COMP 490

Final Report

Winter 2017

Amir El Bawab 27132565
Matthew Davis 26685056

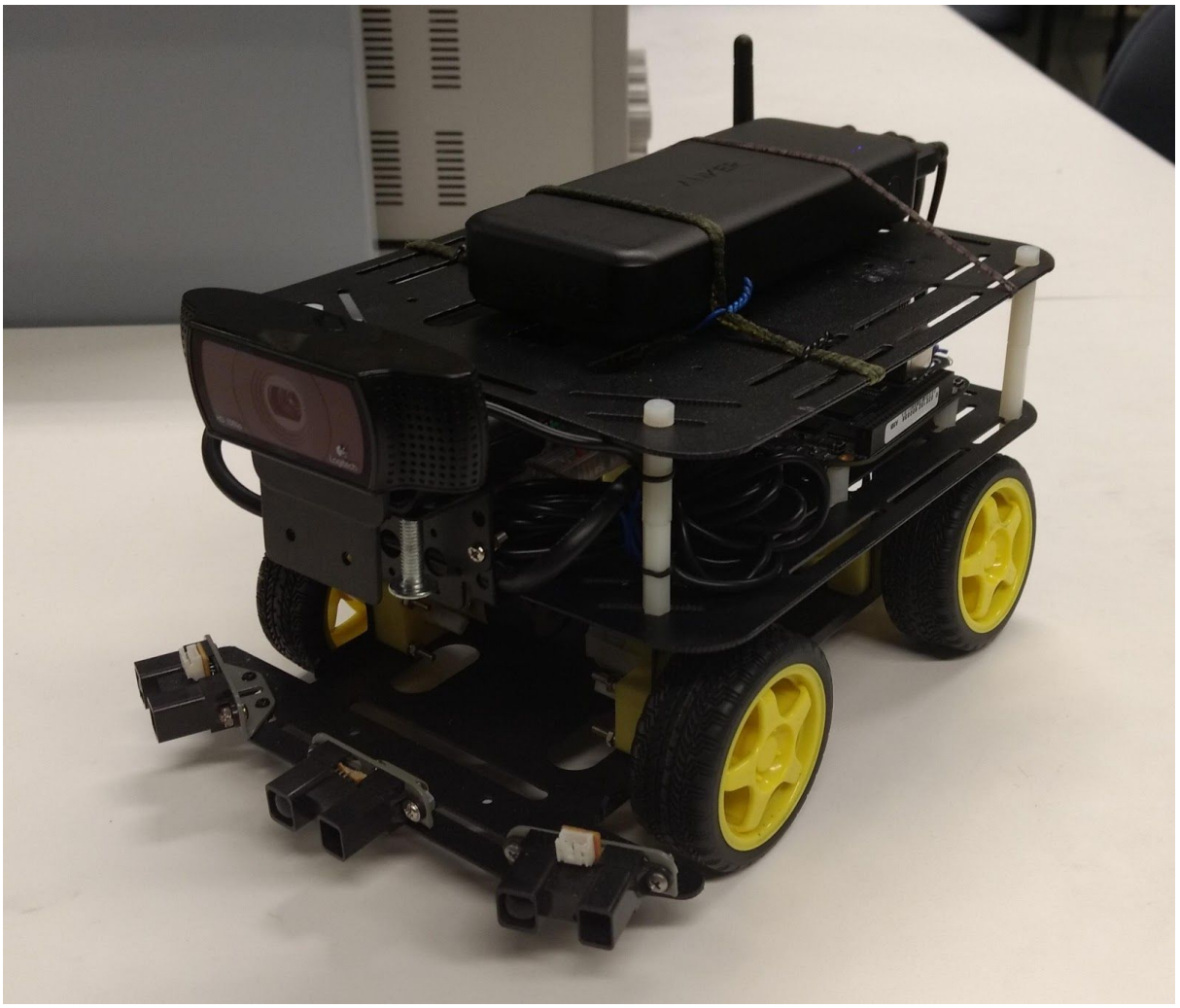


Figure 1

1 - Introduction

Autonomous cars are a hot topic nowadays in technology. Google, Uber, and Tesla have already introduced self driving cars, which are already roaming the streets of San Francisco and other cities. The automobile market expects most top vehicle companies to have self driving cars by 2020. For this project we decided to work on a simplified version of a self driving car.

1.1 - GitHub Repository

All the code, tools, and scripts in this document are available at the following git repository.
https://github.com/ml-davis/self-driving-car

1.2 - Tools Used

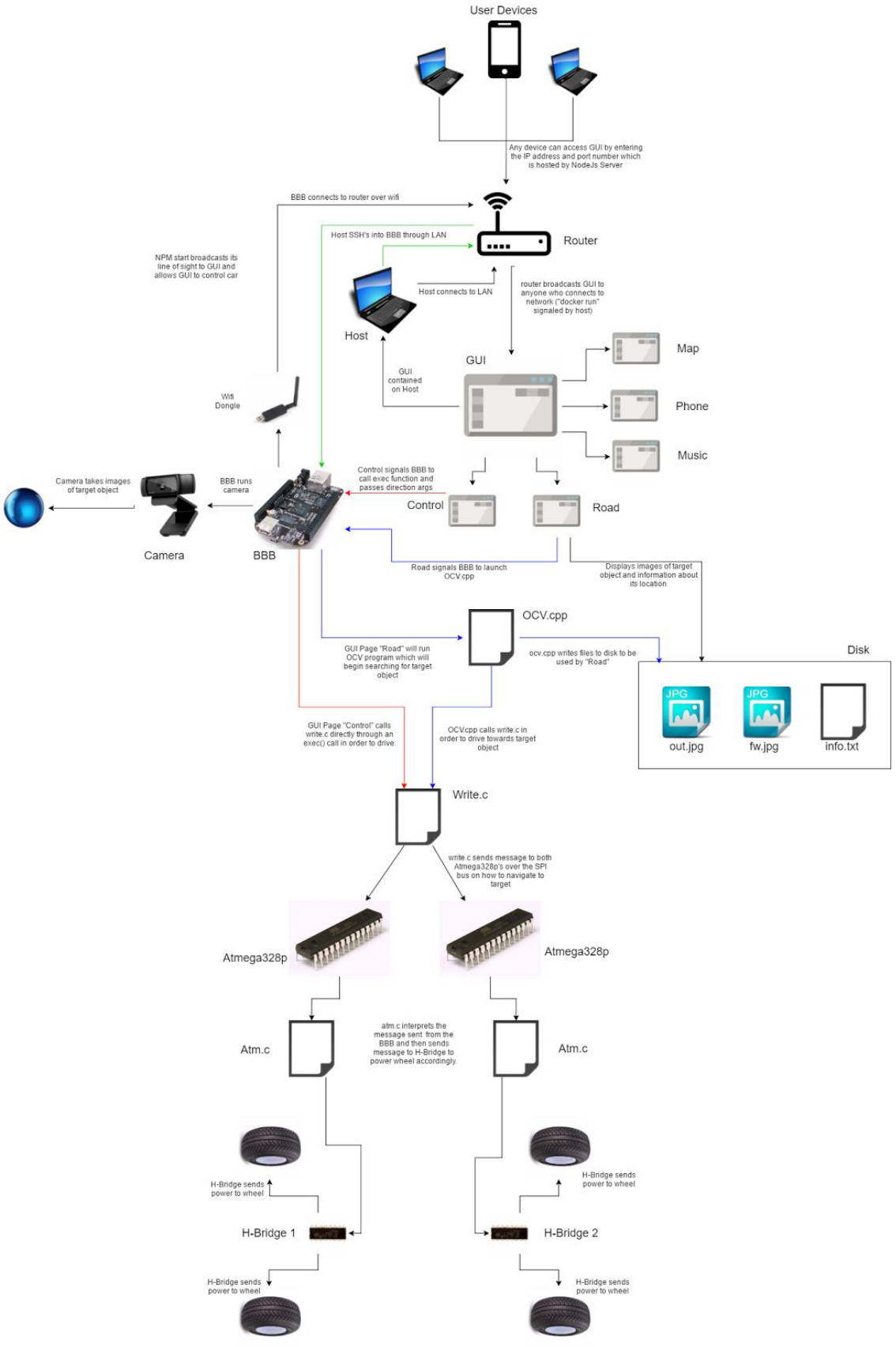
1.2.1 - Hardware Tools

- Car Frame with 4 Wheels and 4 Motors
- Beaglebone Black
- Atmega328p (2x)
- H-Bridge (2x)
- Logitech C920 Webcam
- Level Shifter
- FTDI Basic (2x)
- Wires
- Resistor (2x)
- Capacitor (2x)
- Anker Powercore II 20000 mAh Power Bank
- USB to 2.1mm Barrel Jack Cable
- USB to Mini-USB Cable (2x)
- USB-Hub
- Wifi Dongle

1.2.2 - Software Tools

- Debian (OS)
- Bash
- C
- C++
- OpenCV
- Node JS
- Angular 2
- Docker
- Git

1.3 - Abstract View



We can see in the figure displayed above, an abstract view of how our project works. In the following report, we will go into detail of all the stages.

2 - Setting up our Circuit

2.1 - FTDI Basic

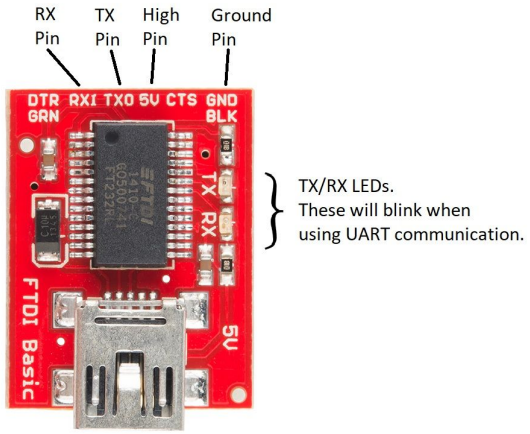


Figure 3

Our FTDI Basic is used for two things:

- To deliver power to our circuit
- To provide UART Communication (*see section 4.3*)

We use four pins on the device:

- GND: This is the ground pin. Used to ground our circuit.
- 5V: This is our high pin. Used to deliver five volts to our circuit
- TX: This is the transmit pin. Used to transmit messages through UART.
- RX: This is the receive pin. Used to receive messages through UART.

2.2 - Atmega328p

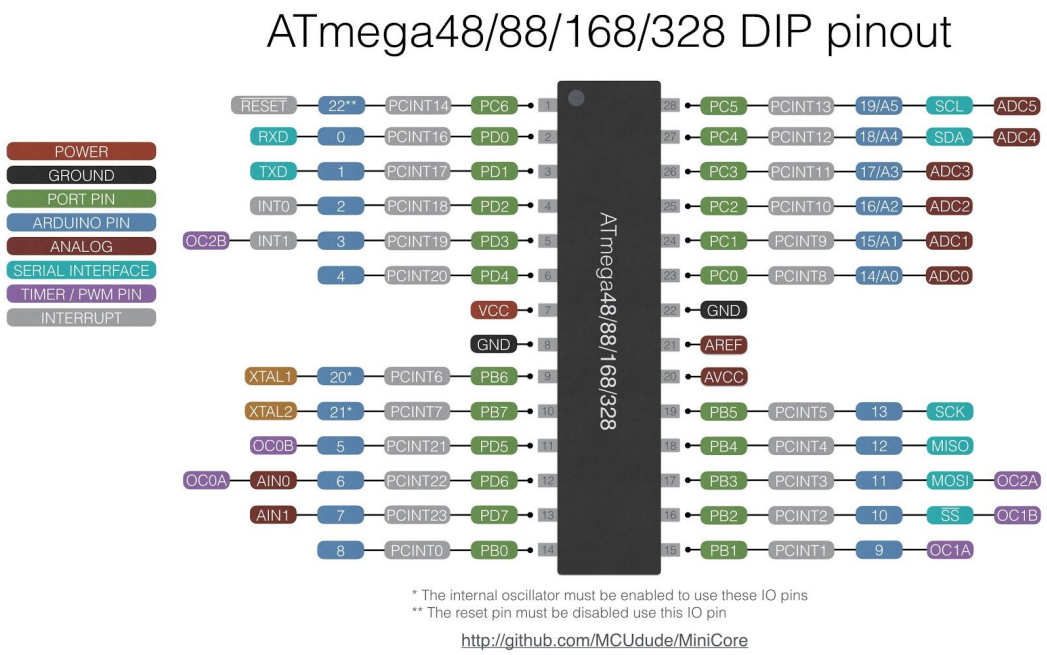


Figure 4

Our Atmega328p is a crucial component of our project. It runs an infinite loop, which awaits messages from the BeagleBone Black. Once these messages are received, they are parsed and interpreted. The Atmega328p requires this information to send signals to the H-Bridges which in turn power the motors that control the wheels on our car.

2.2.1 - Basic Circuit

The first thing we considered when building our circuit diagram, was setting up the basic circuit needed for our Atmega328p. This is needed for basic functionality.

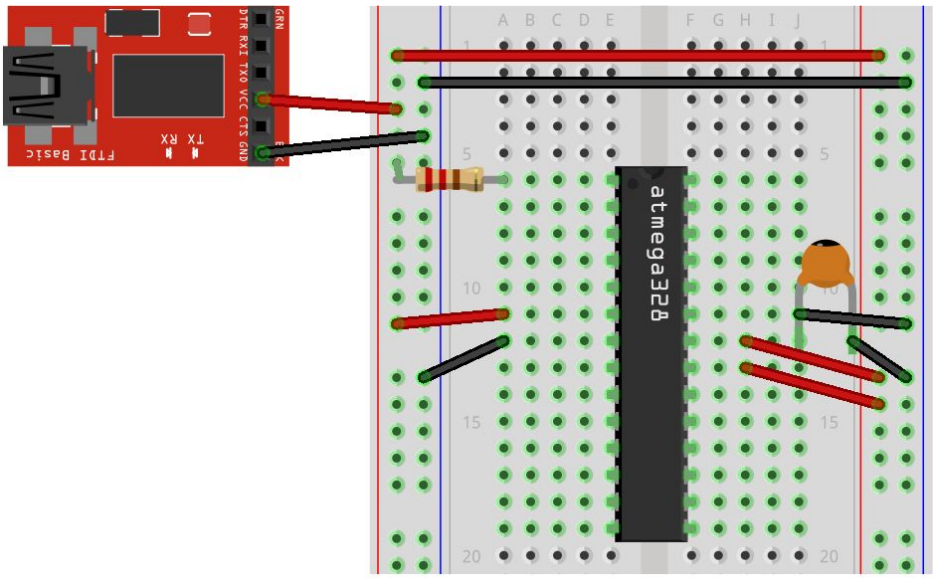


Figure 5

2.2.2 - Connecting to H-Bridges

After the basic circuit is set up, we needed to be able to communicate with the H-Bridges (*see section 2.3*). We used the port pins PD2, PD4, PC2, PC0 to connect to the 1A, 2A, 3A, 4A pins of the H-Bridge. Then we connected the PWM pins PD5 and PD6 to connect to the 1,2EN and 3,4EN enabler pins on the H-Bridge.

Note: It is important that we use pins that support PWM (*see section 4.2*) when connecting to the enabler pins on the H-Bridge. This is because we must be able to pass an analog value (between 0 and 255) which will set the speed of the wheels on our car.

2.2.3 - Connecting to the BeagleBone Black

Our Atmega328p must be able to communicate with our BeagleBone Black. For our project, we decided to use the protocol known as SPI (*see section 4.1*). This uses four pins which are capable of sending and receiving messages. They are labelled on the pinout as pins PB2, PB3, PB4, and PB5.

2.2.4 - Adding UART

The last thing to add is UART communication (*see section 4.3*). This is extremely useful while debugging our Atmega328p programs. It will print messages to the console so that we can see what is happening on the microcontroller.

2.2.5 - Final Atmega328p Circuit

After following section 2.2 our Atmega328p will look like this:

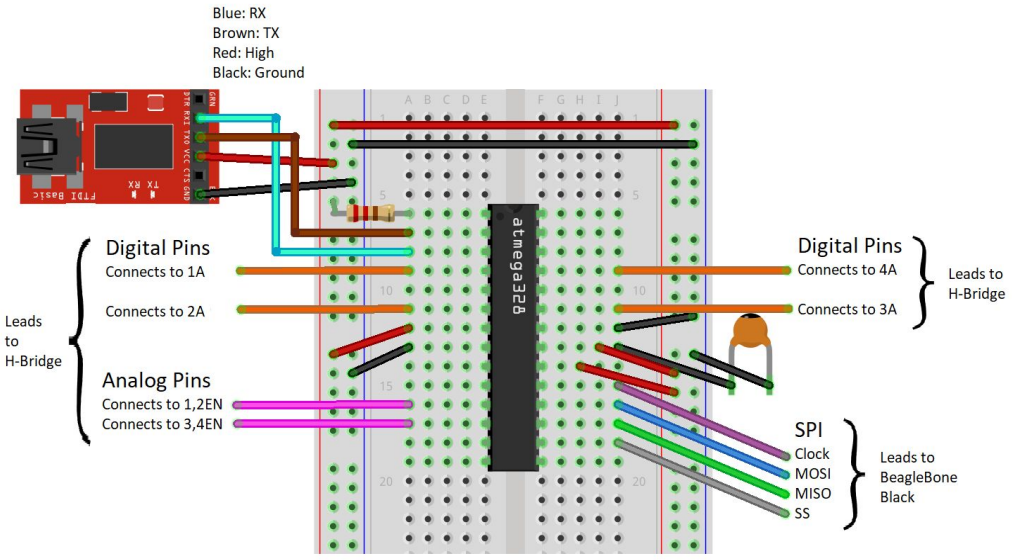


Figure 6

2.3 - H-Bridge

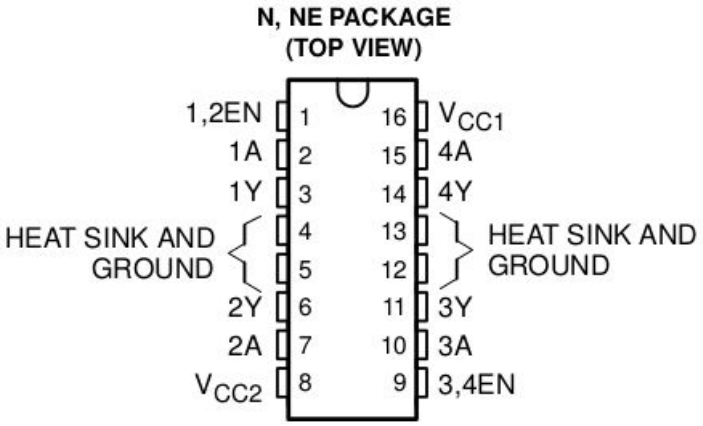


Figure 7

The H-Bridge is used to tell our motors the speed and direction to rotate our wheels.

2.3.1 - Power

Originally, we had our H-Bridges connected to the high line of our circuit. However, we noticed that this caused unstable behaviour in our car. Once we determined the root of our problem, we decided to connect an FTDI Basic directly to the H-Bridge. This seemed to remove the erratic behaviour that we were seeing.

2.3.2 - Digital Pins

The pins 1A, 2A, 3A, 4A receive a digital input (either high or low). Pins 1A and 2A control our left wheel and pins 3A and 4A control our right wheel. The pins will always be opposite of eachother and they will determine the direction that our wheel will spin.

For instance, if we want our left wheel to go forward, pin 1A should be high and pin 2A should be low. If we want it go in reverse, we simply toggle both of them, so that 1A is low and 2A is high.

2.3.3 - Analog Pins

Pins 1,2EN and 3,4EN receive an analog input (a value between 0 and 255). This is achieved by the use of the Atmega328p's PWM feature (*see section 4.2*). 1,2EN controls how much power is delivered to the left wheel and 3,4EN controls how much power is delivered to the right wheel. This allows us to set the speed of our wheels and allows our right and left wheels to operate independent of each other.

2.3.4 - Motors

Pins 1Y, 2Y, 3Y, and 4Y connect directly to the motors of our car. The H-Bridge uses the values that it reads on the digital and analog pins (*see sections 2.3.2 and 2.3.3*) to determine how to power the wheel.

2.3.5 - Ground

Finally, we have our ground. Pins 4, 5, 13, and 12 are simply connected to the ground of our circuit.

After following all steps in 2.3, our circuit should look like this:

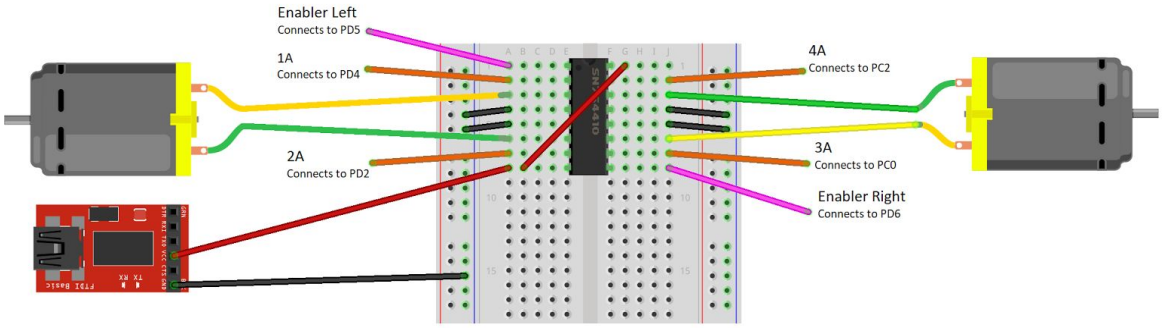


Figure 8

2.4 - Beaglebone Black

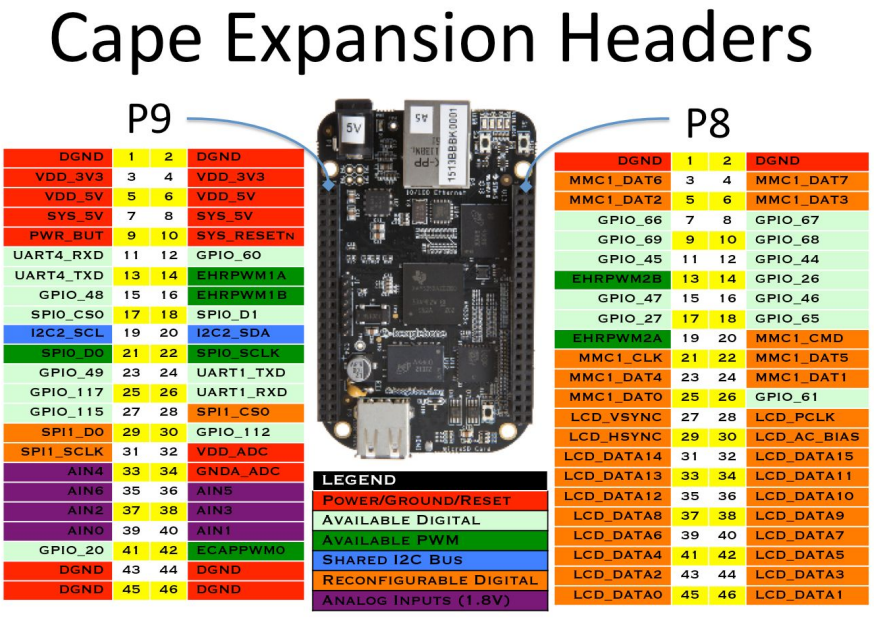


Figure 9

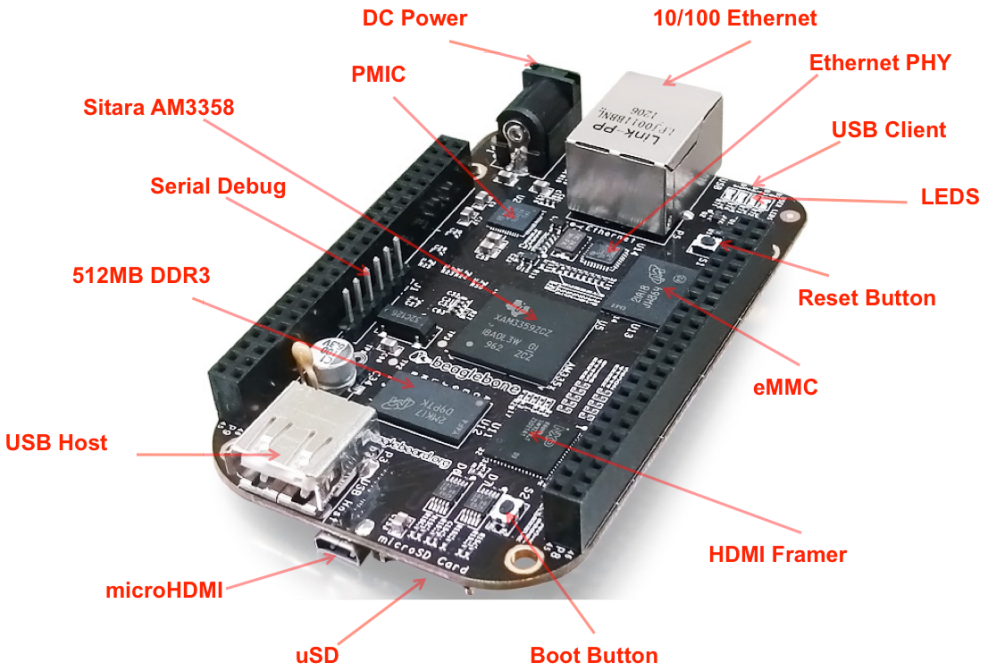


Figure 10

2.4.1 - Power

Since our BeagleBone Black is using wifi, it is necessary that we power it through the 5V barrel power adapter. If we try to simply use the USB Client, we will not supply enough power and our wifi will be unstable and cause all sorts of problems. We picked up an Adafruit usb to barrel cord capable of delivering 5V which we found on Amazon:

Figure 11

2.4.2 - External Devices

The BeagleBone needs to connect to two different devices. Since there is only one USB port, we picked up a USB hub with at least two ports to go into the BeagleBone's USB port.

Connected to our hub is our:

- **Webcam**: used to send images of the car's line of sight to the BeagleBone Black
- **Wifi Dongle**: used so that we can SSH into the BeagleBone and execute programs remotely. The wifi is also used to allow all devices connected to the same network to connect over a TCP socket connection to the graphical user interface.

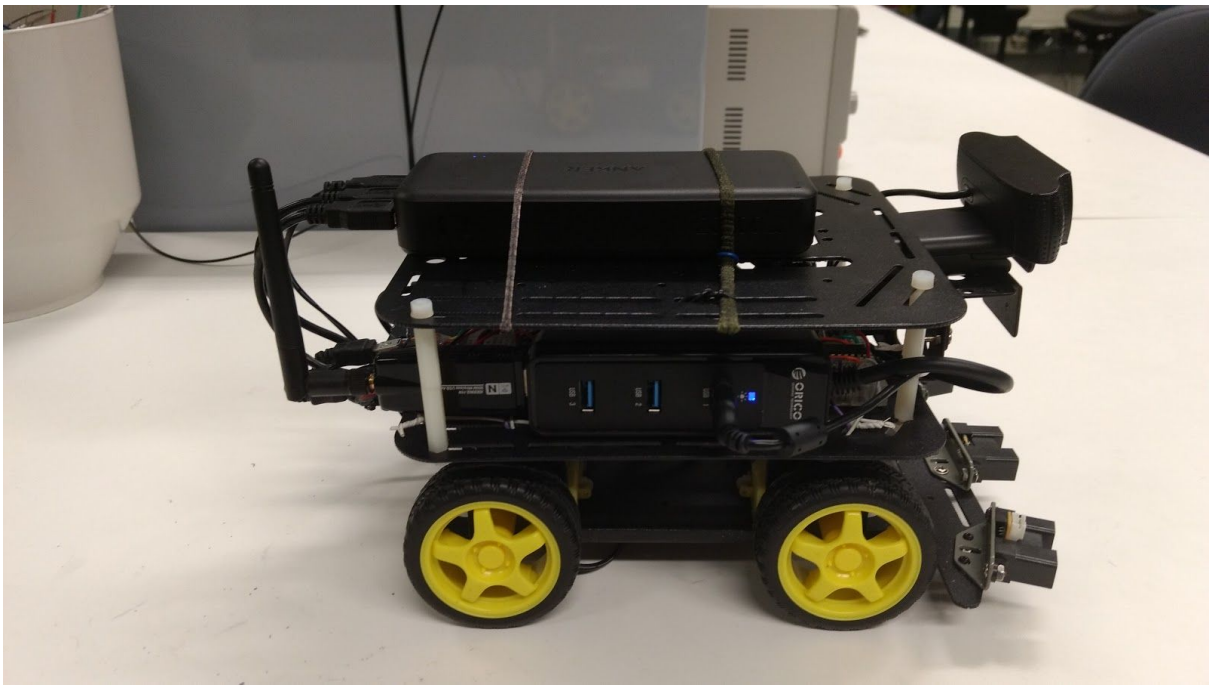


Figure 12

2.4.3 - Communicating to the Atmega328p

Finally, we have our SPI connection which will be used to send messages to the Atmega328p. The four SPI wires connect to:

- P9.17: SS
- P9.18: MISO
- P9.21: MOSI
- P9.22: Clock

We also have a high pin, which is connected to P9.3 and a ground pin which is connected to P9.1.

IMPORTANT: Since the BeagleBone Black operates at 3.3V and our circuit operates at 5V, it is crucial that we place a **level-shifter** in between the two devices. This prevents us from damaging our components. We must connect the BeagleBone to the low end of the level-shifter (signified by L on

the shifter) and we must connect the Atmega328p to the high end (signified by H) to ensure that the voltage is raised or reduced accordingly.

Figure 13

2.5 - Battery

Finally, we have the power source for our car. We chose to go with an Anker PowerCore II 20000 mAh power bank.

Figure 14 ***Figure 15***

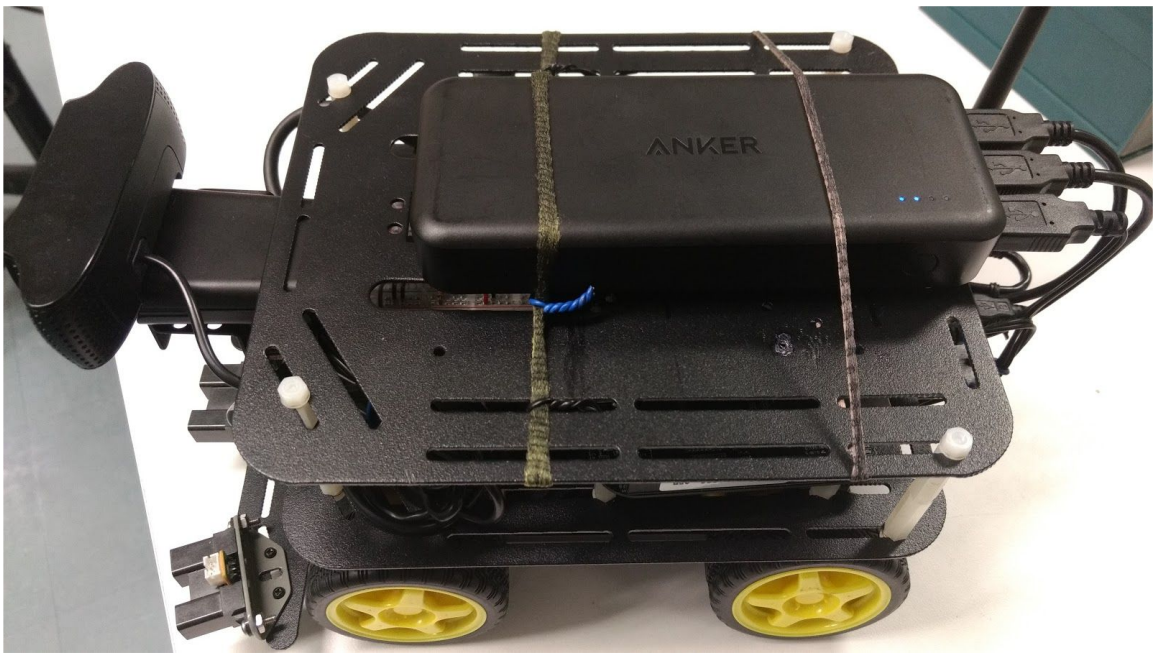
We chose this one due to its high capacity, because it has three USB ports (the amount we needed), because it's rechargeable, and most importantly because of its high amp output.

IMPORTANT: This power bank has a total amp output of 6V. This means that when using all three USB ports, each port is capable of delivering 2A. Many power banks advertise that their ports deliver 2.4A, but they have a low total amp output. This means that when using multiple ports, far less than 2.4A will actually be delivered. For example, consider a power bank which is capable of delivering 2.4A for each USB port but with a 4A total output. If we use all three ports, we will wind up getting a 4/3 = 1.33A output per cable. **In order for our wifi to be stable, we must be able to deliver at least 2A to the 5V barrel connector on the BeagleBone Black.** Make sure that the power bank used has enough total amp output.

Our USB ports are used as follows:

- One to power the BeagleBone Black with the USB to barrel power cable.
- One to power our main circuit (connected to the Atmega328p's)
- One to power our H-Bridges directly

-


ANKER

2.6 - Final Circuit

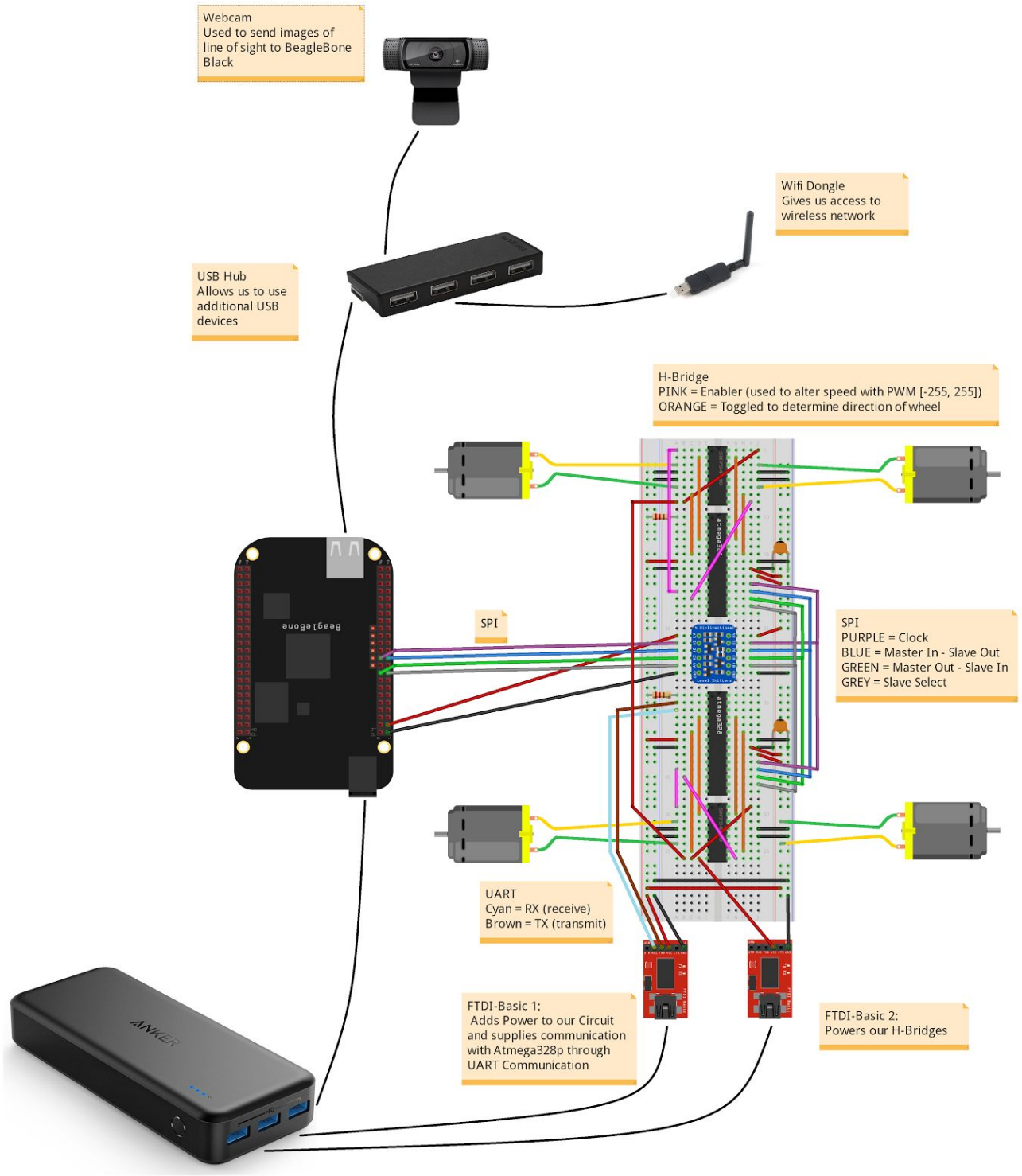


Figure 16

Notice that we used 2 Atmega328p's and 2 H-Bridges. The ones on the bottom are connected to the rear wheels, and the ones on the top are connected to front wheels. Each Atmega328p runs the same program and are connected to the same SPI bus.

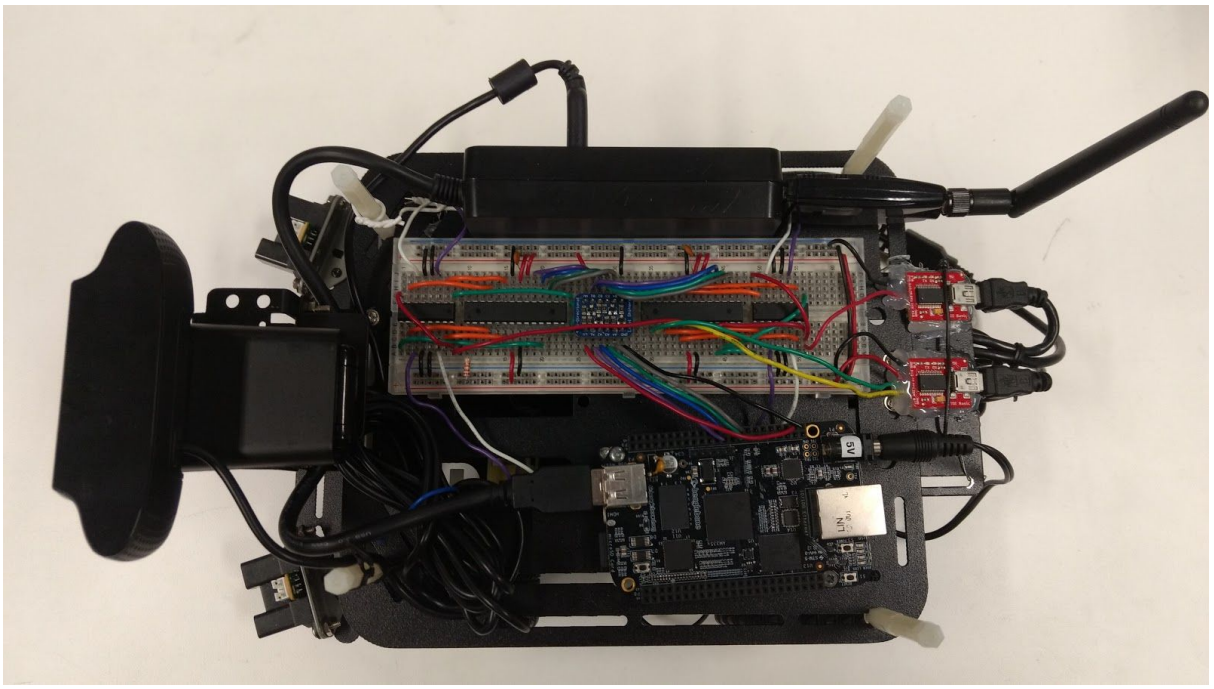


Figure 17

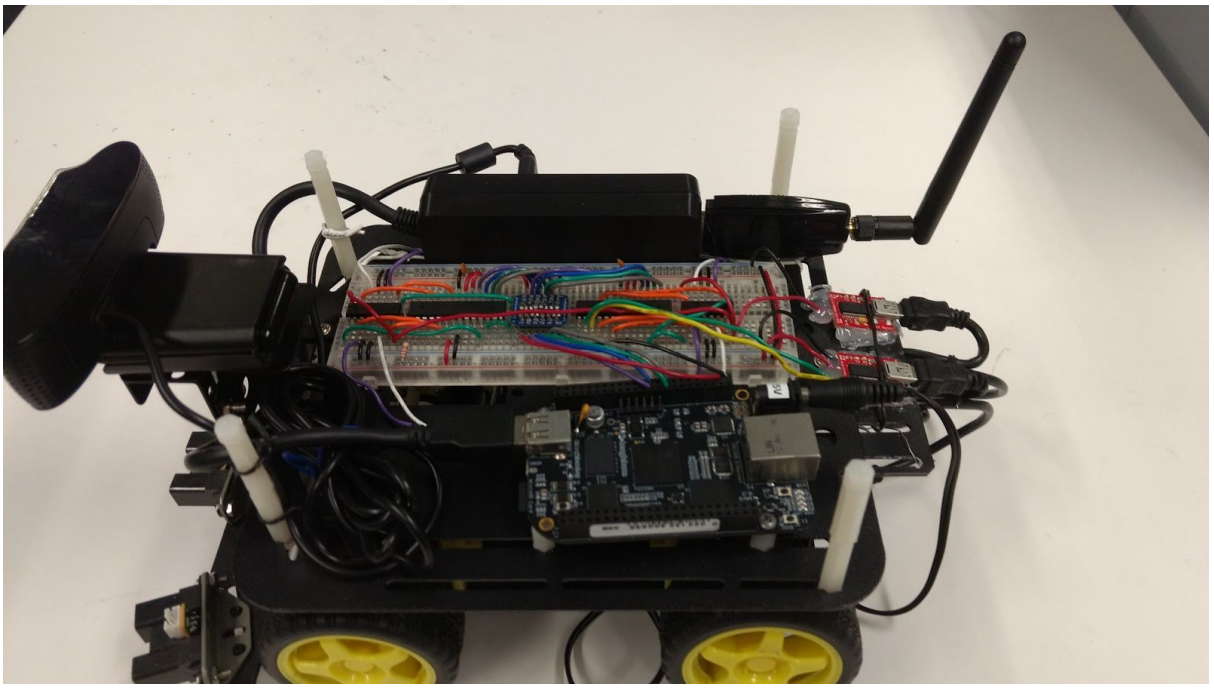


Figure 18

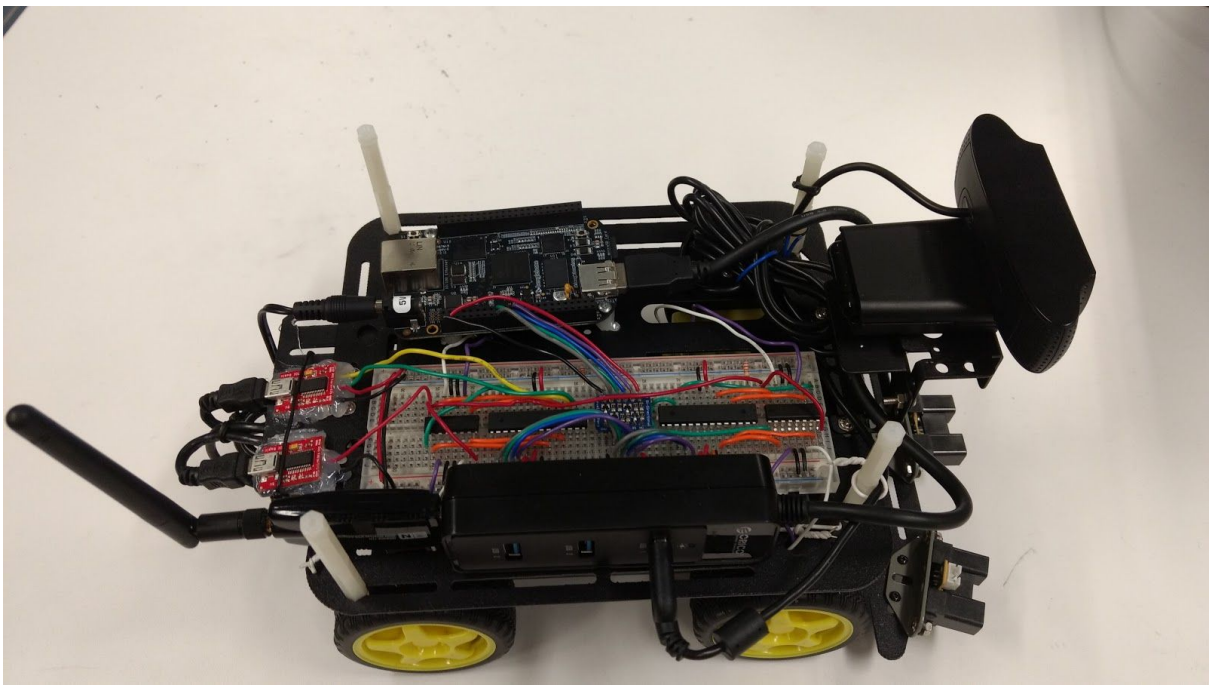


Figure 19

3 - Software

3.1 - Atmega328p

3.1.1 - Reading Data from BeagleBone Black

The Atmega328p is continuously receiving messages from the BeagleBone Black. The messages received must come in the format

```
x,x#
```

Where each *x* is an integer in the range [-255, 255]. The comma is used as a delimiter to separate the integers and the hashtag is used as a delimiter to know that it is the end of the specific message.

The first *x* is used to specify the speed of the left wheel and the second *x* is used to specify the speed of the right wheel. Negative values signify reverse and positive values signify forward with a zero signifying a complete stop. The absolute value of the number will determine how much power is delivered. For example:

- 255 is full speed in the forward direction
- 128 is approximately half speed in the forward direction
- -255 is full speed in the reverse direction
- 0 is stopped

This message is received as a character array, so the Atmega will parse this value and store the two provided integers into an integer array. It uses these values to signal to its corresponding H-Bridge how to power the wheel. It does this by:

- Setting the digital pins (PD2, PD4 and PC2, PC0) accordingly (*see section 2.3.2*)
- Setting the value of the enabler pins (1,2EN and 3,4EN) to the absolute value of the integer (*see section 2.3.3*)

3.1.2 - Writing to Console with UART

The Atmega328p also constantly writes to the console using UART. This was extremely helpful to us while creating the program running on the Atmega328p. It allows us to print messages received by the BeagleBone to the console, print the status of initialization steps to see if they worked, and to see how it processed the received messages (to see if the wheels are receiving the proper data), etc.

To see how to read the UART messages (*see section 4.3*).

3.1.3 - Flashing Atmega328p

In order to flash a program to the Atmega328p, we used the flashing device provided by Concordia University.

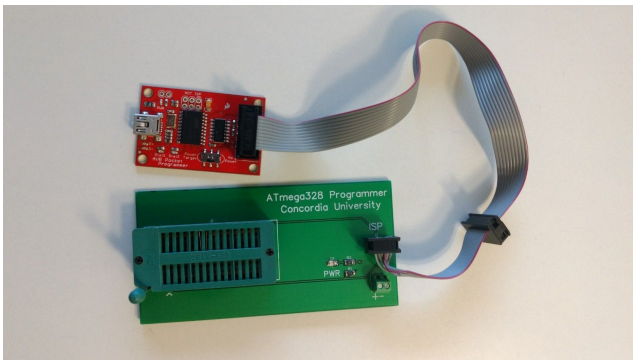


Figure 20

We must insert the Atmega328p into the device, with the groove closest to the lever, then pull the lever down. Now insert a *USB-to-mini-USB cable* from the device to your computer.

Figure 21

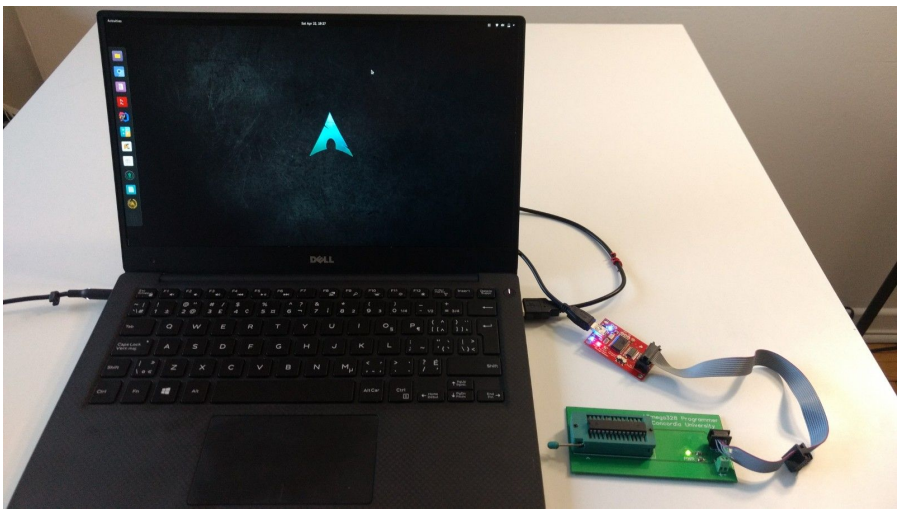


Figure 22

3.1.3.1 - Flashing Image on Linux

We created a function in Bash to flash a program to the Atmega328p in order to save time. It can be added to your .bashrc or run from a Bash file (flash.sh). In this tutorial, we will show you how to add it to your .bashrc.

Edit the .bashrc file in your home directory. If it doesn't already exist, create it.

```
$ vim ~/.bashrc
```

Now add the following function.

```
# Used to flash a C program to the Atmega328p
function flash() {
    # remove extension from arg
    filename=$(echo $@ | cut -d '.' -f 1)

    # create object and hex files
    avr-gcc -std=c11 -mmcu=atmega328 -O -o "$filename.o" $@
    avr-objcopy -O ihex $filename.o $filename.hex

    # flash hex file to atmega
    sudo avrdude -c usbtiny -p m328 -U flash:w:$filename.hex

    # remove object and hex files to keep dir clean
    rm $filename.o $filename.hex
}
```

In order for the changes to take place, run:

```
$ source ~/.bashrc
```

Now you can call the *flash* function as any other command from your terminal (similar to cd, ls, etc). To flash the Atm.c file, simply run:

```
$ flash /path/to/program.c
```

3.1.4 - Atm.c

This is the program ran by the Atmega328p which interprets data from BeagleBone Black and signals the H-Bridges to control the car.

```
#ifndef F_CPU
#define F_CPU 1000000UL
#endif
#define USART_BAUDRATE 4800
#define BAUD_PRESCALE (((F_CPU/(USART_BAUDRATE*16UL)))-1)

#include <avr/io.h>
#include <avr/interrupt.h>
#include <ctype.h>
#include <math.h>
```

```
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <util/delay.h>

enum direction {FORWARD, BACKWARD};

#define ACK 0x7E

//********************** Timer *********************************
void start_timer() {
    cli();
    TIMSK1 = (1 << TOIE1);
    TCCR1B |= ((0 << CS12) | (1 << CS11) | (0 << CS10));
    sei();
}

void stop_timer() {
    cli();
    TCCR1B &= ((0 << CS12) | (0 << CS11) | (0 << CS10));
    PORTC &= (0 << PC3); // make sure led is off when stopped
    sei();
}

// timer interrupt, if timer is running this will toggle the led when timer
counter overflows
ISR(TIMER1_OVF_vect) {
    cli();
    PORTC ^= (1 << PC3);
    sei();
}

/*
* sends char to serial monitor
*/
void sendChar(char c) {
    // Wait until buffer is empty
    while (!(UCSR0A & (1 << UDRE0))) {}
    // Return the data from the RX buffer
    UDR0 = c;
}
```

```
/*
* Send message to serial monitor
*/
void sendMessage(char msg[]) {
    for (int i = 0; msg[i]; i++) {
        sendChar(msg[i]);
        if (msg[i] == '#') {
            break;
        }
    }
}

//**************************  UART  ****************************
void UART_setup() {

    /*Set baud rate */
    UBRR0H  = (BAUD_PRESCALE >> 8);
    UBRR0L  = BAUD_PRESCALE;
    // Enable receiver and transmitter
    UCSR0B |= (1 << RXEN0) | (1 << TXEN0);
    // Set frame: 8data, 1 stp
    UCSR0C |= (1 << UCSZ01) | (1 << UCSZ00);

    sendMessage(">> Initializing UART ...\n");
}

/*
* Receives a character from serial monitor
*/
char receiveChar() {
    // Wait until data is received
    while (!(UCSR0A & (1 << RXC0))) {}
    // Return the data from the RX buffer
    return UDR0;
}

/*
* Receives a message from serial monitor
*/
void receiveMessage(char* message) {
    int index = 0;
    char c = receiveChar();
```

```
    // Characters allowed to be received
    while (isdigit(c) || c == ',' || c == '-') {
        message[index++] = c;
        c = receiveChar();
    }
    message[index] = '\0';
}


char* itoa2(int i, char b[]){
   char const digit[] = "0123456789";
   char* p = b;
   if(i<0){
       *p++ = '-';
       i *= -1;
   }
   int shifter = i;
   do{ //Move to where representation ends
       ++p;
       shifter = shifter/10;
   } while(shifter);
   *p = '\0';
   do{ //Move back, inserting digits as u go
       *--p = digit[i%10];
       i = i/10;
   } while(i);
   return b;
}

// WARNING: int should be < 1000
void sendInt(int num) {
    char num_str[4] = {' ', ' ', ' ', '\0'};
    itoa2(num, num_str);
    sendMessage(num_str);
    sendMessage("\n");
}

//***************** Control car ***********************

// initialize pwm for enablers
void initialize_pwm() {
    TCCR0A = (1 << WGM00) | (1 << WGM01) | (1 << COM0A1) | (1 << COM0B1);
    TCCR0B = (1 << CS00) | (1 << CS02);
```

```
}

void init_wheels() {

    sendMessage(">> Initializing wheels ...\n");

    // Set left wheels as output
    DDRD |= (1 << PD2);
    DDRD |= (1 << PD4);

    // Set right wheels as output
    DDRC |= (1 << PC2);
    DDRC |= (1 << PC0);

    // Set enabler left as output
    DDRD |= (1 << PD5);

    // Set enabler right as output
    DDRD |= (1 << PD6);

    initialize_pwm();
}

void power_left_wheel(enum direction dir, uint8_t duty) {
    if (duty < 0 || duty > 255) {
        return;
    }

    switch (dir) {
        case FORWARD:
            sendMessage("Powering left wheel forward by: ");
            sendInt(duty);
            PORTD |= (1 << PD2);
            PORTD &= ~(1 << PD4);
            break;
        case BACKWARD:
            sendMessage("Powering left wheel backward by: ");
            sendInt(duty);
            PORTD |= (1 << PD4);
            PORTD &= ~(1 << PD2);
            break;
        default:
            PORTD &= ~(1 << PD2);
```

```
            PORTD &= ~(1 << PD4);
    }

    OCR0B = duty;
}

void power_right_wheel(enum direction dir, uint8_t duty) {
    if (duty < 0 || duty > 255) {
        return;
    }

    switch (dir) {
        case FORWARD:
            sendMessage("Powering right wheel forward by: ");
            sendInt(duty);
            PORTC |= (1 << PC2);
            PORTC &= ~(1 << PC0);
            break;
        case BACKWARD:
            sendMessage("Powering right wheel backward by: ");
            sendInt(duty);
            PORTC |= (1 << PC0);
            PORTC &= ~(1 << PC2);
            break;
        default:
            PORTC &= ~(1 << PC2);
            PORTC &= ~(1 << PC0);
    }

    OCR0A = duty;
}

/*
 * Set the speed and direction of each wheel. +ive is forward, -ve is
 * backward
*/
void set_wheels(int left_wheel, int right_wheel){

    if (left_wheel < 0) {
        power_left_wheel(BACKWARD, -left_wheel);
    } else {
        power_left_wheel(FORWARD, left_wheel);
    }
```

```
    if (right_wheel < 0) {
        power_right_wheel(BACKWARD, -right_wheel);
    } else {
        power_right_wheel(FORWARD, right_wheel);
    }
}

//************************ SPI ****************************
// spi functions
void spi_init_slave (void) {
   sendMessage(">> Initializing slave\n");
   DDRB=(1 << 6);                                  //MISO as OUTPUT
   SPCR=(1 << SPE);                                //Enable SPI
}

//Function to send and receive data
unsigned char spi_tranceiver (unsigned char data) {
   SPDR = data;                            //Load data into buffer
   while(!(SPSR & (1<<SPIF) ));           //Wait until transmission complete
   return(SPDR);                          //Return received data
}

void *get_message(char *message) {
    int index = 0;
    char c = spi_tranceiver(ACK);
    while (c != '#') {
        message[index++] = c;
        c = spi_tranceiver(ACK);
    }
    message[index] = '\0';
}

//************************* Text parsing **********************
/*
* Char to int
*/
int ctoi(char d) {
    char str[2];

    str[0] = d;
    str[1] = '\0';
    return (int) strtol(str, NULL, 10);
```

```
}

/*
* Parses a string of integers
*/
int parseInt(char* integer_str) {
   int sign = 1;
   int length = strlen(integer_str);
   int i = 0;
   if (integer_str[0] == '-') {
       sign = -1;
       i = 1;
   }
   int input_int = 0;
   if (length >= 3) {
       input_int = 1;
   }
   for (i; i < length; i++) {
       input_int += ctoi(integer_str[i]) * pow(10, length - 1 - i);
   }
   return (input_int*sign);
}

void parse_message(char *msg, int *nums) {
    int msg_index = 0;
    int num_index = 0;
    int k = 0;
    char c;
    char num[5];

    do {
        c = (char) msg[msg_index++];
        if (c == ',' || c == '#') {
            num[num_index] = '\0';
            nums[k++] = parseInt(num);
            num[0] = '\0'; num[1] = '\0', num[2] = '\0';
            num_index = 0;
        } else {
            num[num_index++] = c;
        }
    } while (c != '#');
}
```

```
int main(void) {

   UART_setup();
   init_wheels();
   spi_init_slave();

   unsigned char data, msg[16];
   while(1) {

        sendMessage("\nAwaiting message\n");
        int index = 0;

         do {
             data = spi_tranceiver(ACK);
             msg[index++] = data;
         } while (data != '#');

         sendMessage("Received message: ");
         sendMessage(msg);
         sendChar('\n');

         int nums[2] = {0, 0};
         parse_message(msg, nums);
         set_wheels(nums[0], nums[1]);
   }
}
```

3.2 - BeagleBone Black

3.2.1 - Image Used

We tried many different Linux images to run on the BeagleBone Black. We discovered many differences between them. How they behaved, the stability of the wifi, the file system, how to enable device trees, how to setup the wifi, the /etc/network/interfaces file, and the uEnv.txt file differed depending on which version of Debian and which version of the Linux kernel we were using.

For our final deliverable, we decided to go with the most recent non-GUI image:

Debian 8.7 Jessie IoT (non-GUI) for BeagleBone via microSD card.

This image can be found at:

https://beagleboard.org/latest-images

3.2.2 - Flashing the Image

After we have downloaded the image file, we must flash it to an SD card.

3.2.2.1 - Flashing on Linux

Plug the SD card into your machine and run the command:

```
$ sudo fdisk -l
```

This command lists the hard drives available on your system (similar to C:///, D:///, etc. on Windows).

If you only have one hard drive on your system, you should see output such as:

```
Disk /dev/sda: 119.2 GiB, 128035676160 bytes, 250069680 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 4096 bytes
Disklabel type: gpt
Disk identifier: E6A3F0A1-4C60-4DC4-8D33-77419A6E5396

Device         Start       End   Sectors   Size Type
/dev/sda1       2048    411647    409600   200M EFI System
/dev/sda2     411648    935935    524288   256M Linux filesystem
/dev/sda3     935936 234020863 233084928 111.1G Linux filesystem
/dev/sda4  234020864 250069646  16048783   7.7G Linux swap


Disk /dev/mmcblk0: 3.7 GiB, 3963617280 bytes, 7741440 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x0468bf7b

Device         Boot  Start     End Sectors  Size Id Type
/dev/mmcblk0p1 *      2048  198655  196608   96M  e W95 FAT16 (LBA)
/dev/mmcblk0p2      198656 3481599 3282944  1.6G  e W95 FAT16 (LBA)
```

`/dev/sda` is the hard drive on your machine. All of the `/dev/sdaX,` where X is a number from 1 to 4 represent the partitions of your disk.

If you have another disk, it will be labeled `/dev/sdb,` and the next will be labeled `/dev/sdc,` and so on.

Our SD card is labeled `/dev/mmcblk0,` it takes on this different naming convention to signify that it is an SD card and not a conventional hard drive or solid state drive.

To flash the image to the SD card, we can run the following command:

```
$ dd if=/path/to/your/img_file.img of=/dev/mmcblk0p2
```

The *"if"* stands for input file and *"of"* stands for output file. Make sure you change the path of the input file to the Debian image that you downloaded from BeagleBoard.org.

This may take a few minutes so be patient. It is normal not to see any output while the image is flashing to the SD card and cancelling it may corrupt your disk. Once it is finished, there should be some output telling you that the image was written successfully flashed to the card.

Make sure the BeagleBone Black is turned off and then insert the SD card. Hold the boot button (*see figure 9 in section 2.4*) and then power on the board. Hold the boot button until all four LEDs on the BeagleBone turn on, then release it.

3.2.3 - SSH into the BeagleBone

In order to SSH into the BeagleBone (since it does not yet have wifi enabled), we must use a USB cord. Not the USB to barrel cord described earlier, but a USB to mini-USB cord.

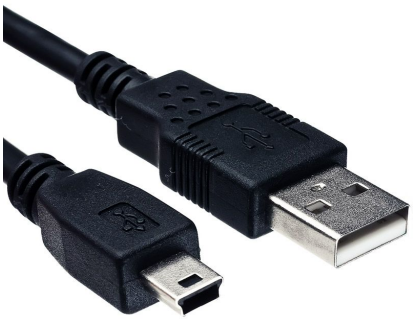


Figure 23

Plug the USB end into your computer, and the mini-USB end into your BeagleBone Black. Now we can ssh in.

3.2.3.1 - SSH using Linux

Enter the command:

```
$ ssh debian@192.168.7.2
```

This specifies that you are logging in as the default username "*debian*", and *192.168.7.2* is the IP address of the device connected over the USB cable (your BeagleBone Black in this case).

It will then prompt you for your password, type this in and press enter. The default password is *"temppwd"* without the quotation marks.

You may now be asked if you wish to add the key to the BeagleBone Black. Say yes and you will then be SSH'd into the BeagleBone Black.

3.2.4 - Enabling Wifi on Kernel 4.x

Enabling Wifi on Kernel 3.x is discussed in a separate section.

To verify that the BeagleBone Black is detecting the wifi dongle, run the *ifconfig* command.

This command will show us available network interfaces, such as ethernet, wifi cards, etc. The device name will be listed on the left and further information on the device will be displayed on the right. If the BeagleBone is detecting the wifi dongle there should be a device named *"wlan0"*.

```
$ ifconfig

eth0      Link encap:Ethernet  HWaddr c8:a0:30:b5:4d:35
          inet addr:192.168.0.149  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::caa0:30ff:feb5:4d35/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST DYNAMIC  MTU:1500  Metric:1
          RX packets:1001 errors:0 dropped:268 overruns:0 frame:0
          TX packets:191 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:69298 (67.6 KiB)  TX bytes:26145 (25.5 KiB)
          Interrupt:173

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:160 errors:0 dropped:0 overruns:0 frame:0
          TX packets:160 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:12960 (12.6 KiB)  TX bytes:12960 (12.6 KiB)

usb0      Link encap:Ethernet  HWaddr c8:a0:30:b5:4d:37
          inet addr:192.168.7.2  Bcast:192.168.7.3  Mask:255.255.255.252
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

usb1      Link encap:Ethernet  HWaddr c8:a0:30:b5:4d:3a
          inet addr:192.168.6.2  Bcast:192.168.6.3  Mask:255.255.255.252
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

wlan0     Link encap:Ethernet  HWaddr 00:0b:81:93:a3:bc
          UP BROADCAST MULTICAST DYNAMIC  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

We can see *wlan0* but we still have to configure it. Notice that *wlan0* does not yet have an IP address. To configure the wifi, run the program *connman* by entering the command:

```
$ connmanctl

connmanctl>
```

We can then see that the prompt changes. This signifies that we are running connman.

Now ensure that wifi is enabled:

```
connmanctl> enable wifi

Error wifi: Already enabled
```

If the wifi is already enabled, it will let us know. Now scan for available networks:

```
connmanctl> scan wifi

Scan completed for wifi
```

Note that this step usually takes some time. You must wait for the output to appear. Now display the networks found:

```
connmanctl> services

SSID1         wifi_000b8193a3bc_42454c4c313832_managed_psk
Home Wifi     wifi_000b8193a3bc_42454c4c333539_managed_psk
Neighbor      wifi_000b8193a3bc_42454c4c333834_managed_psk
Starbucks     wifi_000b8193a3bc_42454c4c333932_managed_psk
```

This will list all available wireless networks in your area. Notice that the input will come in the format:

Name Code

We want to connect to the network *"Home Wifi"*. In order to do this, highlight the code and copy it to your clipboard. So in this case the code will be :

"`wifi_000b8193a3bc_42454c4c333539_managed_psk`*"*

Now, register the agent and connect to your network by pasting the code.

```
connmanctl> agent on

Agent registered

connmanctl> connect wifi_000b8193a3bc_42454c4c333539_managed_psk

Passphrase? xxxxxxx

Connected wifi_000b8193a3bc_42454c4c333539_managed_psk
```

Note that we have to enter the password to the wireless network that you are connecting to (the wifi password).

This may take a moment, but then you should be prompted with a success message. You should now have wifi on your Beaglebone Black!

Now it's time to test your connection. Enter *quit* to exit *Connman* and check *ifconfig* again.

```
connmanctl> quit

$ ifconfig

eth0      Link encap:Ethernet  HWaddr c8:a0:30:b5:4d:35
          inet addr:192.168.0.149  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::caa0:30ff:feb5:4d35/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST DYNAMIC  MTU:1500  Metric:1
          RX packets:3944 errors:0 dropped:961 overruns:0 frame:0
          TX packets:802 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:286346 (279.6 KiB)  TX bytes:100089 (97.7 KiB)
          Interrupt:173

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:170 errors:0 dropped:0 overruns:0 frame:0
          TX packets:170 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:13832 (13.5 KiB)  TX bytes:13832 (13.5 KiB)

usb0      Link encap:Ethernet  HWaddr c8:a0:30:b5:4d:37
          inet addr:192.168.7.2  Bcast:192.168.7.3  Mask:255.255.255.252
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

usb1      Link encap:Ethernet  HWaddr c8:a0:30:b5:4d:3a
          inet addr:192.168.6.2  Bcast:192.168.6.3  Mask:255.255.255.252
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

wlan0     Link encap:Ethernet  HWaddr 00:0b:81:93:a3:bc
```

```
          inet addr:192.168.0.196  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::20b:81ff:fe93:a3bc/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST DYNAMIC  MTU:1500  Metric:1
          RX packets:73 errors:0 dropped:15 overruns:0 frame:0
          TX packets:85 errors:0 dropped:1 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:6206 (6.0 KiB)  TX bytes:16639 (16.2 KiB)
```

We can see that our *wlan0* now has an IP address associated with it. Finally, ping a website to make sure you receive a response.

```
$ ping google.com

PING google.com (172.217.4.238) 56(84) bytes of data.
64 bytes from ord30s31-in-f238.1e100.net (172.217.4.238): icmp_seq=1 ttl=56
time=31.0 ms

64 bytes from ord30s31-in-f238.1e100.net (172.217.4.238): icmp_seq=2 ttl=56
time=33.1 ms

64 bytes from ord30s31-in-f238.1e100.net (172.217.4.238): icmp_seq=3 ttl=56
time=30.5 ms
```

Note: Press ctrl + c to exit the ping process.

We are sending a message to Google and receiving a response approximately every 30 ms. This means that our wifi is working.

If we wish to now make our BeagleBone completely wireless, we can SSH into the new IP address (highlighted above) from any machine on the same network. So in our example:

```
$ ssh debian@192.168.0.196
```

Note: your IP address may be different than the one listed above. Use the IP address we highlighted under ifconfig.

If you have SSH'd into the BeagleBone Black with your new IP address and assuming you are also powering your BeagleBone Black through the barrel port, you can safely remove the USB-to-mini-USB cable. Your car is now wireless!

3.2.5 - Setting Up Linux Environment

Now that we have wifi on our system, we can clone our repo from https://github.com/amirbawab/mr_robot.git

Install packages and then clone the repo under /home/debian:

```
$ sudo apt-get install git
```

```
$ git clone https://github.com/ml-davis/self_driving_car.git
```

Install required packages including opencv:

```
$ apt-get update
$ apt-get install cmake libgtk2.0-dev libavcodec-dev
libavformat-dev libswscale-dev libopencv-dev
```

OPTIONAL:
Now we can copy some of the Linux configuration files to our home directory. This will make our working environment a little more sane. Navigate into the repo and run the following commands:

```
$ cd ~/self_driving_car/linux_config

$ cp -t ~ .bashrc .inputrc .vimrc .profile

$ cp -r .vim ~

$ source ~/.bashrc
```

3.2.6 - Configuring SPI on Kernel 4.x

The BeagleBone Black has two different modes. SPI1, which uses pins 28, 29, 30, 31 and SPI0 which uses pins 17, 17, 21, 22. We chose to use SPI0.

There are two different ways to enable SPI.

3.2.6.1 - Loading Device Tree Overlays

Device tree overlays can be used to set up your BeagleBone Black for certain functionalities (*see section 4.4*). In order for us to get SPI to work, we must load the device tree responsible for enabling SPI0.

In order to see the device tree overlays pre-loaded onto the BeagleBone Black, list the files under the /lib/firmware directory.

```
~ ls /lib/firmware
```

All of the files with the .dtbo extension are device tree overlays. We can see a lot of them already loaded onto the board. The one we are interested in is `BB-SPIDEV0-00A0.dtbo`

We need to add this device to the *slots* file in order to enable it. To see which devices are currently loaded, run:

```
$ more $SLOTS

0: PF----  -1
1: PF----  -1
2: PF----  -1
3: PF----  -1
4: P-O-L-   0 Override Board Name,00A0,Override Manuf,cape-universaln
```

NOTE: If you did not add the .profile file (see section 3.2.5), you will have to write the path to the slots file manually, substitute $SLOTS for "/sys/devices/platform/bone_capemgr" so the command becomes: "$ more /sys/devices/platform/bone_capemgr/slots"

We can see that the cape-universaln device is loaded by default and we cannot enable SPI until we disable this device. There are two different ways to do this.

One way is to echo -4 to our $SLOTS, however, we had lots of problems with this method and experienced it breaking our machine. Many others online have reported the same thing.

The other way, which we recommend, is to disable the device by editing the `/boot/uEnv.txt` file.

```
$ vim /boot/uEnv.txt
```

In this file we should see a line that says:

```
cmdline=coherent_pool=1M net.ifnames=0 quiet cape_universal=enable
```

On our machine, it's on line 49. What we need to do is comment out the part which enables cape_universal. So add a hashtag before cape_universal=enable like this:

```
cmdline=coherent_pool=1M net.ifnames=0 quiet
#cape_universal=enable
```

Now reboot the BeagleBone Black and SSH back in. Make sure the device is now disabled by checking the slots file again:

```
$ more $SLOTS

0: PF----  -1
1: PF----  -1
2: PF----  -1
3: PF----  -1
```

We can see that it's no longer loaded. Now in order to load SPI0, we must load the `BB-SPIDEV0-00A0.dtbo` file mentioned earlier. This is done by running the following:

```
$ sudo sh -c "echo BB-SPIDEV0 > $SLOTS"
```

Now if we view the $SLOTS file one more time, we should see that it is loaded.

```
$ more $SLOTS

0: PF----  -1
1: PF----  -1
2: PF----  -1
3: PF----  -1
4: P-O-L-   0 Override Board Name,00A0,Override Manuf,BB-SPIDEV0
```

3.2.6.2 - Using config-pin

The other method to enable SPI is to use the program *config-pin.* Contrary to the previous method, this time we must make sure that cape_universal is enabled. Therefore, if you commented it out in /boot/uEnv.txt, you will need to uncomment it and reboot your device (*see section 3.2.6.1*).

Once this is done, verify that cape_universal is in fact loaded.

```
$ more $SLOTS

0: PF----  -1
1: PF----  -1
2: PF----  -1
3: PF----  -1
4: P-O-L-   0 Override Board Name,00A0,Override Manuf,cape-universaln
```

If you wish to check the modes available for each pin, enter the command *config-pin -l <pin>*. The *-l* is an argument which stands for *list.* It lists all available modes. For example, if we wish to see the modes available for pin P9-17:

```
$ config-pin -l P9.17

default gpio gpio_pu gpio_pd spi i2c pwm
```

This shows all modes that this pin is capable of providing. Remember, that we wish to enable pins P9.17, P9.18, P9.21, P9.22, for SPI0. You can verify these all have spi mode by running the previous command on them.

In order to set the pins to SPI mode, enter:

```
$ config-pin P9.17 spi
$ config-pin P9.18 spi
$ config-pin P9.21 spi
```

```
$ config-pin P9.22 spi
```

This should set them to SPI mode. If we want to verify that it worked, run "*config-pin -q <pin>"* to make sure it worked. This time the *-q* argument stands for query.

```
$ config-pin -q P9.17
P9_17 Mode: spi

$ config-pin -q P9.18
P9_18 Mode: spi

$ config-pin -q P9.21
P9_21 Mode: spi

$ config-pin -q P9.22
P9_22 Mode: spi
```

We can see all four of our pins are now set to SPI mode.

3.2.7 - Write.c for Kernel 4.x

This file is a modified version of a file which is meant to test the functionality of SPI. The original can be found under the infamous Linux creator Linus Torvalds' repo at:

https://github.com/torvalds/linux/blob/master/tools/spi/spidev_test.c

The modified version can be found under our repo at:

https://github.com/amirbawab/mr_robot/blob/master/tools/control/kernel4/write.c

Write.c

```
/*
 * SPI testing utility (using spidev driver)
 *
 * Copyright (c) 2007  MontaVista Software, Inc.
 * Copyright (c) 2007  Anton Vorontsov <avorontsov@ru.mvista.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License.
 *
 * Cross-compile with cross-gcc -I/path/to/cross-kernel/include
 */

#include <stdint.h>
#include <unistd.h>
#include <stdio.h>
```

```c
#include <stdlib.h>
#include <string.h>
#include <getopt.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <linux/types.h>
#include <linux/spi/spidev.h>

#define ARRAY_SIZE(a) (sizeof(a) / sizeof((a)[0]))

static void pabort(const char *s)
{
	perror(s);
	abort();
}

static const char *device = "/dev/spidev1.0";
static uint32_t mode;
static uint8_t bits = 8;
static uint32_t speed = 10000;
static uint16_t delay;
static int verbose;

uint8_t default_tx[] = {
	0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
	0x40, 0x00, 0x00, 0x00, 0x00, 0x95,
	0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
	0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
	0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
	0xF0, 0x0D,
};

uint8_t default_rx[ARRAY_SIZE(default_tx)] = {0, };
char *input_tx;

static void hex_dump(const void *src, size_t length, size_t line_size, char *prefix)
{
	int i = 0;
	const unsigned char *address = src;
	const unsigned char *line = address;
	unsigned char c;

	printf("%s | ", prefix);
	while (length-- > 0) {
		printf("%02X ", *address++);
		if (!(++i % line_size) || (length == 0 && i % line_size)) {
			if (length == 0) {
				while (i++ % line_size)
```

```
                                printf("__ ");
                        }
                        printf(" | ");  /* right close */
                        while (line < address) {
                                c = *line++;
                                printf("%c", (c < 33 || c == 255) ? 0x2E : c);
                        }
                        printf("\n");
                        if (length > 0)
                                printf("%s | ", prefix);
                }
        }
}

/*
 *  Unescape - process hexadecimal escape character
 *      converts shell input "\x23" -> 0x23
 */
static int unescape(char *_dst, char *_src, size_t len)
{
        int ret = 0;
        char *src = _src;
        char *dst = _dst;
        unsigned int ch;

        while (*src) {
                if (*src == '\\' && *(src+1) == 'x') {
                        sscanf(src + 2, "%2x", &ch);
                        src += 4;
                        *dst++ = (unsigned char)ch;
                } else {
                        *dst++ = *src++;
                }
                ret++;
        }
        return ret;
}

static void transfer(int fd, uint8_t const *tx, uint8_t const *rx, size_t len)
{
        int ret;

        struct spi_ioc_transfer tr = {
                .tx_buf = (unsigned long)tx,
                .rx_buf = (unsigned long)rx,
                .len = len,
                .delay_usecs = delay,
                .speed_hz = speed,
```

```
                .bits_per_word = bits,
        };

        if (mode & SPI_TX_QUAD)
                tr.tx_nbits = 4;
        else if (mode & SPI_TX_DUAL)
                tr.tx_nbits = 2;
        if (mode & SPI_RX_QUAD)
                tr.rx_nbits = 4;
        else if (mode & SPI_RX_DUAL)
                tr.rx_nbits = 2;
        if (!(mode & SPI_LOOP)) {
                if (mode & (SPI_TX_QUAD | SPI_TX_DUAL))
                        tr.rx_buf = 0;
                else if (mode & (SPI_RX_QUAD | SPI_RX_DUAL))
                        tr.tx_buf = 0;
        }

        ret = ioctl(fd, SPI_IOC_MESSAGE(1), &tr);
        if (ret < 1)
                pabort("can't send spi message");

        //if (verbose)
        //      hex_dump(tx, len, 32, "TX");
        //hex_dump(rx, len, 32, "RX");
}

int main(int argc, char *argv[])
{
        int ret = 0;
        int fd;
        uint8_t *tx;
        uint8_t *rx;
        int size;

        fd = open(device, O_RDWR);
        if (fd < 0)
                pabort("can't open device");

        /*
         * spi mode
         */
        ret = ioctl(fd, SPI_IOC_WR_MODE32, &mode);
        if (ret == -1)
                pabort("can't set spi mode");

        ret = ioctl(fd, SPI_IOC_RD_MODE32, &mode);
        if (ret == -1)
```

```
                pabort("can't get spi mode");

        /*
         * bits per word
         */
        ret = ioctl(fd, SPI_IOC_WR_BITS_PER_WORD, &bits);
        if (ret == -1)
                pabort("can't set bits per word");

        ret = ioctl(fd, SPI_IOC_RD_BITS_PER_WORD, &bits);
        if (ret == -1)
                pabort("can't get bits per word");

        /*
         * max speed hz
         */
        ret = ioctl(fd, SPI_IOC_WR_MAX_SPEED_HZ, &speed);
        if (ret == -1)
                pabort("can't set max speed hz");

        ret = ioctl(fd, SPI_IOC_RD_MAX_SPEED_HZ, &speed);
        if (ret == -1)
                pabort("can't get max speed hz");

        //printf("spi mode: 0x%x\n", mode);
        //printf("bits per word: %d\n", bits);
        //printf("max speed: %d Hz (%d KHz)\n", speed, speed/1000);

        input_tx = argv[1];
        if (input_tx) {
                size = strlen(input_tx+1);
                tx = malloc(size);
                rx = malloc(size);
                size = unescape((char *)tx, input_tx, size);
                transfer(fd, tx, rx, size);
                free(rx);
                free(tx);
        } else {
                transfer(fd, default_tx, default_rx, sizeof(default_tx));
        }

        close(fd);

        return ret;
}
```

To compile this file, navigate to the kernel directory directory and run the build file:

```
$ ./build.sh
```

The `write` file located in the `control` directory (parent of the kernel directory) must point to the correct generated executable. To regenerate that `write` file, run from the `control` directory:

```
$ ln -s spi/Write.c Write
```

All dependent program should be ready to execute this file now.

3.2.8 - Tracking Ball

Now that SPI has been enabled on the Beaglebone, we can compile and run the OpenCV program, ocv.cpp. This is a very powerful C++ program which is responsible for three main things:

1) Querying the camera and receiving an input stream of images
2) Processing these images with OpenCV to determine location of ball.
3) Calling our write program to send navigation info to Atmega328p
4) Writing information to files to be used by our graphical user interface

3.2.8.1 - Camera Used

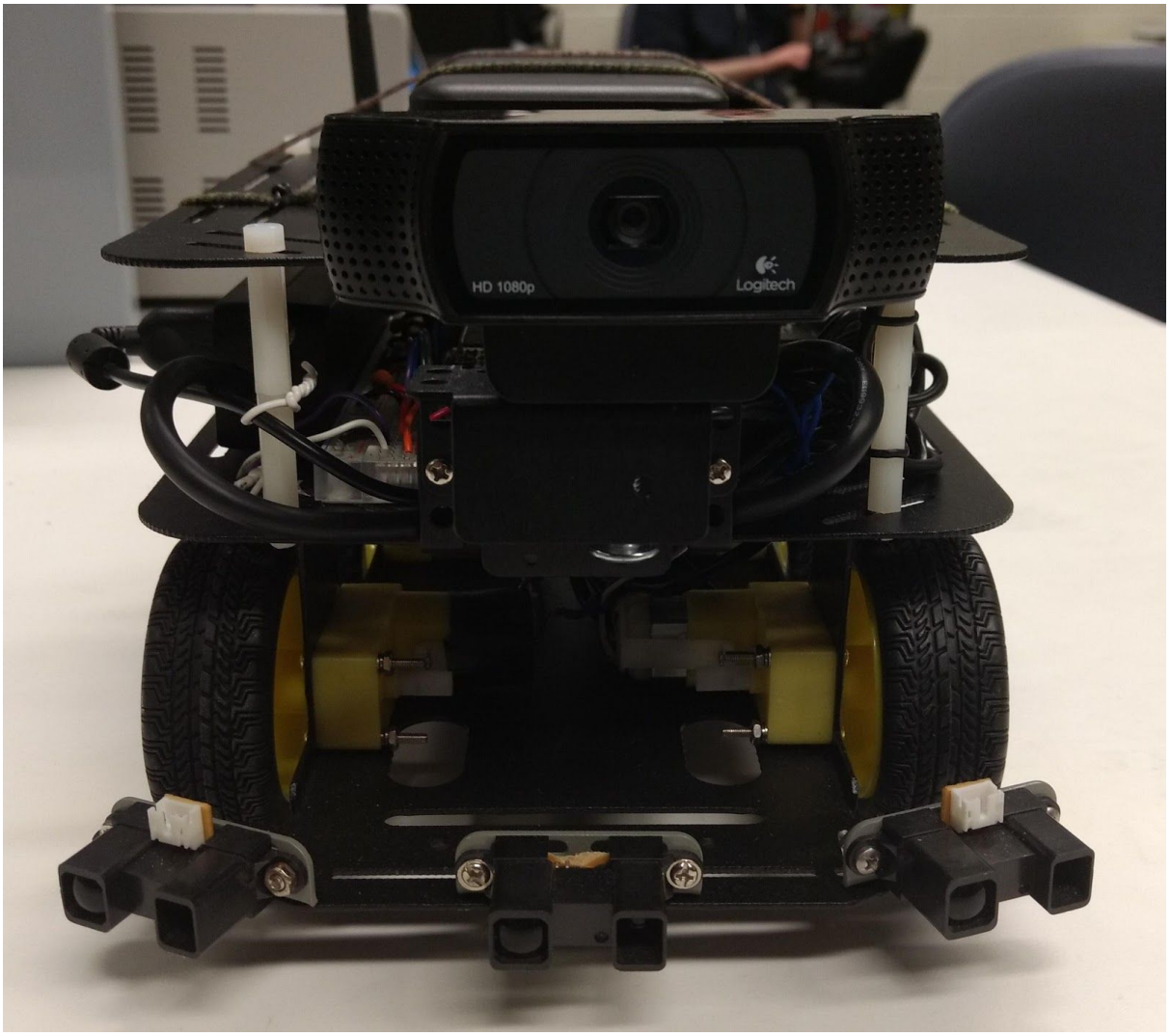


Figure 24

The camera we used as the eyes of our project is the Logitech C920 webcam. It is capable of recording high definition video in 1920x1080 and uses the H.264 video compression standard. This is significant since it reduces the latency of processing the images in our Beaglebone Black. We mounted it to the front of the car so that it could have a clear of its line of sight.

In order to effectively and efficiently detect our target object, we need to choose a good resolution to use for the images taken by our camera. If the resolution was too high, the latency of processing the image was increased and created a long delay from the event happening and it showing up in our program. If the resolution was too low, the program had a hard time detecting the target object. Through experimentation, we found that using a resolution of **320x180** gave us the most desirable results. We then send these images to the Beaglebone approximately five times per second.

3.2.8.2 - Navigation

Navigation of the car in the direction of the target relied on two main variables: distance between the car and the tracked object and the angle at which the object is from the camera.

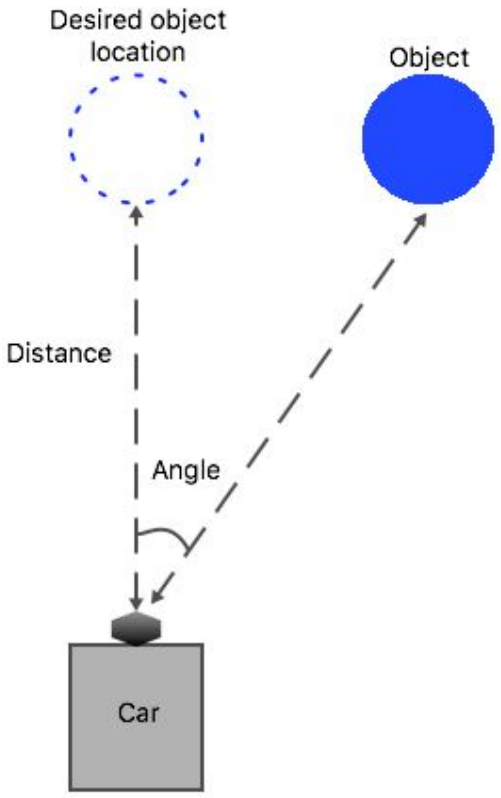


Figure 25

The goal of the car is to center the object in the middle of the field of view. In cases where the object is out of the field of view, the car will go in reverse to gain more insights on the location of the object. Once the location is identified the car will rotate based on the angle, as shown above. Rotation occurs by halting the front wheel on the side of the rotation and accelerating the other three wheels. For example, to make a right turn, the front right wheel will stop rotating to form an axis of rotation while the other three wheels continue spinning to force a rotation of the body of the car. Upon rotating the car, the car will stop for a second to refresh the images in order to make proper decision on where to go next.

3.2.8.3 - Calibration Process

In order to be able to infer the object distance and angle from an image, the camera had to be calibrated. Using objects with a known diameter, and distance from the camera, we were able to calculate the focal length of the camera in pixels using the following formula:
Add image

The camera we used as the eyes of our project is the Logitech C920 webcam. It is capable of recording high definition video in 1920x1080 and uses the H.264 video compression standard. This is significant since it reduces the latency of processing the images in our Beaglebone Black. We

mounted it to the front of the car so that it could have a clear of its line of sight.

In order to effectively and efficiently detect our target object, we need to choose a good resolution to use for the images taken by our camera. If the resolution was too high, the latency of processing the image was increased and created a long delay from the event happening and it showing up in our program. If the resolution was too low, the program had a hard time detecting the target object. Through experimentation, we found that using a resolution of 320x180 gave us the most desirable results. We then send these images to the Beaglebone approximately five times per second.
Add image

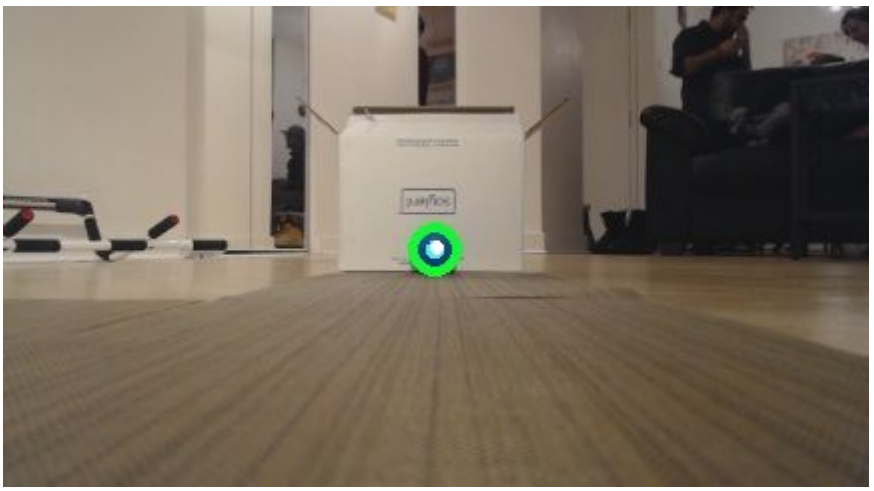
The camera we used as the eyes of our project is the Logitech C920 webcam. It is capable of recording high definition video in 1920x1080 and uses the H.264 video compression standard. This is significant since it reduces the latency of processing the images in our Beaglebone Black. We mounted it to the front of the car so that it could have a clear of its line of sight.

In order to effectively and efficiently detect our target object, we need to choose a good resolution to use for the images taken by our camera. If the resolution was too high, the latency of processing the image was increased and created a long delay from the event happening and it showing up in our program. If the resolution was too low, the program had a hard time detecting the target object. Through experimentation, we found that using a resolution of 320x180 gave us the most desirable results. We then send these images to the Beaglebone approximately five times per second.

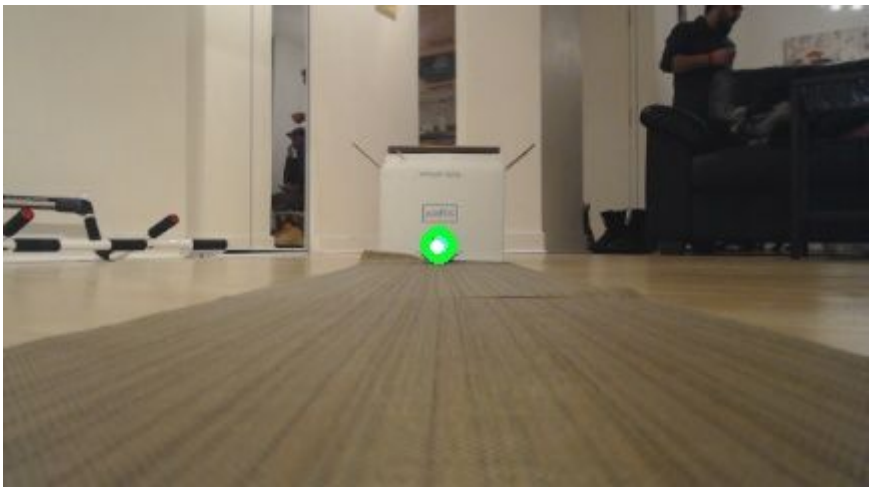
Focal Length = (Distance * Digital_Width) / Real_Width

The focal length is later used to infer the real distance of any object from the camera by measuring its digital diameter.

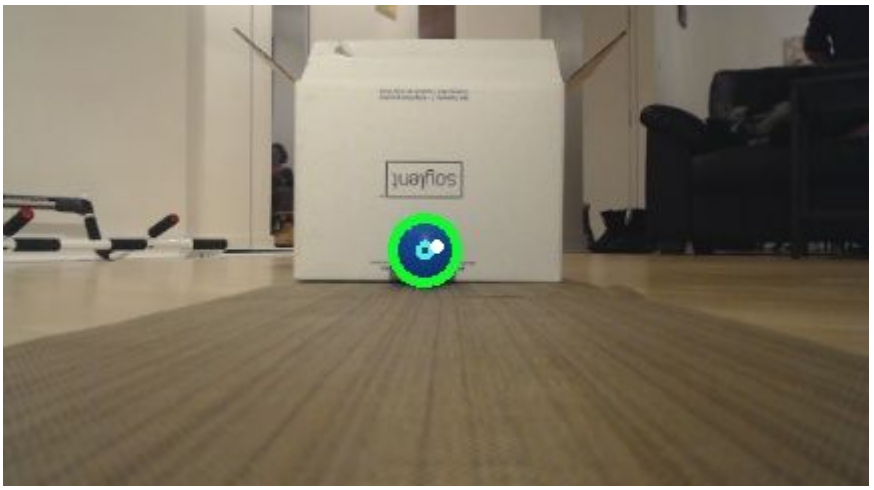
Object at 45.3 cm from camera



Object at 90.6 cm



Object at 135.9 cm



Object at 60.3 cm

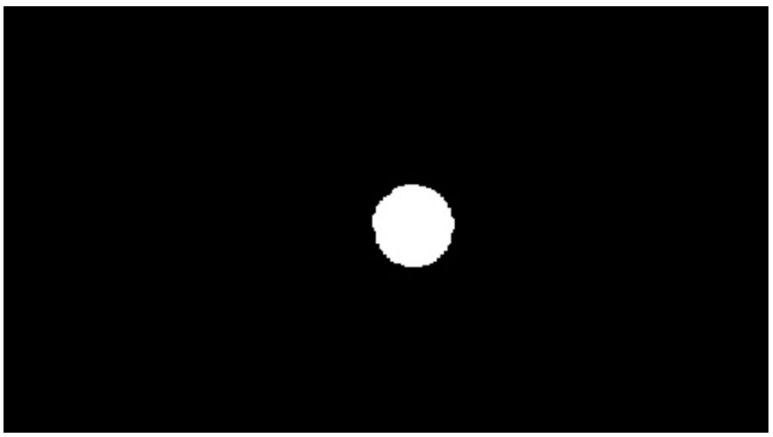
Object at 23 cm

The focal length was obtained from every image from the above using the known distance, the measured digital diameter and the real diameter of 6.5cm, then the result was averaged:

Real Distance (cm)	Digital Diameter (px)	Focal Length (px)
45.3	49	341.492
90.6	26	362.4
135.9	17	355.4307
60.3	35	324.692
23	88	311.38
Average focal length		**339.079**

3.2.8.4 - Target Tracking

For every image captured from the camera stream, our OpenCV program changes the hue, saturation and brightness value of the image until it brings out the areas of the target color. In a copy image, these areas will be colored in white, while all the rest of the image is black, the copy image would look like as such:



Blue ball in a binarized image

From the binarized image, the moments will be calculated and used to locate the centroids (x_center, y_center) of the object of interest. Also the contour of the object will be traced in green, as seen in images above, and an edge point on the contour will be used to find the digital diameter of the ball, which is used to estimate the distance of the object from the camera.

3.2.8.5 - OCV.cpp

```
/**
* OpenCV (OCV) - Video capture on BBB
*
* GOAL:
* This program takes continuous images from the host camera and detect a
circular
* object of specific color. The color and other configuration are provided
as
* input argument to the program.
*
* INPUT:
* The program takes two arguments:
* - The video device node (0/1)
* - The object color (blue/red)
*
* OUTPUT:
* On each frame captured by the program, three files are exported into the
* host filesystem (apache directory specifically):
* - A colored image with a green circular around the circular object
* - A binarized image of the colored image (previous bullet)
* - A text file containing information about the object captured
* In addition to storing images, at each frame iteration, the program
executes a shell script
```

```
* to communicate information about the frame with the ATMEGA328.
*
* HOST REQUIREMENTS:
* The following packages are required to be installed on the host:
* - NodeJS server
* - OpenSSH server
**/

#include <fstream>
#include <iostream>
#include <cstring>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <string>
#include <vector>
#include <math.h>
#include <stdlib.h>
#include <cstdlib>
#include <sstream>
#include <stdio.h>
#include <ctime>
#include <unistd.h>

#define PI 3.14159265

using namespace cv;
using namespace std;

// Keep track of video camera frame number
long frameNumber = 0;

// Write to files every time the counter reaches zero
const int FRAME_EVERY = 3;
int currentFrame = FRAME_EVERY;

// Assign unique ID for each direction.
// The ID must be in sync with the GUI direction values
// (Please refer to the documentation for more information about the GUI
code)

int direction = 0;
clock_t start_time;
```

```
const int FORWARD = 1;
const int REVERSE = 2;
const int RIGHT = 3;
const int LEFT = 4;
const int ROTATE = 5;
const int PAUSE = 6;
const int STOP = 7;

/**
* Get the distance between the object and the camera
* Please refer to the documentation report for more information
* about the calculations
* @return Distance in cm
**/
double getDistance(double realDimention, double digitalDimention) {
    double FOCAL_LENGTH = 339.079; // in pixels
    int ERROR_MARGIN = 0; //pixels lost due to selection of circular shape
    return realDimention * FOCAL_LENGTH / (digitalDimention +
ERROR_MARGIN);
}

/**
* Detect the edge point of the object. This information allows the
* conclusion of the object digial radius
* @param imageDest Current video frame
* @param def Default point value in case no object was found
* @return arbitrary point on the object edge
**/
Point edgePoint(Mat imageDest, Point def) {
    int thresh = 100;

    // Canny Algorithm for object edge detection
    Canny(imageDest, imageDest, thresh /*threshold1*/, thresh*2
/*threshold2*/, 3/*apertureSize*/);

    // Prepare data structure to store the edge points
    vector<vector<Point> > contours;
    vector<Vec4i> hierarchy;

    // Perform the countour finder
    findContours(imageDest, contours, hierarchy, CV_RETR_EXTERNAL,
CV_CHAIN_APPROX_SIMPLE, Point(0, 0));
```

```
    // If no object foud, return the provided default value
    if(contours.size() == 0) {
        return def;
    }

    // Return a point from the countour
    return contours[0][0];
}

/**
* Predefined set of colors that can be detected by the program
* The colors supported by the current program are blue and red
* @param specs Data strucutre to hold the HSV channel color information
* based on the color specified
* @param color Color of the object (blue/red)
**/
void get_color_specs(vector<vector<int> > &specs, string color){

    if (!color.compare("blue")) {
        specs[0][0] = 100;
        specs[0][1] = 115;
        specs[0][2] = 50;
        specs[1][0] = 130;
        specs[1][1] = 255;
        specs[1][2] = 255;
    } else if (!color.compare("red")) {
        specs[0][0] = 0;
        specs[0][1] = 197;
        specs[0][2] = 109;
        specs[1][0] = 182;
        specs[1][1] = 255;
        specs[1][2] = 255;
    } else {
        specs[0][0] = 255;
        specs[0][1] = 255;
        specs[0][2] = 255;
        specs[1][0] = 255;
        specs[1][1] = 255;
        specs[1][2] = 255;
    }
}

void drive(int left, int right) {
```

```
    stringstream ss;
    ss << "/home/debian/self_driving_car/Write " << left << "," << right <<
"#" << endl;
    cout << left << "," << right << "#" << endl;
    system(ss.str().c_str());
}

/**
* Drive the car based on the angle and distance
* Decisions are mainly taken based on experiments
* @param angle Car angel
* @param distance Distance of the car from the camera
* @param diameter Digital diameter of the circular object
* @param loop_count Necessary to keep from writing to Atmega328p faster
than it can read messages
* @return Direction code
**/
void find_ball(double angle, double distance, double diameter, int
&loop_count) {

    int full_speed = 200;
    int rotation_speed = 115;
    int turn_speed = 40;
    int pausing_distance = 75;
    int target_angle = 5;

    if (loop_count > 2) {

        // If no object found: rotate
        if(diameter == 0 && direction != ROTATE) {
            cout << endl << "rotating ";
            drive(rotation_speed, -rotation_speed);
            direction = ROTATE;
            loop_count = 0;

            start_time = clock();

        // If object is within pausing_distance and visible: pause
        } else if(distance <= pausing_distance && diameter > 0 && direction
!= PAUSE) {
            cout << endl << "pausing ";
            drive(0, 0);
```

```
            direction = PAUSE;
            loop_count = 0;

            cout << "\n**** BALL FOUND ****\n" << endl;

        // If object more than target_angle degrees to right and farther
than pausing distance: turn right
        } else if (angle > target_angle && distance >= pausing_distance &&
direction != RIGHT){
            cout << endl << "turning right ";
            drive(turn_speed, -turn_speed);
            direction = RIGHT;
            loop_count = 0;

        // If object more than target_angle degrees to left and farther
than pausing distance: turn left
        } else if (angle < -target_angle && distance >= pausing_distance &&
direction != LEFT) {
            cout << endl << "turning left ";
            drive(-turn_speed, turn_speed);
            direction = LEFT;
            loop_count = 0;

        // If ball is past pausing distance and within target_angle:
forward
        } else if (distance > pausing_distance && angle < target_angle &&
angle > -target_angle && direction != FORWARD) {
            cout << endl << "going forward ";
            drive(full_speed, full_speed);
            direction = FORWARD;
            loop_count = 0;

        // If ball rotates ~360 degrees and doesn't see ball: stop
        } else if (direction == ROTATE) {
            clock_t rotation_duration = (clock() - start_time) /
(double)(CLOCKS_PER_SEC);
            cout << "\trot time: " << rotation_duration << " s" << endl;
            if (rotation_duration > 12) {
                drive(0, 0);
                cout << "\n**** BALL NOT FOUND ****\n" << endl;
                direction = STOP;
            }
        }
```

```
    }

    loop_count++;
}

/**
* Start the OpenCV program
**/
int main( int argc, char** argv ) {

    // Capture video number is: /dev/video(0/1)
    int cap_num  = atoi(argv[1]);

    // Color blue/red
    string color = argv[2];

    // Start camera
      VideoCapture cap;
      if(!cap.open(cap_num))
      return 1;

    // Configure the camera for fast capture and good resolution
    cap.set(CV_CAP_PROP_FRAME_WIDTH, 320);
    cap.set(CV_CAP_PROP_FRAME_HEIGHT,180);
    cap.set(CV_CAP_PROP_FPS , 30);

    // loop_count is used to make sure that the find_ball function has
looped at least twice
    // before sending a message to the Atmega328p. This was created in
response to a bug found
    // when writing too fast to the Atmega328p. The direction variable
would be modified however
    // the write message wouldn't be interpreted. This causes the car to
get stuck in a
    // certain direction. Initialized to 2 so that the car begins moving
immediately
    int loop_count = 2;

    // Keep taking pictures
    while(1) {

        // Store the frame in a matrix
        Mat imageSrc;
```

```
        cap >> imageSrc;

        // Fetch the color information
        vector<vector<int> > color_specs(2, vector<int>(3));
        get_color_specs(color_specs, color);

        // HSV low-high values
        int lowH = color_specs[0][0];
        int highH = color_specs[1][0];

        int lowS = color_specs[0][1];
        int highS = color_specs[1][1];

        int lowV = color_specs[0][2];
        int highV = color_specs[1][2];

        // Destination image
        Mat imageDest;

        // Convert BGR to HSV
        cvtColor(imageSrc, imageDest, COLOR_BGR2HSV);

        // Get colors in specified range
        inRange(imageDest, Scalar(lowH, lowS, lowV), Scalar(highH, highS,
highV), imageDest);

        // Morphological opening
        erode(imageDest, imageDest, getStructuringElement(MORPH_ELLIPSE,
Size(5, 5)) );
        dilate(imageDest, imageDest, getStructuringElement(MORPH_ELLIPSE,
Size(5, 5)) );

        // Morphological closing
        dilate(imageDest, imageDest, getStructuringElement(MORPH_ELLIPSE,
Size(5, 5)) );
        erode(imageDest, imageDest, getStructuringElement(MORPH_ELLIPSE,
Size(5, 5)) );

        // Create moment
        Moments mmts = moments(imageDest);

        // Calculate center x and y (Centroids)
        double x_object = mmts.m10 / mmts.m00;
```

```
        double y_object = mmts.m01 / mmts.m00;

        // Center of image
        cv::Size size = imageSrc.size();
        double x_center = size.width/2.0f;
        double y_center = size.height/2.0f;

        // Contour
        Mat tmpDest = imageDest.clone();
        Point point = edgePoint(tmpDest, Point(x_center, y_center));

        // Calculate digital diameter in cm
        double diameter = norm(Point(x_object, y_object)- point)*2;
        double realDiameter = 6.5;

        // Calculate the real distance
        double distance = getDistance(realDiameter, diameter);

        // Get rotation angle
        int digitalDiff = x_object - x_center;
        double realDiff = digitalDiff * (realDiameter / diameter);
        double rotation_angle = atan(realDiff / distance) * 180 / PI;

        // If no object found, then diameter is set to zero
        if(isnan((double)x_object) && isnan((double)y_object)) {
            diameter = 0.0;;
        }

        // Log the calculated information
        cout << endl;
        printf("\t%-10s%4.2f\n", "angle:", rotation_angle);
        printf("\t%-10s%4.2f\n", "distance:", distance);
        printf("\t%-10s%4.2f\n", "diameter:", diameter);

        // Draw circle at x and y
        Mat tmpSource = imageSrc.clone();
        circle(tmpSource, Point(x_object,y_object), 3, Scalar(229, 240,
76), 2);
        circle(tmpSource, Point(x_object,y_object), diameter/2, Scalar(44,
252, 14), 3);

        // Center
        circle(tmpSource, Point(x_center,y_center), 2, Scalar(255, 255,
```

```
255), 2);

        // Logic to navigate to ball
        find_ball(rotation_angle, distance, diameter, loop_count);

        // Write images and text into the file system/
        // Director /var/www/html correspond to the path for
        // Apache2 server. All files placed in this directory will be
        // accessible on all users in the network over host IP and port 80

        string path =
"/home/debian/mr_robot/dashboard/mr-robot-node/public/debug/";
        string outPath = path + "out.jpg";
        string bwPath = path + "bw.jpg";
        string infoPath = path + "info.txt";

        if(--currentFrame == 0) {
            imwrite(outPath, tmpSource);
            imwrite(bwPath, imageDest);
            currentFrame = FRAME_EVERY;
        }
        ofstream myfile;
        myfile.open (infoPath.c_str());
        myfile << "Distance from camera: " << distance << " cm\n";
        myfile << "Rotation angle: " << rotation_angle << "\n";
        myfile << "Digital diameter: " << diameter << " px\n";
        myfile << "Frame number: " << ++frameNumber << "\n";

        string dir_message = "Direction: ";
        switch (direction) {
            case FORWARD:
                myfile << dir_message << "Forward" << endl;
                break;
            case REVERSE:
                myfile << dir_message << "Reversing" << endl;
                break;
            case RIGHT:
                myfile << dir_message << "Right" << endl;
                break;
            case LEFT:
                myfile << dir_message << "Left" << endl;
                break;
            case ROTATE:
```

```
                myfile << dir_message << "Rotating" << endl;
                break;
            case PAUSE:
                myfile << dir_message << "Pause" << endl;
                break;
            case STOP:
                myfile << dir_message << "Stop" << endl;
                break;
        }
        myfile << "DIR_CODE: " << direction << "\n";
        myfile.close();

        if (direction == STOP) {
            return 0;
        }

    }

    return 0;
}
```

It is required to build this file in order for the GUI to start the program. To build the file, run:

```
$ cd /home/debian/self_driving_car/open_cv/
$ ./run clean build
```

3.2.9 - Enabling Wifi on Kernel 3.x

Unlike on Kernel 4.x, wifi on Kernel 3.x is not as easy to setup. We tried several tutorials online, but the one that worked for us was a tutorial by Adafruit Learning System.

3.2.9.1 - Quick tutorial

Uncomment the following line (or a similar one containing HDMI) from /boot/uboot/uEnv.txt:

```
cape_disable=capemgr.disable_partno=BB-BONELT-HDMI,BB-BONELT-HDMI
N
```

Note: Be careful not remove the line containing HDMI/eMMC because you might not be able to boot the Beaglebone next time you run it. More details in the full tutorial (link below)

Now uncomment and enter ssid and password in /etc/network/interfaces:

```
     auto wlan0
     iface wlan0 inet dhcp
          wpa-ssid "<SSID>"
          wpa-psk  "<PASSWORD>"
```

To request an IP from the router, run:

```
ifup wlan0
```

After running `ifup` command, your Beaglebone must be connected. To verify that an IP was assigned to your wifi interface, run `ifconfig`. More details about using this command can be found in "Enabling wifi on Kernel 4.x" section.

3.2.10 - Configuring SPI on Kernel 3.x

3.2.10.1 - Hardware configuration

The hardware part of configuring the SPI is the same one as described in "Configuring SPI on Kernel 4.x" section.

3.2.10.2 - Cape manager

The main difference between both kernels is the location of the bone_capemgr. On kernel 3.x, bone_capemgr under `/sys/devices/bone_capemgr.x` where x is a number which can be different on different devices.

Display the content of `/sys/devices/bone_capemgr.x/slots`:

```
$ more /sys/devices/bone_capemgr.9/slots
     0: 54:PF---
     1: 55:PF---
     2: 56:PF---
     3: 57:PF---
     4: ff:P-O-L Bone-LT-eMMC-2G,00A0,Texas
Instrument,BB-BONE-EMMC-2G
     5: ff:P-O-- Bone-Black-HDMI,00A0,Texas
Instrument,BB-BONELT-HDMI
     6: ff:P-O-- Bone-Black-HDMIN,00A0,Texas
Instrument,BB-BONELT-HDMIN
```

Note that the HDMI is not loaded because the L flag is not enabled.

If an L is displayed for the HDMI, then the uEnv.txt needs to be configured. The configuration of the uEnv.txt is similar to the one done in the “Enabling Wifi on Kernel 3.x”.

If a line showing that SPIDEV is loaded, then this does not necessarily mean that the SPI is going to work. It is recommended that the uEnv.txt is edited to disable auto loading SPIDEV.

To load SPI, similar to procedure on Kernel 3.x, echo SPIDEV into the slots file:

```
$ echo BB-SPIDEV0 > /sys/devices/bone_capemgr.9/slots
$ more slots
      0: 54:PF---
      1: 55:PF---
      2: 56:PF---
      3: 57:PF---
      4: ff:P-O-L Bone-LT-eMMC-2G,00A0,Texas Instrument,BB-BONE-EMMC-2G
      5: ff:P-O-- Bone-Black-HDMI,00A0,Texas Instrument,BB-BONELT-HDMI
      6: ff:P-O-- Bone-Black-HDMIN,00A0,Texas Instrument,BB-BONELT-HDMIN
      7: ff:P-O-L Override Board Name,00A0,Override Manuf,BB-SPIDEV0
```

Note that SPI line appears and the L flag is enabled.

3.2.11 - Write.c for Kernel 3.x

This file is a modified version of a file which is meant to test the functionality of SPI. The original can be found under the infamous Linux creator Linus Torvalds’ repo at:

https://github.com/torvalds/linux/blob/master/tools/spi/spidev_test.c

The modified version can be found under our repo at:

https://github.com/ml-davis/self_driving_car/spi/kernel3.x/write.c

Write.c

```
/*
 * Use this program to send input data from the BBB to the ATMega using SPI
 * Protocol:
 *      <size>:<message>
 */

#include <stdint.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
```

```c
#include <getopt.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <linux/types.h>
#include <linux/spi/spidev.h>
#include <string.h>
#include <math.h>

#define ARRAY_SIZE(a) (sizeof(a) / sizeof((a)[0]))

static void pabort(const char *s)
{
	perror(s);
	abort();
}

static const char *device = "/dev/spidev1.0";
static uint8_t mode = 0;
static uint8_t bits = 0;
static uint16_t delay = 0;
char *input_tx;

// This speed value works
static uint32_t speed = 0;

/*
 *  Unescape - process hexadecimal escape character
 *      converts shell input "\x23" -> 0x23
 */
static int parse(char *_dst, char *_src, size_t len)
{
	int ret = 0;
	char *src = _src;
	char *dst = _dst;
	unsigned int ch;

	while (*src) {
		*dst++ = *src++;
		ret++;
	}
	return ret;
}

static void transfer(int fd, uint8_t const *tx, uint8_t const *rx, size_t len)
{
	int ret;

	struct spi_ioc_transfer tr = {
```

```
                .tx_buf = (unsigned long)tx,
                .rx_buf = (unsigned long)rx,
                .len = len,
                .delay_usecs = delay,
                .speed_hz = 0,
                .bits_per_word = 0,
        };

        ret = ioctl(fd, SPI_IOC_MESSAGE(1), &tr);
        if (ret < 1)
                pabort("can't send spi message");
}

void print_usage(char *str) {
        printf("Program usage:\n    %s <message>\n", str);
}

int main(int argc, char *argv[]) {
        int ret = 0;
        int fd;
        uint8_t *tx;
        uint8_t *rx;
        int size;

        fd = open(device, O_RDWR);
        if (fd < 0)
                pabort("can't open device");

        /*
         * spi mode
         */
        ret = ioctl(fd, SPI_IOC_WR_MODE, &mode);
        if (ret == -1)
                pabort("can't set spi mode");

        ret = ioctl(fd, SPI_IOC_RD_MODE, &mode);
        if (ret == -1)
                pabort("can't get spi mode");

        /*
         * bits per word
         */
        ret = ioctl(fd, SPI_IOC_WR_BITS_PER_WORD, &bits);
        if (ret == -1)
                pabort("can't set bits per word");

        ret = ioctl(fd, SPI_IOC_RD_BITS_PER_WORD, &bits);
        if (ret == -1)
```

```
                pabort("can't get bits per word");

        /*
         * max speed hz
         */
        ret = ioctl(fd, SPI_IOC_WR_MAX_SPEED_HZ, &speed);
        if (ret == -1)
                pabort("can't set max speed hz");

        ret = ioctl(fd, SPI_IOC_RD_MAX_SPEED_HZ, &speed);
        if (ret == -1)
                pabort("can't get max speed hz");

        if(argc != 2) {
                printf("No message provided.");
                print_usage(argv[0]);
                return 1;
        }

        input_tx = argv[1];
        // puts("Calculating size before parsing ...");
        size = strlen(input_tx);

        // printf("size: %d\n", size);

        // Allocate memory
        tx = malloc(size);
        rx = malloc(size);

        // puts("Calculating size after parsing ...");
        size = parse((char *)tx, input_tx, size);

        // printf("size: %d\n", size);

        // puts("Starting transfer ...");
        transfer(fd, tx, rx, size);
        free(rx);
        free(tx);

        close(fd);

        puts("Sent message from write.c");

        return ret;
}
```

3.3 To compile this file, navigate to the kernel directory directory and run the build file:

```
$ ./build.sh
```

The `write` file located in the `control` directory (parent of the kernel directory) must point to the correct generated executable. To regenerate that `write` file, run from the `control` directory:

```
$ ln -s kernel3/write.c write
```

All dependent program should be ready to execute this file now.

3.4 - Dashboard

3.4.1 - Front-end

The dashboard front-end is built using Angular 2 framework, a library developed by Google. The primary programming language used in this framework is TypeScript, which brings the idea of object oriented programming to JavaScript. The front-end part of the dashboard is responsible for handling all the request and interactions from the user.

In order to request actions from the car, the front-end communicates with the back-end through Web Socket communication. An example of such a communication, is controlling the car by clicking on directional arrows triggering physical movements of the car.

The Beaglebone Black IP address must be registered correctly in the following file:

```
self_driving_car/dashboard/mr-robot-html/src/assets/server.json
{
  "url":"http://192.168.0.199",
  "port":3030
}
```

The Angular 2 application is located under: mr_robot/dashboard/mr-robot-html/. Since the application is built over a framework, running it requires installing Node and Angular 2 on the machine first. The technologies version used are `node:6.6` and `angular-cli@1.0.0-beta.28.3`. After installing Angular 2, the program can be launched by navigating into mr-robot-html directory and then typing: `ng serve --host=0.0.0.0`.

*An alternative method (**recommended**) is suggested later in this report under the Docker section.*

3.4.2 - Back-End

The dashboard backend is built using NodeJS technology. The choice of this server was made because it handles socket communications easily. The server runs as a background process and keeps listening for a client to establish a connection. Once a client connects, the user can emit several events from the front-end side, passing them to the server through a socket frame, which in its turn executes commands directly on the operating system in order to control the car.

The server directory is located under: `self_driving_car/dashboard/mr-robot-node/`. Running the server requires two steps. First, installing the application dependencies using the following command: `` `npm install` ``. Second, starting the server by running: `` `npm start` ``.

3.4.3 - Docker for the front-end

Starting the front-end requires several steps and specific version of software and libraries to be installed on the user's machine. To simplify this process, a Dockerfile text has been created to embed the application into a container where the environment of the system is configured to work with the technologies used.

The Dockerfile is located under: `self_driving_car/dashboard/mr-robot-html`.

Dockerfile

```
FROM node:6.6
ENV ANG_APP=/opt/mr-robot-html
RUN mkdir $ANG_APP
WORKDIR $ANG_APP
COPY . $ANG_APP
RUN npm install -g angular-cli@1.0.0-beta.28.3 && npm cache clean
RUN rm -rf node_modules
RUN npm install
EXPOSE 4200
CMD ng serve --host=0.0.0.0
```

The only requirement to start building an image of the front-end is installing Docker on the machine. Docker's installation depends on which OS the host is running. Installation details can be found on the Docker website.

Once Docker is installed, the first step is to build the image:

```
$ docker build -t dashboard:latest .
```

This command builds the docker image of the front-end and labels the image as dashboard with the latest release. Second, to start the application, a container of the created image must be launched:

```
$ sudo docker run --name self_driving_car -p 4200:4200 -it dashboard:latest
```

This command runs the container and forward the port 4200 from the container to the host machine. After starting the container, the application can be accessed at: http://localhost:4200

To stop the container, run from another terminal the following command:

```
$ sudo docker stop self_driving_car
```

To remove the container completely from your machine, run:

```
$ sudo docker rm self_driving_car
```

To remove the built image from your machine, run:

```
$ sudo docker rm dashboard:latest
```

Note: Configure the Beaglebone Black IP address in the server.json file before building the Docker image.

3.4.4 - Functionality

3.4.4.1 - Main Page

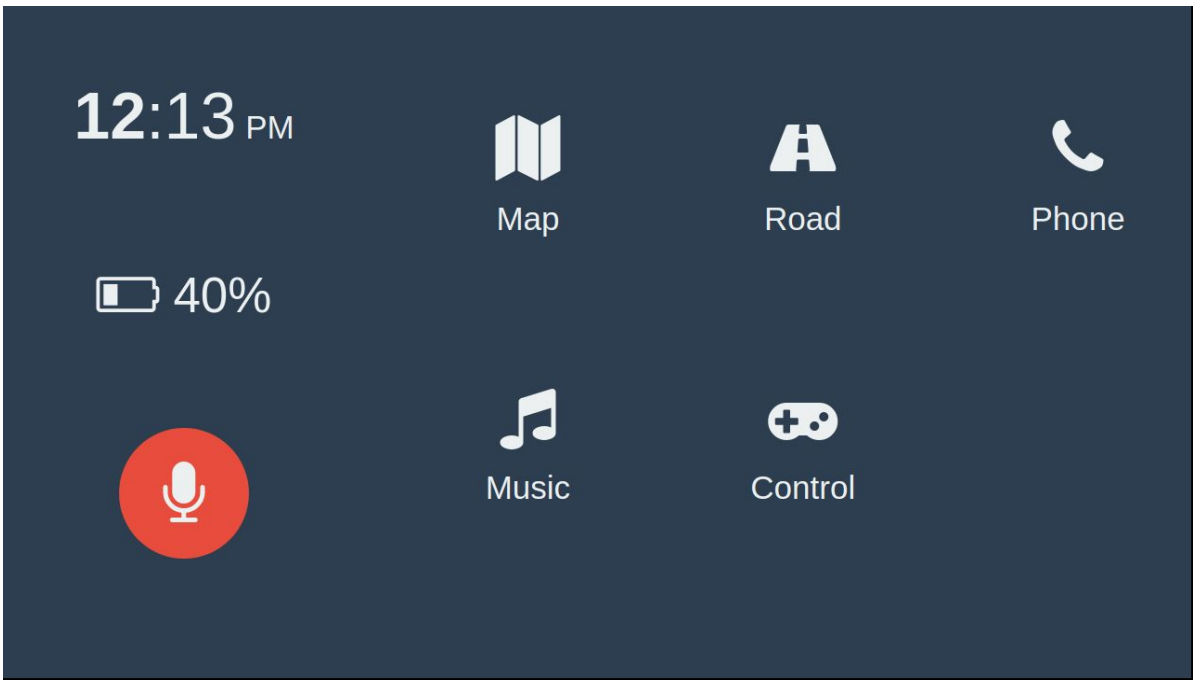


Figure 26

The main page is the first thing our users will see when they launch the GUI. It is supposed to simulate an interface similar to what a user would see if they were in a self-driving car. There are five buttons on the main screen which we will go into in more detail.

3.4.4.2 - Map

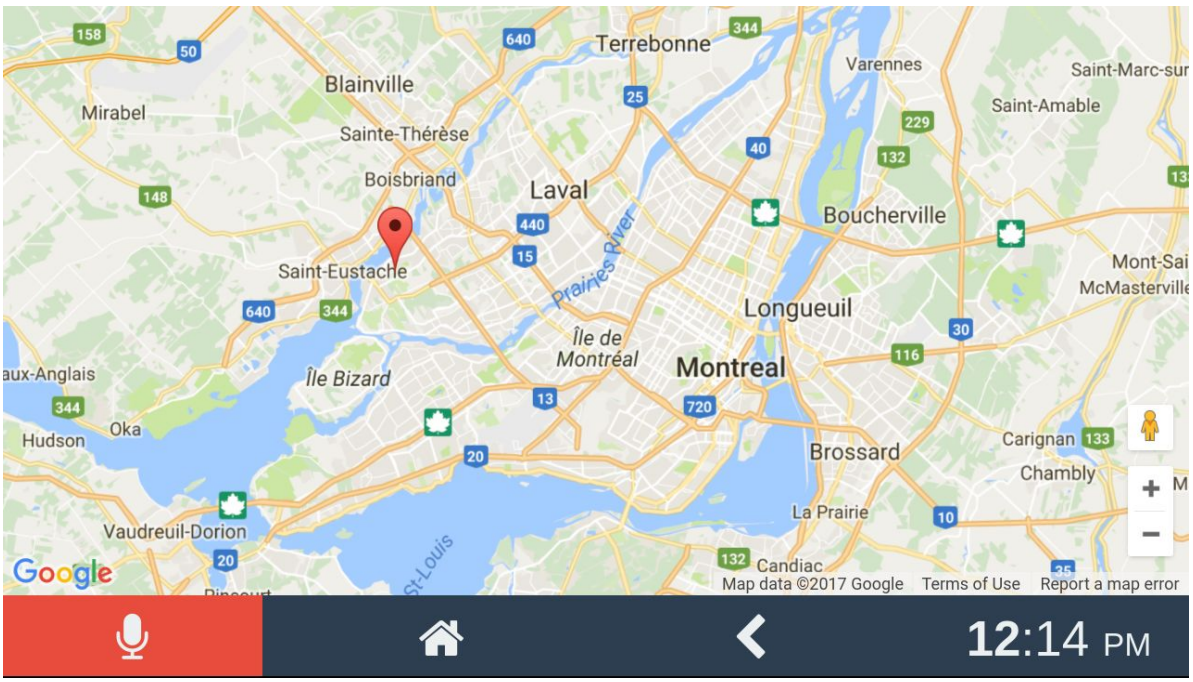
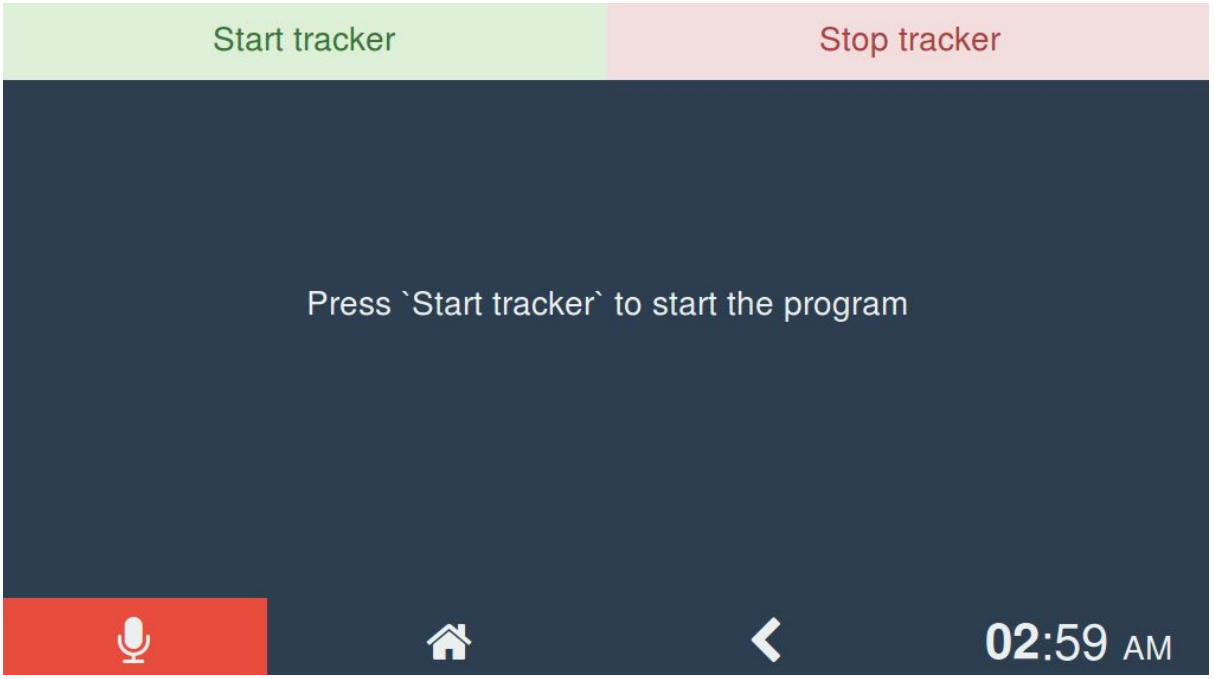


Figure 27

The map is similar to what users see with their navigation systems. If we had a GPS added to our BeagleBone Black, we could use it to track our movements. However, we decided for this project to select random coordinates as a proof of concept.

3.4.4.3 - Road



The Road feature does a few things. It shows us the line of sight of the vehicle using the webcam for footage. It shows us the binarized image, what the ball is detecting to track. Finally, it has buttons to launch and stop our ball-tracking program BallTracker.cpp.

Once the “Start Tracking” button is pressed, the car will begin searching for our target object and trying to navigate to it. If we hit “Stop Tracking”, the program will stop.

3.4.4.4 - Phone

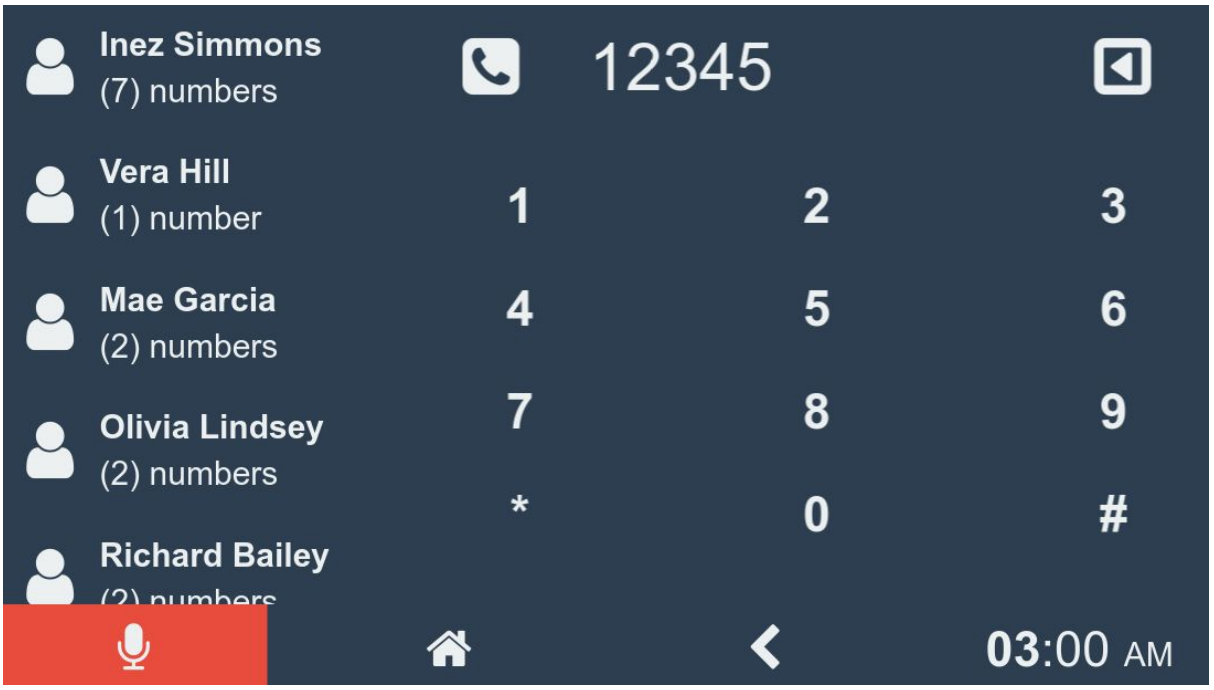


Figure 28

Phones are common features in cars now a days. We added this feature for fun. It can be used to type numbers and save contacts, however, it can not actually make calls. We could always connect to a VOIP service in the future in order to get real functionality.

3.4.4.5 - Music Player

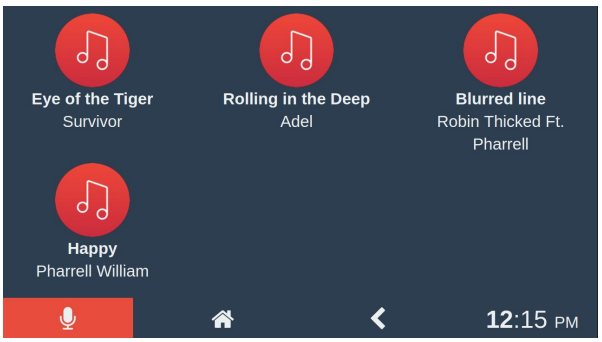


Figure 29

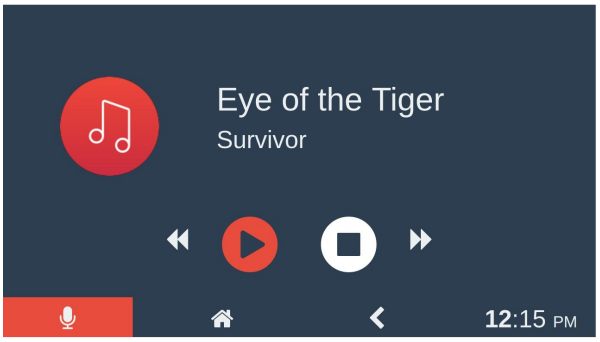
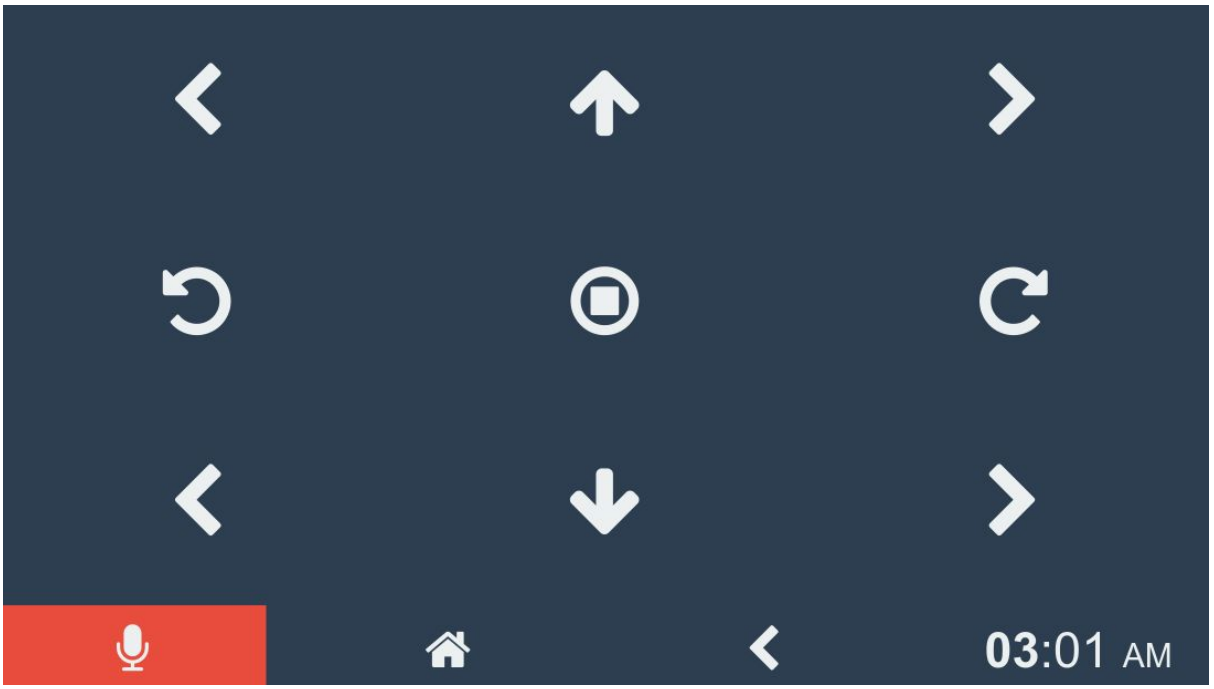


Figure 30

A music player is another extremely common feature in car dashboard these days. We added a music player and pre-loaded it with a few songs. In the delivered release of this project, the music does not

start an audio process because adding speakers creates instability for other mounted devices like the wifi and the camera.

3.4.4.6 - Control



Control is a feature that allows us to manually override the car. It gives us nine buttons.

- Forward
- Reverse
- Rotate Left
- Rotate Right
- Turn Forward Left
- Turn Forward Right
- Turn Reverse Left
- Turn Reverse Right

These allow us to control the car as if it were a remote control car.

3.4.5 - Code Structure

We decided to leave out the code for the GUI as it was too long and would take up far too much space. However, it can be found in our repo under:

https://github.com/ml-davis/self_driving_car/dashboard/

4 - Theory

4.1 - SPI

SPI is the protocol we used to send messages from our Beaglebone Black to the Atmega328p. SPI is typically used with a master/slave paradigm where the master is connected to a bus with multiple slaves also connected to the bus. The master can then send messages over the bus which will be received by all slaves. In our case, the Beaglebone Black is the master and the Atmega328p's are the slaves. Note, the master is the one which initiates a connection and controls it. Once a connection is established, the chips are free to transmit and receive data with each other. SPI needs the following four wires for communication:

- MISO: Master In, Slave Out
- MOSI: Master Out, Slave In
- SCK: This is the SPI clock line
- SS: Slave Select

4.2 - PWM

PWM stands for pulse width modulation. It is a way to get an analog value when only dealing with a digital means (ones and zeros). It creates this analog value by examining the square waves created by the alternating of the digital signals. By determining how long the square wave is at a high point versus its time at a low point, it produces an analog value between 0 and 255.

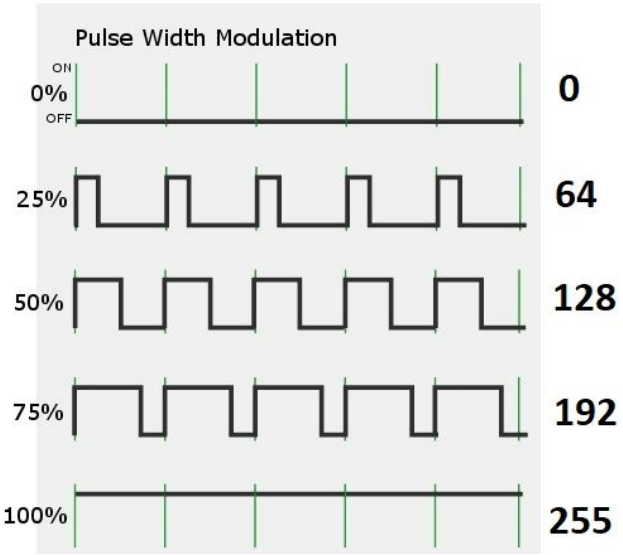


Figure 31

4.3 - UART

4.3.1 - Reading UART

In order to see these messages provided by UART, the Atmega328p using the TX and RX pins must be plugged into a computer, rather than the battery. We can then run the Bash script monitor.sh to view the UART messages. In order to run this program, we must pass it two arguments:

- Which device it's using (ttyUSB0 or ttyUSB1)
- And the baud rate. (we used 4800)

The script then reads from the tty device specified (in our case `/dev/ttyUSB0`) and prints it to the console.

monitor.sh

```
monitor() {
    if [[ ${#@} < 2 ]]
    then
        cat<<EOF
Monitor Serial communication.

Usage:
    monitor ttyUSB0 4800
    monitor ttyUSB1 2400
    etc ...
EOF
    else
        if [[ ! -e /dev/$1 ]]
        then
            echo "$1 not found! Here's the list of ttyUSB found:"
            ls /dev/ | grep ttyUSB | xargs -I {} echo "    - {}"
        else
            sudo chmod o+rw /dev/$1
            stty -F /dev/$1 $2 raw -echo -echoe -echok -echoctl
-echoke && cat -v < /dev/$1
        fi
    fi
}

monitor $1 $2
```

4.3.2 - Writing to UART

This script was not used during the final implementation of our project but it was a great tool to troubleshoot and ensure that UART is working properly. This script can be used to write to UART. So if one terminal is running monitor.sh and another is running write.sh the monitor terminal can see the messages sent from the write terminal.

write.sh

```
write() {
    if [[ ${#@} < 2 ]]
    then
        cat<<EOF
Write to Serial communication.

Usage:
    write ttyUSB0 "Message 1"
    write ttyUSB1 "Message 2"
    etc ...
EOF
    else
        if [[ ! -e /dev/$1 ]]
        then
             echo "$1 not found! Here's the list of ttyUSB
found:"
             ls /dev/ | grep ttyUSB | xargs -I {} echo "    - {}"
        else
            sudo chmod o+rw /dev/$1
            echo -e -n "$2"  > /dev/$1
        fi
    fi
}

write $1 $2
```

4.4 - Device Tree Overlays

A device tree is a data structure that describes hardware. It can be used to provide the operating systems with details of a device so that we can work with it to achieve certain functionality. This data structure is passed to the operating system at boot time. The format of a device tree is a simple tree structure of nodes and properties. Properties are key-value pairs, and nodes may contain both properties and child nodes. They are typically stored in files with a dts extension. For example, taken from http://elinux.org/Device_Tree_Usage

```
/dts-v1/;
```

```
/ {
	node1 {
		A-string-property = "A string";
		a-string-list-property = "str1" , "str2";
		// hex is implied in byte arrays. no '0x' prefix is required
		a-byte-data-property  =   [  01  23  34  56 ];
		child-node1 {
			first-child-property;
			second-child-property = <1>;
			a-string-property = "Hello, world";
		};
		child-node2 {
		};
	};
	node2 {
		an-empty-property;
		a-cell-property = <1 2 3 4>;  /* each number (cell) is a uint32 */
		child-node1 {
		};
	};
};
```

The following shows an arbitrary example of the structure of a device tree and the types that can be nested within them.

Now that we know the basic structure of a device tree, we can go about replacing the arbitrary values provided in our simplified example with relevant properties of the device we are interested in managing. We can see question 4 for a more detailed example of a dts file.

After we have our dts file, we can compile it with the command :

```
$ dtc -O dtb -o <compiled-device-tree-name>.dtb <device-tree-name>.dts
```

This will compile our dts file and convert it to a binary file with the extension dtbo. This file must then be moved to the /lib/firmware folder. After that, it can be loaded into the cape manager with the command:

```
$ sudo sh -c "echo device > /sys/devices/platform/bone_capemgr/slots"
```

Note: This slots path is only valid for kernel 4.x. If you are still using version 3.x, see section 3.3.3.2 for location of slots file.

5 - Debugging and Troubleshooting

5.1 - Atmega328p

The Atmega328p is a little hard to troubleshoot because it does not have an operating system running at its core. One method to investigate problems on the microcontroller is using input and output. In order to write to and read from the Atmega328p, the Arduino IDE can be used. However, the Arduino IDE requires a desktop environment to start, which might not be a common case in the embedded world. For that reason, two alternative tools has been developed to accomplish an external interaction with the Atmega328p. Both tools were discussed previously in the UART section (*see section 4.3*).

5.2 - SPI

SPI is a major part of the project and had to be explored extensively in order to be integrated into the program. It is important to know which kernel version is running on the Beaglebone Black otherwise the instructions found in the Theory SPI section will not be functional.

To verify that SPIDEV0 is loaded, the following files should be available under /dev directory:

- spidev1.0
- spidev1.1

If those file existed by default, it does not necessarily means that the SPI will work for the program. It is possible that the wrong device tree blob objects are loaded at boot time. If that is the case, then the uEnv.txt must be configured.

5.3 - SPI and write.c

If the write.c does not work on your program, it is possible that the configuration is different on your device. The write.c files are a modified version of the spidev_test.c file, as mentioned in previous sections. Therefore, you might have to reconfigure those files.

Instead of configuring them, configure the original file spidev_test.c then modify that file to look similar to write.c.

To build spidev_test.c:

```
$ gcc -o spidev_test spidev_test.c
```

To test spi with different configuration:

```
$ ./spidev_test -D /dev/spidev1.0 -p "my message" -s 10000
```

Keep trying different flags and values until the SPI message is read by the Atmega328p.

To see all the possible configurations:

```
$ ./spidev_test --help
```

5.4 - uEnv.txt

Several uEnv.txt files might exist in different Beaglebone Black images. If some of the changes are being made to those files without noticing changes, it is possible that the wrong uEnv.txt file is being edited.

5.5 - Camera

Several methods can be used to take pictures from the camera. First, take a picture every second and store them in a directory. Second, stream the video and broadcast over a UDP connection. Third, use a library like OpenCV to takes care of taking pictures and processing them.

5.5.1 - Software required

v4l-utils

- To install it run `apt-get install v4l-utils`

avconv

- To install it run `sudo apt-get install libav-tools`

5.5.2 - Configuring the camera

Configuring the camera requires attaching the device into the Beaglebone Black first.

To check what are the format supported by the camera:

- Run: `v4l2-ctl --list-formats`

To check which format is activated:

- `v4l2-ctl --all`

To change video format to index 1 (just an example):

- `v4l2-ctl --set-fmt-video=pixelformat=1`

To change the resolution captured:

- `v4l2-ctl --set-fmt-video=width=740,height=420`

5.5.3 - Streaming

This is an example of streaming a live video from the Beaglebone Black to the host machine on port 1234 over UDP. The video format is selected to be MJPEG, Other format require different parameters.

The example is based on a repository aiming on providing simple functionalities that can be extended to be used at a bigger scale. The repository can be found at: https://github.com/derekmolloy/boneCV.

To start the example, clone the repository:
```
git clone https://github.com/derekmolloy/boneCV
```

To built the boneCV project, run:
```
cd boneCV && ./build
```

Capture photos using the camera found at /dev/video0 and output the raw data on the STDOUT, then pipe the output to avconv which will take care of streaming it to the host machine using UDP
```
./capture -o -c0 | avconv -f mjpeg -i pipe:0 -f mpegts udp://192.168.1.187:1234
```

Explanation:

- capture The capture library modified to easily handle H265 video format
- -o Output to the STDOUT
- -c0 Infinite frames
- avconv Used to stream in this case. Can also store to files.
- -f mjpeg format MJPEG
- -i pipe:0 Read input from the STDOUT of the piping program
- -f mpegts mpeg-ts
- udp://192.168.1.187:1234 Host IP and port

To display the video streamed on the host machine, install vlc then run:
```
vlc udp://@:1234
```

To verify that the host machine is listening on port 1234, run:
```
netstat -na | grep 1234
```

Troubleshooting

- If the capture program gives **select timeout**, run the following command on the Beaglebone Black:
    - rmmod uvcvideo
    - modprobe uvcvideo nodrop=1 timeout=5000 quirks=0x80
- If the avconv program gives **pipe:: Invalid data found when processing input** make sure you have the avconv flags set in the correct order first, or try different arguments.
- To verify that the host machine is receiving data over UDP, run Wireshark or run: sudo tcpdump udp port 1234

5.5.4 - Taking a picture every 10 seconds

```
avconv -f video4linux2 -i
/dev/v4l/by-id/usb-046d_HD_Pro_Webcam_C920_2A4F7FDF-video-index0 -vf fps=1/10
test%3d.jpeg
```

Explanation:

- avconv Used to stream in this case. Can also store to files.

- -i /dev/v4l/by-id/usb-046d_HD_Pro_Webcam_C920_2A4F7FDF-video-index0 The device file. It is not allowed to use /dev/video0. This value depends on the camera attached.
- -vf fps=1/10 One pictures per 10 seconds
- test%3d.jpeg Output file pattern

5.6 - Wifi

The wifi is configured differently on different kernel versions. The instructions must be followed corresponding to the kernel version installed on the Beaglebone Black. If a connection has been established and an IP address has been assigned to the device on the wifi interface, but there still no Internet, then the wifi card might not be compatible with Beaglebone Black or there is not enough power to keep the network connection stable.

6 - Removed features

6.1 - Voice Recognition

Initially, the plan was to integrate Nuance or Google voice recognition into the car software. The idea was to be able to control the car hands-free. However, we decided not to add this feature since the Beaglebone black was already performing slowing when the main processes were running.

6.2 - Tensorflow

In addition to tracking a ball, we planned to teach the car to recognize street signs and traffic lights. For this purpose we worked with Tensorflow, a machine learning library developed by Google, in order to start training the car. The training phase was successful, however the processing part was very slow. On the Beaglebone Black, each object recognized was taking several seconds to produce a result which defeats the purpose of a real-time system. For this reason, this feature was removed from the project.

7 - Installation reference

This section serves as a quick reference for installing the project on the Beaglebone Black. It is important to read the report before proceeding to the steps.

7.1 - Beaglebone Black setup

- Install image on the Beaglebone Black (Image must have kernel 3.x or 4.x)
- Install the required packages including OpenCV
- Clone mr-robot repository under /home/debian/ (create the user debian if not there)
- Install the server dependencies:
    - `cd /home/debian/self_driving_car/dashboard/mr-robot-node/`

    - `npm install`
- Build the OCV.cpp file:
    - `cd /home/debian/self_driving_car/open_cv/`
    - `./run clean build`
- Build the write.c file based on the kernel version:
    - `cd /home/debian/self_driving_car/spi/kernel3.x/ #or`
    - `cd /home/debian/self_driving_car/spi/`
    - `gcc -o Write Write.c`
- [Optional] Setup the wifi

7.2 - Beaglebone Black software

- Load SPI:
    - `sudo -s`
    - `echo BB-SPIDEV0 > /sys/devices/bone_capemgr.*/slots #or`
    - `echo BB-SPIDEV0 > /sys/devices/platform/bone_capemgr/slots`
- Run Node JS server
    - `cd /home/debian/self_driving_car/dashboard/mr-robot-node/`
    - `npm start`

7.3 Host machine software

- Clone mr-robot repository
- Set the Beaglebone IP address in `self_driving_car/dashboard/mr-robot-html/src/assets/server.json`
- Build the Angular 2 Docker image
    - `sudo docker build -t dashboard:latest .`
- Run a container of that image
    - `sudo docker run --name self_driving_car -p 4200:4200 -it dashboard:latest`
- Go to your browser and visit: http://localhost:4200/ to see the GUI