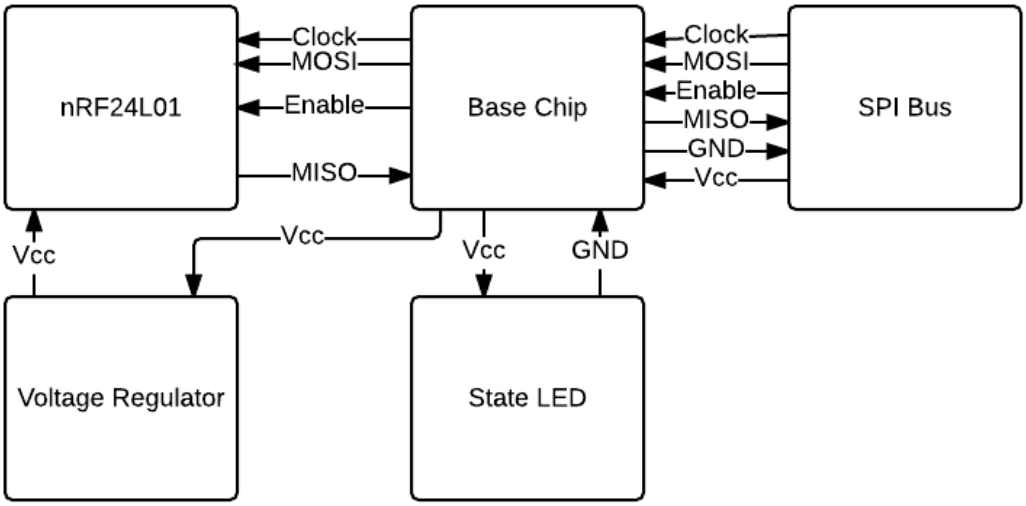

Figure 4: Radio block diagram.

The radio has 2 lines: MISO for sending buffered data to the arduino board, and MOSI for receiving packet data from the board to broadcast. Additionally, the radio uses a control line for firing an interrupt when a new packet is received, and 3 control lines for radio state.

The radio is capable of transmitting at 2 Mbps along the 2.4GHz band (ISM) consuming a maximum of 3.6V.

3.0 Base Station:

The base station is the central control unit for the cops and robbers game. The game state and roomba states are ultimately controlled by the base station.

3.1 Overview

The base station is the central component for radio based communication. The state of the cops and robbers game and each roomba currently playing are controlled by the base station. The station is responsible for ensuring all roombas are kept up to date on what is happening in the game, and detecting when a game is completed.

A set of debug LEDs are used to visually represent the game for debug and demonstration purposes. A joystick is used to provide user control over the base station. The joystick allows a controller to start a new round of the game, and force kill any player within the game. This feature is useful primarily for testing and control over the roombas should something go wrong.

Demonstration Video: https://www.youtube.com/watch?v=o1UZjN8_TYE

3.2 Hardware

The hardware we implemented acts as a base station for the roomba, sending the Roomba commands, and controlling how it operates. The base station is also an ATmega2560 board, controlling the nRF24L01 radio hardware, and a joystick for input. The joystick is used to provide external control over the gamestate and shut down roombas if necessary.

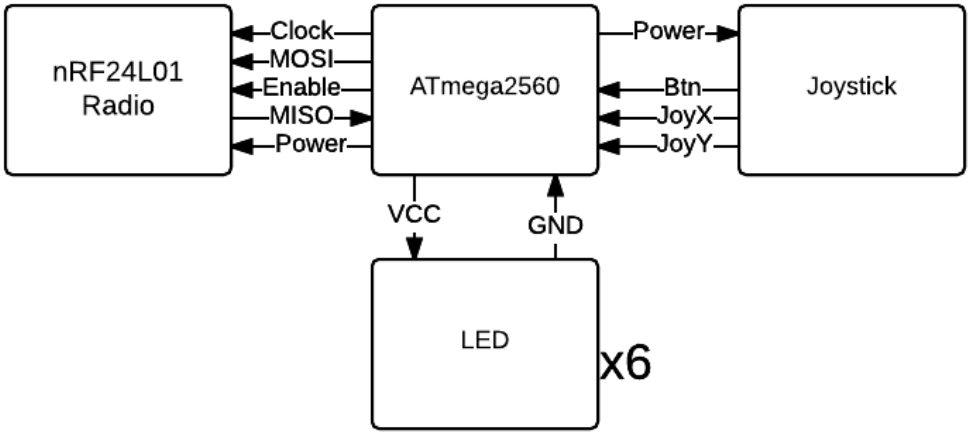

Figure 5: Base station hardware block diagram.

3.2.1 Joystick

A simple dual axis sony joystick was used for user input in our system. The joystick supports 2 axes of movement and a simple push button.

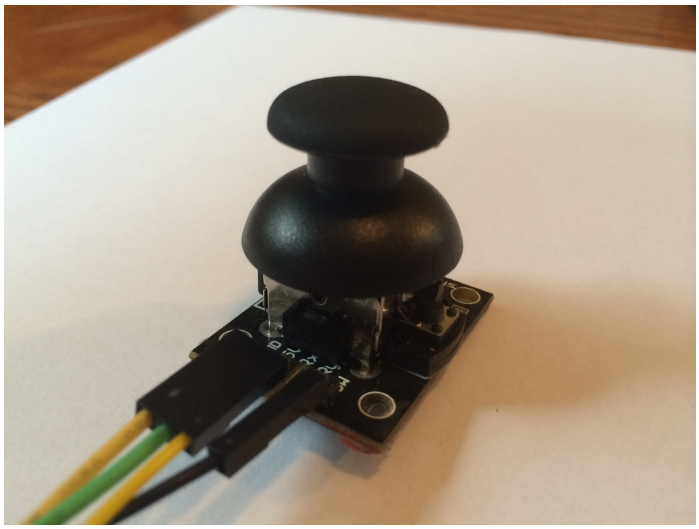

Figure 6: Sony Joystick.

The joystick button emits an unstable signal when not pressed. When it's pressed the circuit is grounded through the button. The signal is stabilized using a pull up resistor. A 5V input is hooked up in parallel with the button using a 10KΩ resistor that leads to ground (see figure 7). The resulting circuit is connected to a digital pin on the ATmega2560 board. When the button is open an analog signal of 1023 is received in the digital input, resulting in a high value. When the button is pressed, a much easier path to ground is created through the button for the 5V power line, and the connected pin receives 0 or a low signal.

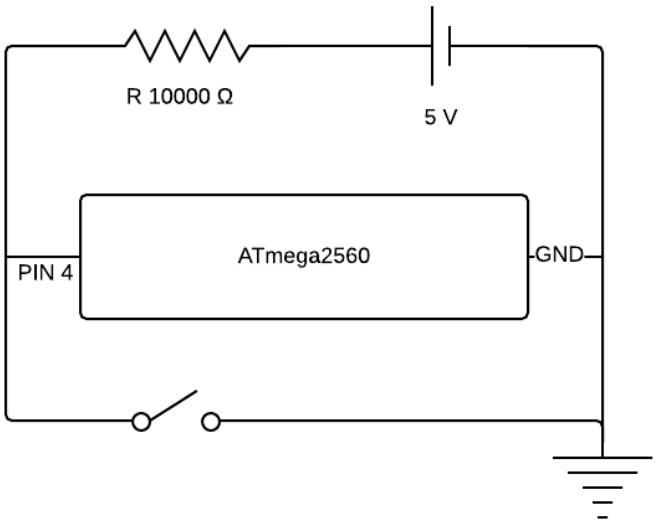

Figure 7: Joystick button circuit.

3.2.2 Debug LEDs

We made use of six debug LEDs for state display on the base station.

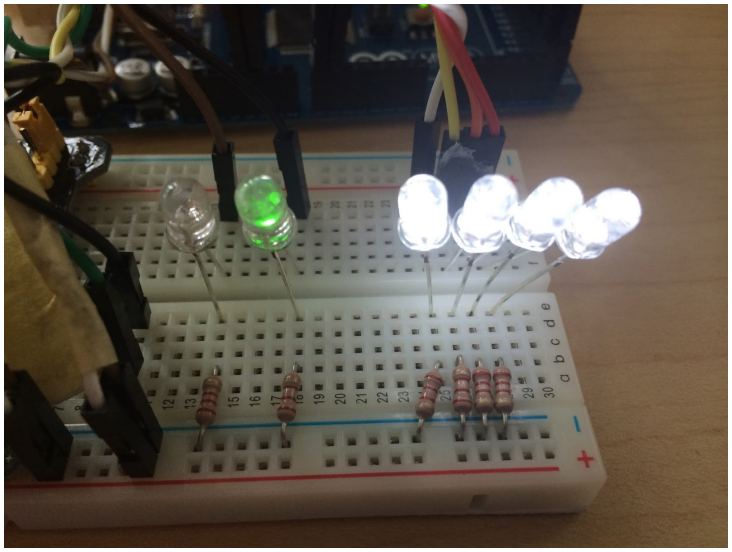

Figure 8: Debug Leds.

The LEDs are hooked up with 2.2 kΩ resistors, each powered by a digital pin on the ATmega2560. The voltage travels through the LED into the resistor and into ground. A simple port map is used to control the state of each LED.

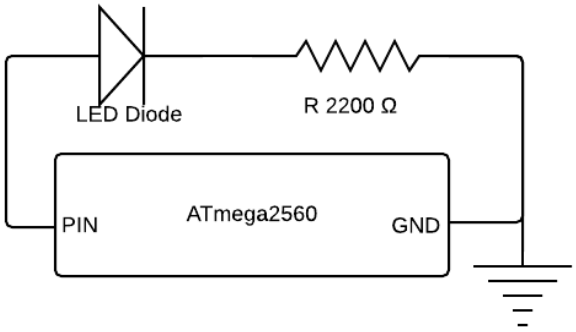

Figure 9: Debug Led Circuit.

3.3 Game States

The base station is in control of the cops and robbers game state. The system has three separate states.

The first is game_starting which is the initial state of the system. Inside game_starting all roombas are forced to be alive, and are not allowed to move. When the user presses the joystick button the game_starting state changes to game_playing.

The game_playing state encompasses the actual cops vs robbers game. When this state is entered each roomba is brought to life. During this state roombas are responsible for reporting their own alive state to the base station and ensuring that it has correct information. If both roombas on one team die, the base station transitions to the game_over state.

The game_over state is used to inform the roombas that they are no longer playing. In this state the base station is in control of the roombas' living states. During this state live roombas are free to celebrate their victory, but no shots may be fired, and no roomba may change state. When the user presses the joystick the state is changed to game_playing again, and another round starts.

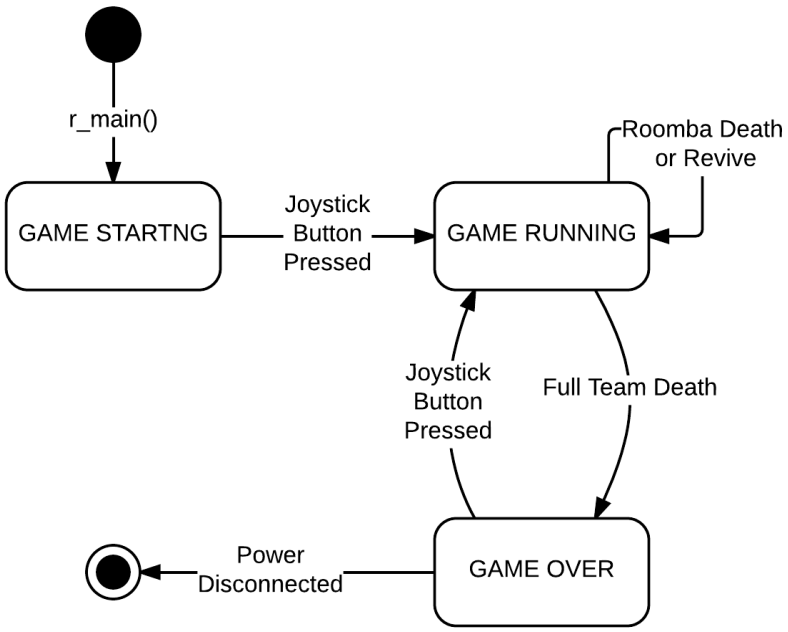

Figure 10: Game state State Diagram.

3.4 Radio Communication

The drivers for our nRF24L01 radios are based on the drivers from project one. They are implemented in an OS independent form, however they are designed to run on the ATmega2560. We made small modifications to improve reusability and move control to our main script.

Two primary packets are used as part of the cops and robbers game. These packets are specially designed for cops and robbers, and are extensions of the packet format provided with the radio drivers.

Game State Packet:

game_state (1 bytes) - The current state of the game.

roomba_states (4 bytes) - An array of one-byte states for each roomba. The current states known by the base station.

Roomba State Packet:

roomba_id (1 byte) - Unique identifier for the roomba responding to a gamestate packet.

roomba_state (1 byte) - The correct state of the roomba corresponding to the provided ID.

The base station sends a game state packet to the roombas four times a second. The packet is sent to a generic address that all roombas share. The station only requires an ack from one roomba before the next packet is sent. This format is used to keep data flowing to the roombas as fast as possible, without requiring any extra communication or handshakes. The station essentially broadcasts its state.

If a roomba does not agree with its corresponding state in the received game state packet, it responds once with a roomba state packet. The roomba state packet simply tells the base station what the roomba's state actually is. When a packet is received by the base station, the gamestate is updated with the new roomba state. If the roomba is happy with the provided gamestate, no manual response is sent beyond the automated ack packet.

All communication runs on frequency 102. We use address [0xff, 0xff, 0xff, 0xff 0xff] for the base station and address [0x4A, 0x4A, 0x4A, 0x4A, 0x4A] for each roomba. Since the radios in our design are similar to UDP there is no need to know which roomba the packet is intended for. This allows the base station to simply broadcast packets to all four roombas at once.

3.5 Software

The base station is driven by a simple set of software. In order to utilize as much of the CPU as possible we are using our real time operating system (RTOS) developed for project two. The RTOS allows for context switching and different task scheduling schemes. We make heavy use of the RTOS for both timing and CPU optimization in this project.

3.5.1 RTOS

Our RTOS is capable of running three tiers of tasks. The lowest is a tick-based round robin. Tasks are taken off the round robin queue, run for one tick (5ms) and placed back on the end of the queue. The middle priority tasks are periodic. Each periodic task is defined by period, worst case execution time, and a start delay. Periodic tasks will take priority over round robin tasks. When the periodic task is complete, round robins will resume. Finally the RTOS supports system level tasks. System level tasks are the highest priority and will even take processing time from periodic tasks. System tasks should never loop and are usually triggered by interrupts or specific events in the system's logic. Our base station implementation makes use of all three levels of tasks.

The operating system also supports services. A non-periodic task may subscribe to a service. When this happens the task is set to idle and will not be run by the OS. A separate task or interrupt may publish to a service, which will release all subscribed tasks for execution. We make use of services to control when system tasks are run within the OS.

3.5.2 Task Allotment

Our base station makes use of six tasks and two interrupts. The tasks are allocated as follows:

```
radio_send_service = Service_Init();
radio_receive_service = Service_Init();

Task_Create_System(send_packet, 0);
Task_Create_System(receive_packet, 0);
Task_Create_Periodic(send_state, 0, 50, 5, 1000); // 4 times a second.
Task_Create_RR(user_input, 0);
Task_Create_RR(update_gamestate, 0);
Task_Create_RR(display_gamestate, 0);
```

Snippet 1: Base station task allocation.

A single periodic task is used to determine the rate at which the base station sends messages to the roombas.

The send_packet and recieve_packet system tasks are blocked on their respective services. When the radio causes an interrupt indicating that a packet has been received the receive service is published to, causing one loop of recieve_packet to execute. The loop handles all packets on the radio's queue.

The send_packet system task is executed by the send state task. The task simply sends a packet, however a system task is used to ensure the data transfer cannot be preempted. Each time send service is published to, a single game state packet is sent by the task, which promptly subscribes to the service again afterwards.

By placing these tasks at a system level we can ensure that the radio is never blocked, and that nothing ever interrupts the important parts of send and receive logic. This ensures that the radio state is always stable.

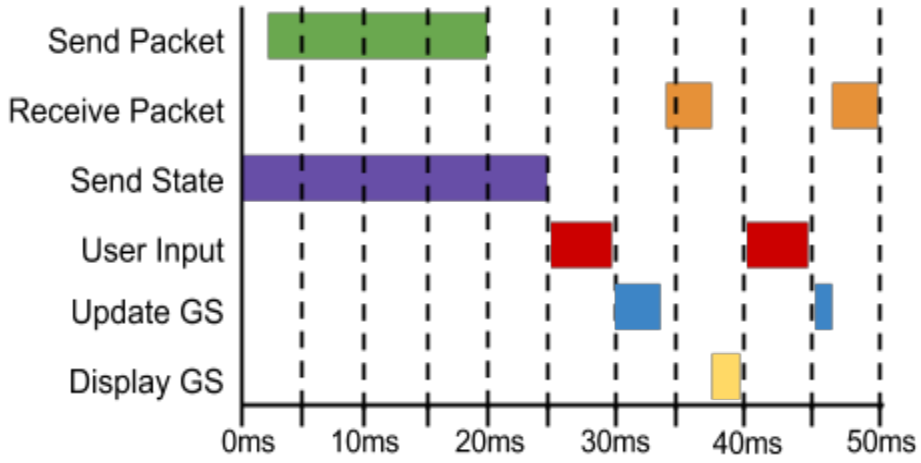

Figure 11: Base station task scheduling diagram.

A periodic task is used to trigger game state messages to roombas. The periodic task controls timing and rate of radio messages. Additionally, it allowed us to break messages to each roomba up into separate calls to the send_packet task. In practice however the system

worked much better if one packet was sent to all four roombas at once. Without the need to send four separate packets the periodic task simply acts as an alarm that reactivates the send_packet system task every 0.25 seconds.

Task	Priority	Purpose
`send_packet`	System	Sends a single packet to a single address as a system task.
`receive_packet`	System	Processes all packets on the radio queue.
`send_state`	Periodic	Triggers the base station to broadcast gamestate 4 times a second.
`user_input`	RoundR	Polls joystick data and applies controls.
`update_gamestate`	RoundR	Handles automatic game state changes. (Win conditions)
`display_gamestate`	RoundR	Updates the hardware LED display.

Table 1: Base Station Tasks.

Finally, three round robin tasks are used to handle less important functionality within the base station. One task listens to the user input via the joystick and updates the game state accordingly. Another checks the game state for win conditions, causing it to change based on updates received from the roombas. The final task simply updates the LED displays to correctly represent the gamestate. Each round robin task runs for at most 5ms. However, by using Task_Next() to freely give up operation time, the RTOS will cycle between the three tasks as quickly as possible.

4.0 Roomba:

Roomba is a brand of robot created by iRobot. Specifically we are using the iRobot Create 2 Series robots. The Create 2 is designed specifically to help teach STEM fields. In this project our Create 2 robots fill in the role of players. The robots are autonomous and communicate regularly with the base station in order to understand the state of the game they are in.

4.1 Overview

Each Create 2 is designed by an individual team of two members to play our cops and robbers game. Each robot is controlled by an ATmega2560 board running the individual team's own RTOS and logic code. Additionally, the strategies and autonomous behavior of the robot is defined and programmed by the teams themselves. In this way we produce four separate robots that act different within the game. The end result is a dynamic autonomous game of cops vs robbers that features four Create 2s each with their own design and personality.

4.2 Hardware

The Create 2 is mounted with a few extra pieces of hardware that are necessary to implement our cops and robbers game. The most important components are an IR transmitter and receiver and the nRF24L01 wireless radio detailed in section 2.3. A number of sensors built into the Create 2 were used for navigation and autonomy, while the external sensors are used for monitoring game state, and shooting or being shot by other devices.

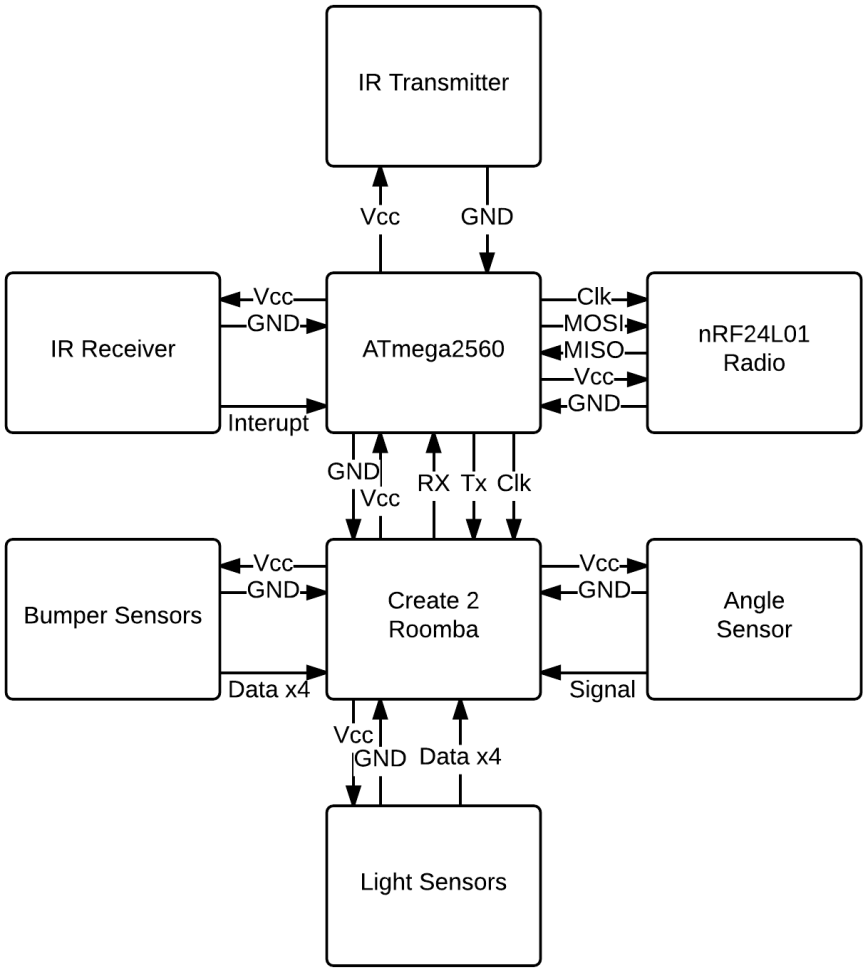

Figure 12: Roomba hardware block diagram.

4.2.1 IR Transmitter

The IR Transmitter is a simple IR LED that produces IR (Infra-Red) light when powered. This LED is blinked on and off following a specific protocol to transmit an 8-bit value to any visible receiver. These different signals are used to identify the team of the roomba that is shooting.

The IR protocol is split into bits. Each bit is represented by a 500µs high or low (On or Off) signal. During each bit the signal pulses every 26.31µs (38000 times per second) providing a signal frequency. This allows the receiver to ignore ambient IR and other potential signals within the room transferring at different frequencies. By turning the signal off during low bits and on during high bits, a proper bit transfer can be achieved. The bit transfer is initiated by a high bit followed by a low bit, and then the 8 bit signal follows from lowest to highest bit.

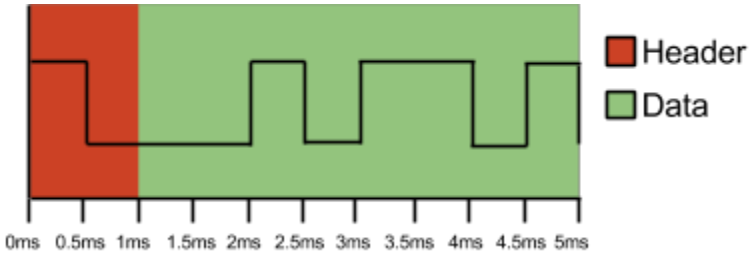

Figure 13: IR Protocol Image.

Since LEDs have little to no resistance, the IR transmitter has to be wired in series with a 2.2KΩ resistor. By using the resistor, current can be properly controlled and risk of damaging the ATmega2560 board is removed.

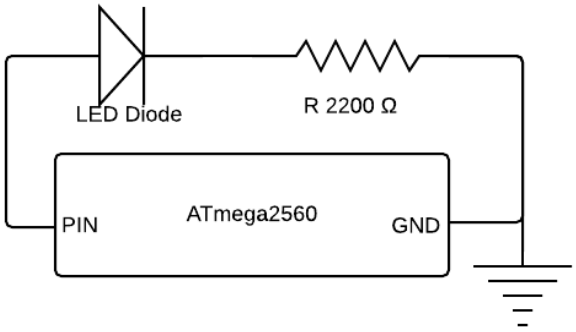

Figure 14: IR transmitter circuit diagram.

The IR transmitter was wired to the ATmega2560's timer1 on a Pulse Width Modulation (PWM) setting to pulse the LED matching the required signal frequency. ATmega2560's timer3 on Clear Timer on Compare (CTC) mode is used to fire an interrupt every 500µs allowing the software to change the transferred bit. By controlling the length of the high portion of the PWM timer (between 0 and 50% of the pulse length) the IR transmitter can be turned on and off.

4.2.2 IR Receiver

An IR receiver is used to detect incoming IR signals from other roombas. The receiver works using a light sensitive diode and an internal low pass filter to limit responses to light in the IR range. [5] The device is protected by a shell to prevent our own Roomba's transmitter from affecting the device.

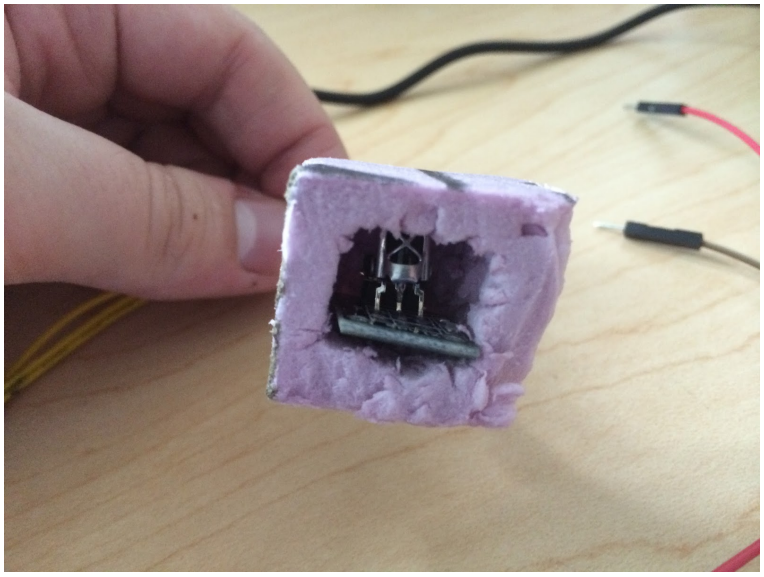

Figure 15: IR Receiver casing

The receiver uses 3.3V - 6V of power, and has a ground and output line. The output line is high when IR light is detected and low otherwise.

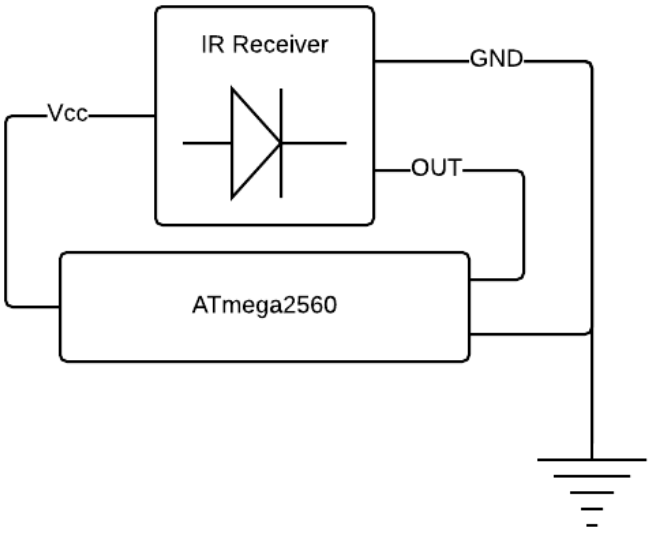

Figure 16: IR receiver circuit diagram

By listening to the IR signal and filtering for the correct frequency using software, our IR library (written by Curtis St. Pierre and Mark Roller based on provided code and previous work) then listens to the filtered sensor data to determine each bit within a signal. When a full piece of data is detected, an interrupt is called within the RTOS in order to handle the signal however the OS pleases.

4.2.3 Roomba Control (UART)

In order to pass commands to the Create 2, and receive sensor data, we use the Universal Asynchronous Receiver/Transmitter (UART) standard. The UART standard supports two-way serial communication between two devices. Using a single timer to control data transfer speeds, the UART protocol transfers a single bit at a time across two lines (one to transmit, one to receive) that connect the two devices.

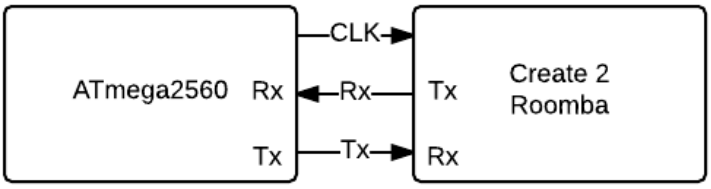

Figure 17: UART block diagram

UART is used to send packets to the Create 2, and receive data back in response. Our UART drivers were provided by the class, however modifications needed to be made to correctly initialize the ports on our ATmega2560 board. Thanks is given to Curtis St. Pierre and Mark Roller for their assistance debugging this issue. With UART in place our RTOS is capable of providing commands to the Create 2 via the Create Open Interface. [4]

4.2.4 Roomba Sensors

The Create 2 has a large number of onboard sensors. The available sensors can all be seen in the documentation for the Create 2 interface, however for this project we only used three sets. We make use of light and bumper sensors (seen in purple and orange in figure 18), as well as distance and angle sensors (seen in blue and green in figure 18). These sensors are retrieved as part of different data packets that can be requested from the Create 2. In particular we make use of three packets.

The first is the Bumps and Wheel drops, which contains five bit flags in a single byte. The two end bits contain bumper left and bumper right state, and are used for detecting walls and impacts. The other three bits contain wheel state. Should a wheel drop (Cliff detection, or lift detection) they can be used to prevent the Create 2 from driving off an edge.

The second packet is for internal sensors. Primarily we make use of distance and angle sensors, which record the distance covered since the last time the packet was requested. Additional data for charge state is available in the packet, however we do not make use of it in our implementation.

The third packet is for light sensor data. The light sensors have the ability to detect walls and surfaces before impact. By using light sensors we attempt to prevent the Create 2 from hitting walls altogether, however the sensors do not always pick up obstacles, which is why the bumper is used as a backup solution.

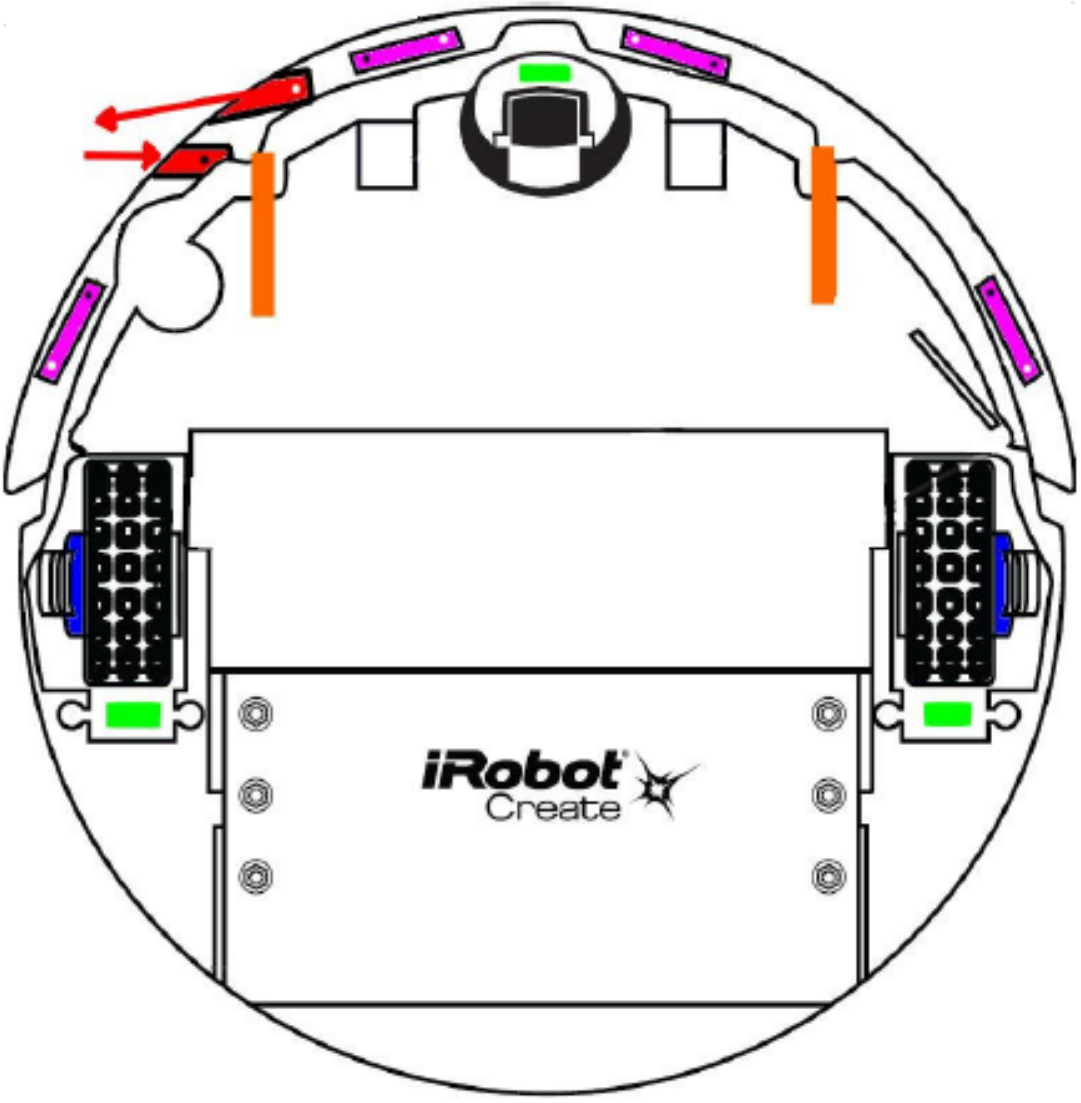

Figure 18: Create 2 Sensors (credit: https://labviewhacker.com/)

Packets can also be sent to the Create 2. We make extensive use of the drive packet, which provides motor instructions to the robot. Additionally, we use the restart packet to set the Create 2 in the correct state for operation whenever the RTOS starts.

4.3 Software

The Create 2 controller runs on a similar system to the base station. The system utilizes our RTOS as well as a similar set of networking tasks. The system also communicates with the Create 2 periodically and runs a full set of decision making tasks that allow it to perform a navigation pattern and react autonomously to obstacles. This is all done while replicating the game state detailed in section 3.3, and dealing with the two IR devices.

4.3.1 Task Allotment

Our Create 2 controller makes use of five tasks and two interrupts. The tasks are allocated as follows:

```
radio_receive_service = Service_Init();
radio_send_service = Service_Init();

Task_Create_System(radio_receive, 0);
Task_Create_System(radio_send, 0);
Task_Create_Periodic(roomba_interface, 0, 20, 8, 200);
Task_Create_RR(user_input, 0);
Task_Create_RR(decision_making, 0);
```

Snippet 2: Base station task allocation.

A periodic task is used to control communication with the Create 2, while two round robin tasks control decision making and data processing. Finally, two system tasks are used to deal with packet communication with the base station.

The radio_receive and radio_send system tasks are blocked on their respective services. When the radio causes an interrupt indicating that a packet has been received the receive service is published to, causing one loop of the radio_receive task to execute. The loop handles all packets on the radio's queue. Each packet contains a game state update from the base station. If the game is running, and the packet contains the incorrect living state for the Create 2, a response packet is sent.

The radio_send system task is executed immediately after the radio receive task. The task assembles a roomba state packet and sends it to the base station. A system task is used to ensure the data transfer cannot be preempted. Each time send service is published to, a single packet is sent by the task, which promptly subscribes to the service again afterwards.

The logic for radio tasks follows that of the base station fairly closely. The only real differences are in the way a packet is handled, and the trigger for sending a packet to the other device.

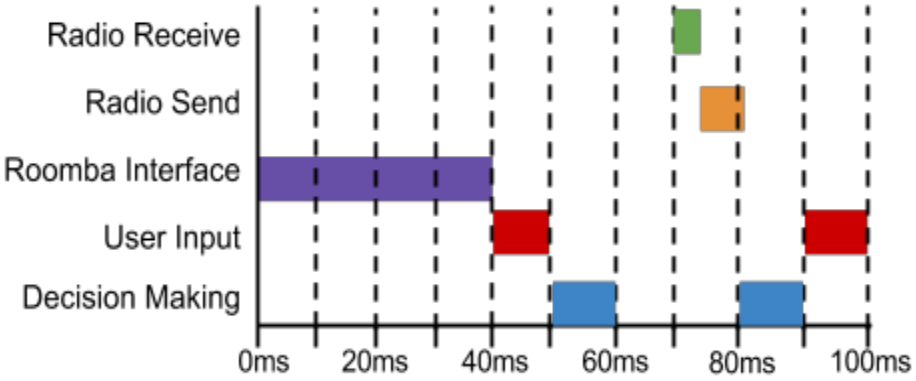

Figure 19: Create 2 controller task scheduling diagram.

A periodic task is used to communicate with the Create 2. The periodic task is run every 100ms (i.e. ten times a second). The task actually has three separate running modes, each performing a different action to the roomba. This is done because querying sensor data from the roomba can take up to 17ms. Sending an IR packet can take six, and a drive packet can also take six. We use three different sensor packets. Thus, in order to do everything with the roomba it would take approximately 56ms, which is a very slow process. To keep the system running quickly and give every task the time it needs to run, we limited the periodic task to pull one packet each iteration. This keeps the WCET of the task under 40ms even when interrupted by a radio task. The design emulated a round robin, pulling each packet in order at even rates.

Task	**Priority**	**Purpose**
`radio_receive`	System	Called to process packets from the base station.
`radio_send`	System	Assembles a roomba state packet and sends it to the base station.
`roomba_interface`	Periodic	Retrieves roomba sensor packets. Fires IR and gives the Roomba movement commands.
`user_input`	RoundR	Polls joystick data and applies controls for debug purposes.
`decision_making`	RoundR	Handles state machine and automation decision making.

Table 2: Create 2 Controller Tasks.

Finally, two round robin tasks are used to handle less time-sensitive functionality on the Create 2. The first task simply listens to user input. This task is used to debug the roomba's behavior. The button simply acts as an IR hit either killing or reviving the roomba. The second round robin task is responsible for making navigation and automation decisions based on system state. The task looks at sensor and game state data and generates movement output for the roomba.

4.3.2 Game States

The Create 2 controller is designed to correctly handle the three game states described in section 3.3. By simply using the states from figure 10 as our primary state controller for decision making, we split the roomba into 3 different modes. The first is idle, which corresponds directly to the game starting state. The second is autonomous, which corresponds to the game running state. The final is a celebration state, which corresponds to the game over state. The primary decision making state machine exists within the game running game state, and is detailed in section 4.3.3.

Demonstration: https://www.youtube.com/watch?v=Fv2k2-gHDM0

4.3.3 Navigation States

We used a basic state-based system to drive our roomba's navigation during play. The system works by driving two meters in a straight line. After the line is complete the roomba rotates 270 degrees firing IR signals rapidly. Once the roomba reaches 270 degrees it goes back to driving in a straight line, and stops firing.

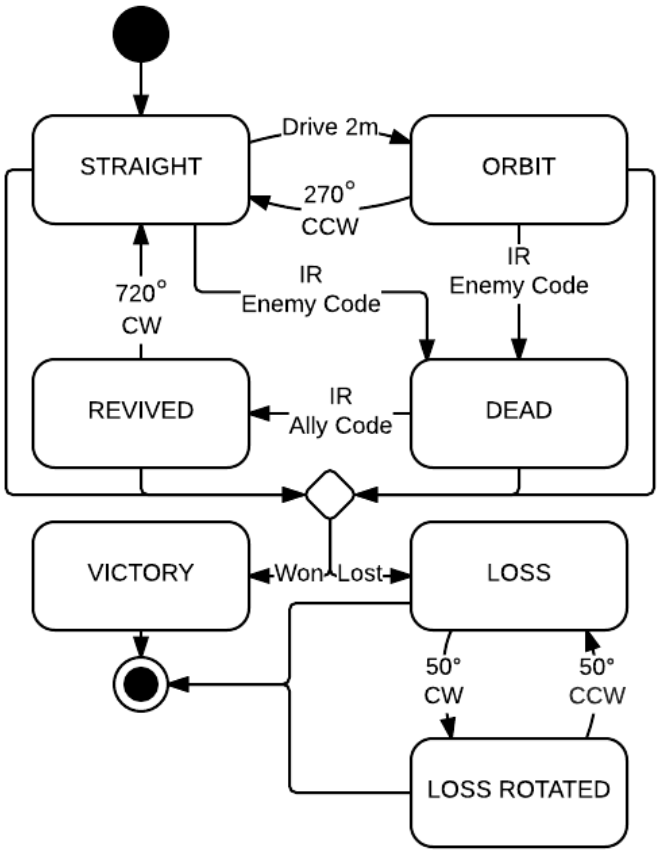

Figure 20: Navigation state diagram.

If the roomba is hit by an opponent it enters a dead state, in which it sits still. When revived from the dead state by an ally, the roomba enters a revived state. In this state, the roomba rotates as fast as possible firing IR as much as possible, in an attempt to protect itself. Once two full revolutions have been completed the roomba returns to its normal patrol route.

This pattern was chosen primarily to give some structure to the roomba's movements, and to try to bring a unique strategy to our final game.

Two additional states exist that occur during the game over game state. The first causes the roomba to spin around in joy after it has won a match. The second causes the roomba to shake its head in despair after losing a match.

Demonstration: https://www.youtube.com/watch?v=MDUcXPSZ2Mc

4.3.4 Automation

Automation comes in to play when the roomba encounters an obstacle. Obstacles may be other players, walls, or even human feet. In our case the only time an impact can happen is when the roomba is driving in a completely straight line. During the straight navigation state, we use light sensors and bumper sensors to detect an imminent or immediate impact with a surface. When an impact is detected, the remainder of the straight state is cancelled, and the roomba immediately begins turning 270 degrees in an attempt to escape the obstacle.

Demonstration: https://www.youtube.com/watch?v=557K5ScJtMI

5.0 Final Demonstration

We brought our base station and autonomous Create 2 robot together with three other teams, each implementing a Create 2 robot designed to interact with our base station using the protocol discussed in section 3.4. The result of our game of cops and robbers can be seen in the following clip.

Demonstation: https://www.youtube.com/watch?v=j16uoO778tk

6.0 Conclusion

By making use of our RTOS from the previous project, four iRobot Create 2s, and five ATmega2560s, we were able to create robots capable of autonomously playing cops vs robbers. Radio communication was used to keep all four robots in sync, and IR signals were used to detect shots robots took at one another. Sensors from the Create 2 robots were used for autonomous behaviour primarily including collision detection and wall detection while performing a patrol route. Our base station and Create 2 robot were successful in communicating and playing with our team members' Create 2 implementations, and a series of games of cops and robbers were played as seen in section 5.0.

References:

1. http://www.atmel.com/images/doc2549.pdf
2. http://www.atmel.com/devices/ATMEGA2560.aspx
3. http://www.nordicsemi.com/eng/Products/2.4GHz-RF/nRF24L01
4. http://www.irobot.com/~/media/MainSite/PDFs/About/STEM/Create/create_2_Open_Interface_Spec.pdf
5. https://www.sparkfun.com/datasheets/Sensors/Infrared/tsop382.pdf
6. http://whatis.techtarget.com/definition/UART-Universal-Asynchronous-Receiver-Transmitter