Advanced Engineering Projects

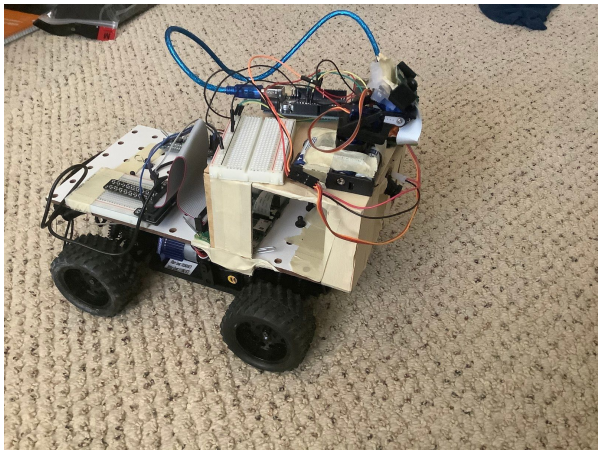
Semester Two Report

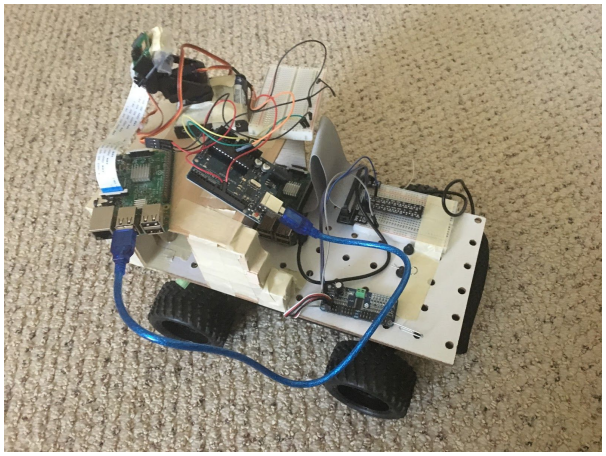
Prototype of Autonomous Rover 1.3

Stefan Prestrelski

Advisor: Mr. Eric Neubauer (Period 4)

Torrey Pines High School





Objectives

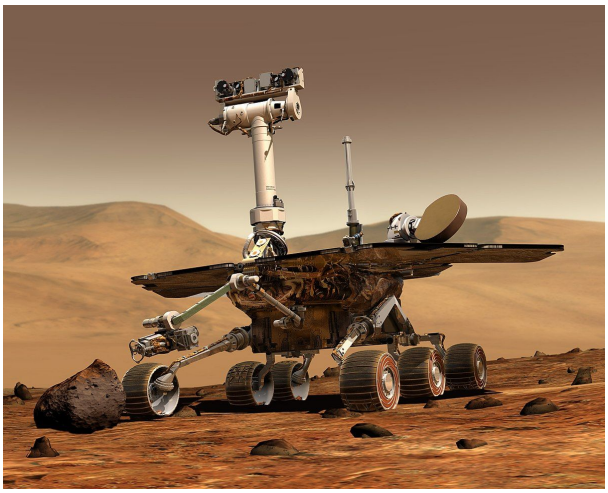
Build a rover that can be controlled remotely using an onboard camera and can perform functions autonomously with a second onboard camera using computer vision that I built in Semester One.

Introduction

Autonomous vehicles have a lot of utility in today's world. They can go places where people cannot, or where it would be too costly to send a human to, such as other planets. These rovers need to be able to operate with minimal, if any, instruction because of the delay of radio signals being broadcast from earth to extraterrestrial bodies eliminating the possibility of real time control.

Another big component of autonomous rovers is cameras. Cameras are perhaps the trickiest sensor to work with, but they can provide unparalleled amounts of data to researchers in addition to helping the rover navigate. The upcoming Mars 2020 Rover will have 6 cameras for better data collection.

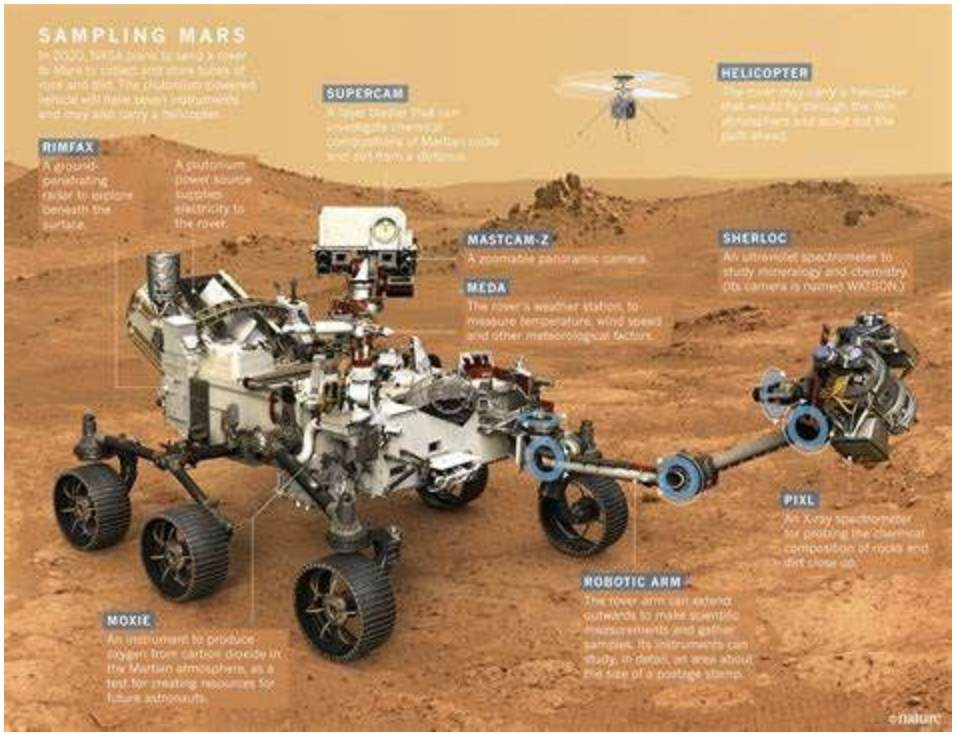
Inspiration



The primary inspiration for my rover was the Mars rovers, including the Pathfinder's Sojourner rover, the Opportunity and Spirit rovers, and Curiosity. All 4 rovers are fully autonomous and Curiosity can even be updated remotely. What separates Curiosity from the other three, however, is its use of machine learning. Previous rovers would need routes to be pre-programmed into them where Curiosity

can receive an endpoint and figure out how to get there itself and avoid potential obstacles.

All the rovers have cameras, but curiosity uses them for more than taking pictures. It also uses the cameras for navigation and obstacle avoidance to allow the rover to navigate unfamiliar terrain. This design inspired the camera mounts on my rover. As shown below, the future Mars 2020 rover has 6 different cameras with different purposes. I hope to learn more about this design and incorporate some of its features onto my rover.



Concept art of Mars 2020 rover with 6 cameras and various other sensors.

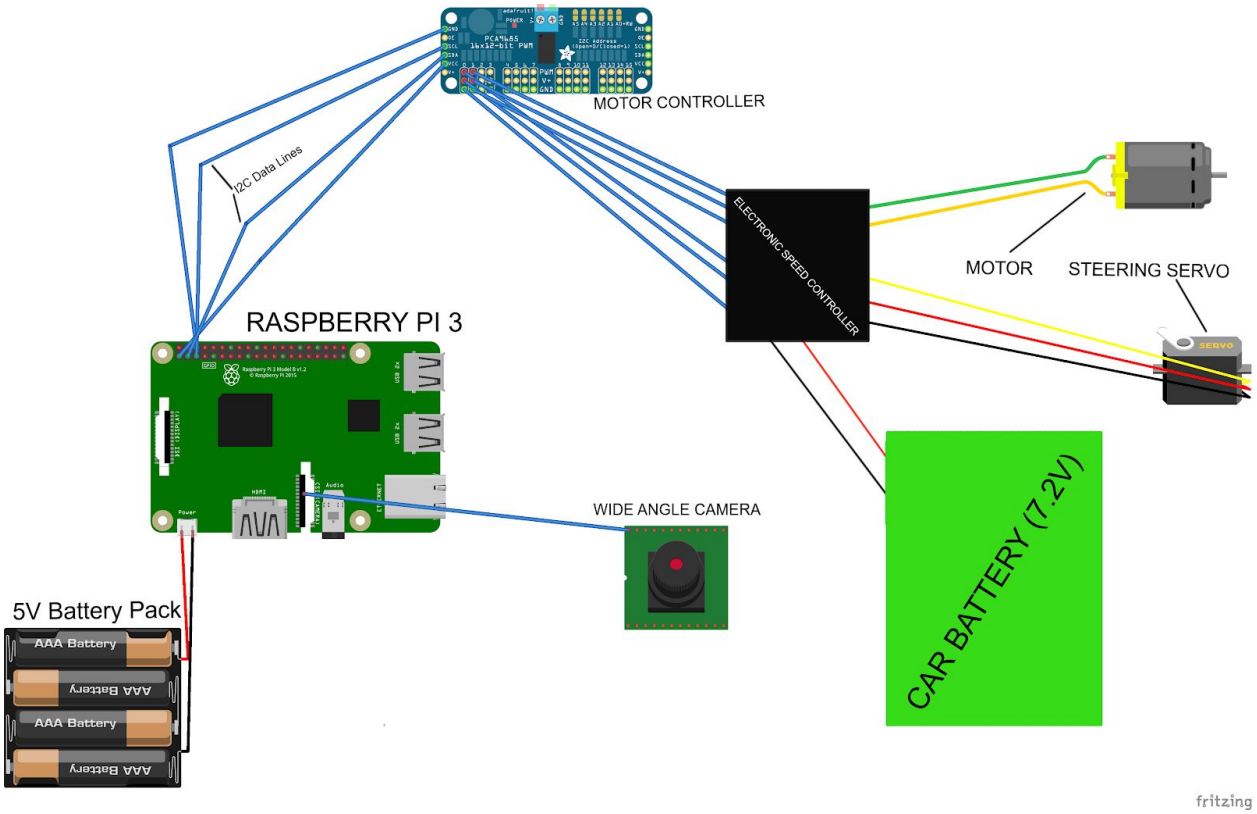
Materials and Hardware

- Raspberry Pi x2
- Arduino
- RC Car Chassis
- Raspberry Pi Camera x2
- Servo x2
- Motor Driver for RC Car
- Jumper Wires
- Cables to connect Pi to Arduino
- Wooden supports and floorplate

Design

A. <u>Electrical Connections and Hardware</u>

The rover's electronics follow the diagram shown below. The camera is connected to a Raspberry pi that controls the Motor driver. This motor driver then controls the electronic steering controller on the rover. Various battery packs provide power to the rover and Pi, which are independent of each other.

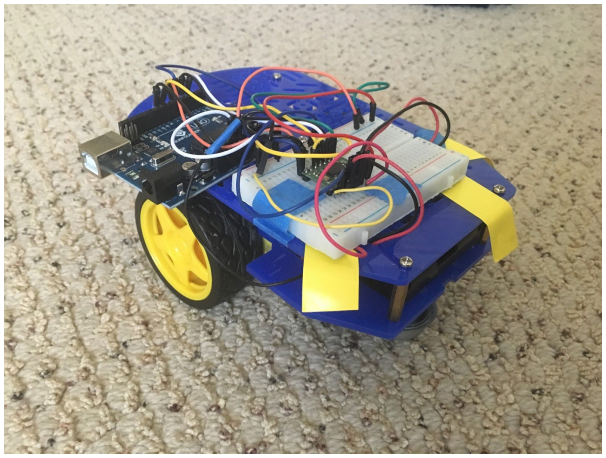
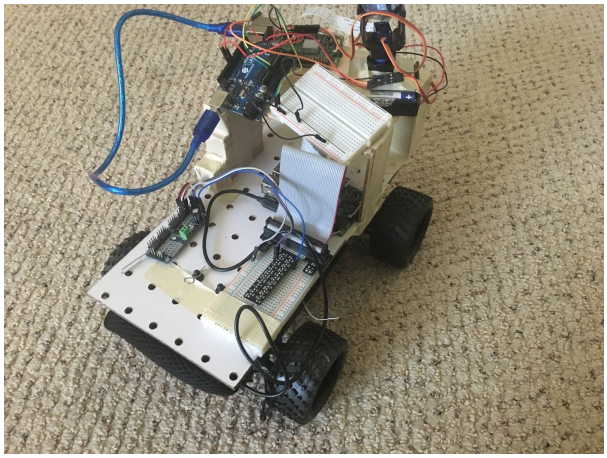


- Rechargeable USB Battery pack powers Pi
- Pi powers Motor controller and uses I2C to send and receive data

- Pi takes inputs from Wide Angle Camera
- Motor Controller uses Pulse Width Modulation to communicate with Car's speed controller
- 7.2V battery powers Car *Separate from Pi's battery
- Servo and Motor controlled by car speed controller

The structure of the car is built on an RC car chassis. All the equipment for the cameras and motor driver is kept on a wooden structure using wood from the Torrey Pines High School Woodshop held together with masking tape. I used tape because wood glue took too long to dry and was finicky to use.

The original car was made on a custom chassis with two wheels and independent motors. While this design worked well, it was simply too small to accomodate all the sensors and hardware needed.

Left: V1.0. Right: V1.3

B. Versions

1. V1.0 arduino with ultrasonic Ping Sensors, line detecting sensor, Color sensor
    - V1.0.1 Ultrasonic ping sensor - sent out sound wave, calculated distance based on time it takes for sound to return
        - Car stops and turns around if object is detected under a certain distance
    - V1.0.2 Line following sensor - 8 line sensors - car turns with tightness depending on where line was detected
        - Middle = no turn
        - Slight L/R turn
        - Moderate L/R turn

        - Tight L/R turn
    - V1.0.3 Color Sensor
        - Returns a color depending on light reflected
        - Circular track of 3 colors, turns left if one color, turns right if another color, went forward if a third color
        - *Include diagram of track*
2. V1.1 the camera controlled by user input
    - V1.1.1 tracked objects and repositioned camera to keep object in center of frame
3. V1.2 Attempt to integrate Object tracking camera to Car (unsuccessful due to size limitations of car)

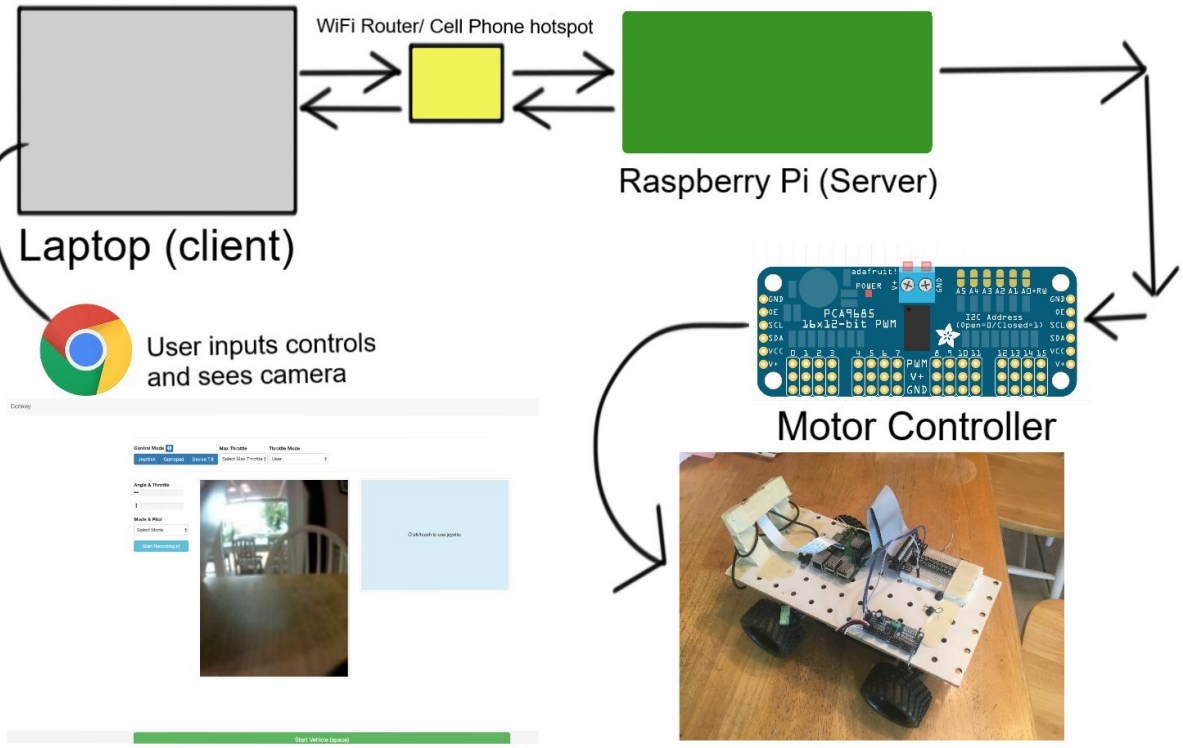
4. V1.3 Larger chassis with an onboard camera with remote control to navigate. A second camera using computer vision I developed in Semester One was mounted on the chassis for object-tracking functionality

Data Exchange and Flow

Data is exchanged between the laptop and the raspberry pi.

The Laptop connects to the WiFi network that the pi is also connected to, and the two establish a connection (usually via an SSH). The command to start the donkey car is then entered, and the Pi establishes a server for the viewing window. The laptop then can connect to the viewing window via a web browser which then communicates with the Pi to send control inputs and receive camera data.

Data Flow Chart:



Results

It is possible to control the car from a computer remotely. However, the car can also be trained to use an autopilot based on tubs of data collected. When collecting a tub of data, the pi takes the position of the throttle, steering, and the image from the camera 30 times per second, then compiles the data. The data has to be offloaded to another computer because the Pi lacks the processing power needed. Once the data is processed, the car will replicate the test drives very accurately.

Algorithms and Computer Code

V1.0.1 Obstacle Avoiding Car

Language: C++

```
int BIN2 = 4;
int PWMB = 5;
int PWMA = 6;
int AIN1 = 7;
int BIN1 = 9;
int triggerPin = 12;
int echoPin = 13;
int AIN2 = 10;
float distance;
void setup() {
  // put your setup code here, to run once:

pinMode (BIN2, OUTPUT);
pinMode (PWMB, OUTPUT);
pinMode (PWMA, OUTPUT);
pinMode (AIN1, OUTPUT);
pinMode (BIN1, OUTPUT);
pinMode (AIN2, OUTPUT);
pinMode(echoPin, INPUT); //remember: the Echo pin is the microphone pin
pinMode(triggerPin, OUTPUT); //remember: the Trig pin is the speaker pin
digitalWrite (BIN2,LOW);
digitalWrite (PWMB,LOW);
digitalWrite (PWMA,LOW);
digitalWrite (AIN1,LOW);
digitalWrite (BIN1,LOW);
digitalWrite (AIN2,LOW);
delay(200);
digitalWrite (PWMB, HIGH);
digitalWrite (PWMA, HIGH);
Serial.begin(9600);
}
void loop() {
  // put your main code here, to run repeatedly:
driveForward();
delay(50);
float dis = getDistance();
if (dis<10){
```

```
  stahp();
  delay(100);
  goBackwards(500);
  int direct = random(1,3);
  if (direct==1){
          goLeft(250);

          }
  else {
          goRight(250);
          //delay(200);
  }

  }
}
void goForward(int duration){
  digitalWrite(AIN1, HIGH);
  digitalWrite(BIN1, HIGH);
  delay(duration);
  digitalWrite(AIN1, LOW);
  digitalWrite(BIN1, LOW);
  //delay(duration);
}
void goBackwards(int duration){
  digitalWrite(AIN2, HIGH);
  digitalWrite(BIN2, HIGH);
  delay(duration);
  digitalWrite(AIN2, LOW);
  digitalWrite(BIN2, LOW);
  //delay(duration);
}
void goRight(int duration){
  digitalWrite(BIN1, HIGH);
  delay(duration);
  digitalWrite(BIN1, LOW);
  //delay(duration);
}
void goLeft(int duration){
  digitalWrite(AIN1, HIGH);
  delay(duration);
  digitalWrite(AIN1, LOW);
  //delay(duration);
}
```

```
void driveForward(){
  digitalWrite(AIN1, HIGH);
  digitalWrite(BIN1, HIGH);
}
void stahp(){
  digitalWrite(AIN1, LOW);
  digitalWrite(BIN1, LOW);
}
float getDistance(){
long duration;
float cm;
float inches;
digitalWrite(triggerPin, LOW);
  delayMicroseconds(2);
  digitalWrite(triggerPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(triggerPin, LOW);

  /* The echo pin is used to read the signal from the PING))): a HIGH
  pulse whose duration is the time (in microseconds) from the sending
  of the ping to the reception of its echo off of an object.*/

  duration = pulseIn(echoPin, HIGH);

  // convert the time into a distance
  cm = float(duration) /29.0 /2.0;
  inches = cm /2.54;

  Serial.print(inches);
  Serial.println(" inches");

  return inches;
}
```

V1.0.2 Line Following Car
Language: C++

```
int BIN2 = 4;
int PWMB = 5;
int PWMA = 6;
int AIN1 = 7;
int BIN1 = 9;
int triggerPin = 12;
int echoPin = 13;
int AIN2 = 10;
float distance;
#include <Wire.h>
#define uchar unsigned char
uchar t;
//void send_data(short a1,short b1,short c1,short d1,short e1,short f1);
uchar data[16];
int thresh = 93;
int loSpeed = 0;
int medSpeed = 50;
int hiSpeed = 70;
int defSpeed = 90;
unsigned long start;
int state = 0;
int lastState = 0;
void setup() {
  // put your setup code here, to run once:

pinMode (BIN2, OUTPUT);
pinMode (PWMB, OUTPUT);
pinMode (PWMA, OUTPUT);
pinMode (AIN1, OUTPUT);
pinMode (BIN1, OUTPUT);
pinMode (AIN2, OUTPUT);
pinMode(echoPin, INPUT); //remember: the Echo pin is the microphone pin
pinMode(triggerPin, OUTPUT); //remember: the Trig pin is the speaker pin
digitalWrite (BIN2,LOW);
digitalWrite (PWMB,LOW);
digitalWrite (PWMA,LOW);
digitalWrite (AIN1,LOW);
digitalWrite (BIN1,LOW);
digitalWrite (AIN2,LOW);
delay(200);
```

```
digitalWrite(AIN1, HIGH);
digitalWrite(BIN1, HIGH);
Serial.begin(9600);
Wire.begin();
 t = 0;
}
void loop() {
  // put your main code here, to run repeatedly:
getLine();
if (data[6]<thresh||data[8]<thresh){
  driveForward();
  state = 1;
}
else if (data[4]<thresh){
  driveLeft();
  state = 1;
}
else if (data[10]<thresh){
  driveRight();
  state = 1;
}
else if (data[2]<thresh){
  driveLeftMed();
  state = 1;
}
else if (data[12]<thresh){
  driveRightMed();
  state = 1;
}
else if (data[0]<thresh){
  driveLeftHi();
  state = 1;
}
else if (data[14]<thresh){
  driveRightHi();
  state = 1;
}
else{
  state = 0;
  if (state != (lastState)){
          start = millis();
  }
  if (millis() - start >= 500){
```

```
          stahp();
  }
}
lastState = state;
}
void goForward(int duration){
  digitalWrite(AIN1, HIGH);
  digitalWrite(BIN1, HIGH);
  delay(duration);
  digitalWrite(AIN1, LOW);
  digitalWrite(BIN1, LOW);
  //delay(duration);
}
void goBackwards(int duration){
  digitalWrite(AIN2, HIGH);
  digitalWrite(BIN2, HIGH);
  delay(duration);
  digitalWrite(AIN2, LOW);
  digitalWrite(BIN2, LOW);
  //delay(duration);
}
void goRight(int duration){
  digitalWrite(BIN1, HIGH);
  delay(duration);
  digitalWrite(BIN1, LOW);
  //delay(duration);
}
void goLeft(int duration){
  digitalWrite(AIN1, HIGH);
  delay(duration);
  digitalWrite(AIN1, LOW);
  //delay(duration);
}
void driveForward(){
analogWrite (PWMB, defSpeed);
analogWrite (PWMA, defSpeed);
}
void stahp(){
  analogWrite(PWMA, 0);
  analogWrite(PWMB, 0);
}
void driveRight(){
analogWrite (PWMB, defSpeed);
```

```
analogWrite (PWMA, hiSpeed);
}
void driveLeft(){
analogWrite (PWMB, hiSpeed);
analogWrite (PWMA, defSpeed);
}
void driveRightMed(){
analogWrite (PWMB, defSpeed);
analogWrite (PWMA, medSpeed);
}
void driveLeftMed(){
analogWrite (PWMB, medSpeed);
analogWrite (PWMA, defSpeed);
}
void driveRightHi(){
analogWrite (PWMB, defSpeed);
analogWrite (PWMA, loSpeed);
}
void driveLeftHi(){
analogWrite (PWMB, loSpeed);
analogWrite (PWMA, defSpeed);
}
void getLine(){
Wire.requestFrom(9, 16);       // request 16 bytes from slave device #9
  while (Wire.available())   // slave may send less than requested
  {
          data[t] = Wire.read(); // receive a byte as character
          if (t < 15)
          t++;
          else
          t = 0;
  }
  Serial.print("data[0]:");
  Serial.println(data[0]);
  Serial.print("data[2]:");
  Serial.println(data[2]);
  Serial.print("data[4]:");
  Serial.println(data[4]);
  Serial.print("data[6]:");
  Serial.println(data[6]);
  Serial.print("data[8]:");
  Serial.println(data[8]);
  Serial.print("data[10]:");
```

```
  Serial.println(data[10]);
  Serial.print("data[12]:");
  Serial.println(data[12]);
  Serial.print("data[14]:");
  Serial.println(data[14]);
}
float getDistance(){
long duration;
float cm;
float inches;
digitalWrite(triggerPin, LOW);
  delayMicroseconds(2);
  digitalWrite(triggerPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(triggerPin, LOW);

  /* The echo pin is used to read the signal from the PING))): a HIGH
  pulse whose duration is the time (in microseconds) from the sending
  of the ping to the reception of its echo off of an object.*/

  duration = pulseIn(echoPin, HIGH);

  // convert the time into a distance
  cm = float(duration) /29.0 /2.0;
  inches = cm /2.54;

  Serial.print(inches);
  Serial.println(" inches");

  return inches;
}
int getColor(){


}
```

V1.1 Servos for Camera (Semester One)
Language: C++

```
#include <Servo.h>
Servo myservo;
Servo moservo;
String inByte;
String onByte;
String dir;
String ver;
int pos = 90;
int pis = 25;
void setup() {

  myservo.attach(9);
  moservo.attach(10);
  myservo.write(pos);  //initialize servos
  moservo.write(pis);
  Serial.begin(115200);
}
void loop()
{
  if(Serial.available())  // if data available in serial port
          {
          inByte = Serial.readStringUntil(','); // read data until newline
          onByte = Serial.readStringUntil('\n');
          dir = inByte;   // change datatype from string to integer
          ver = onByte;
          if(dir == "left"){
          pos += 5;
          }
          if(dir == "right"){
          pos -= 5;
          }
          if(ver == "up"){
          pis -= 5;
          }
          if(ver == "down"){
          pis += 5;
          }
          myservo.write(pos);    // move servo
          moservo.write(pis);
          Serial.print("Servo1 in position: ");
```

```
            Serial.println(pos);
            Serial.print("Servo2 in position: ");
            Serial.println(pis);
            }
}
```

V1.1 Object Tracking Camera (Semester One)

Language: Python

```
from collections import deque
from imutils.video import VideoStream
from picamera.array import PiRGBArray
from picamera import PiCamera
import numpy as np
import argparse
import cv2
import imutils
import time
import serial
arduino = serial.Serial('/dev/ttyACM0', 115200)
horiz = ""
vert = ""
# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-v", "--video",
	help="path to the (optional) video file")
ap.add_argument("-b", "--buffer", type=int, default=32,
	help="max buffer size")
ap.add_argument("-f", "--fps", type=int, default=32, help="Framerate")
ap.add_argument("-x", "--horizontal", type=int, default=480, help="Horizontal screen size")
ap.add_argument("-y", "--vertical", type=int, default=360, help="Vertical screen size")
ap.add_argument("-s", "--size", type=int, default=8, help="Object Size")
args = vars(ap.parse_args())
# define the lower and upper boundaries of the "green"
# ball in the HSV color space
greenLower = (29, 86, 6)
greenUpper = (64, 255, 255)
# initialize the list of tracked points, the frame counter,
# and the coordinate deltas
counter = 0
(dX, dY) = (0, 0)
direction = ""
screenx = args["horizontal"]
screeny = args["vertical"]
# if a video path was not supplied, grab the reference
# to the webcam
if not args.get("video", False):
	print "grabbing video"
	#vs = VideoStream(src=0).start()
	camera = PiCamera()
```

```
        camera.resolution = (screenx, screeny)
        camera.framerate = args["fps"]
        rawCapture = PiRGBArray(camera, size=(screenx, screeny))
# otherwise, grab a reference to the video file
else:
        vs = cv2.VideoCapture(args["video"])
# allow the camera or video file to warm up
time.sleep(2.0)
# keep looping
for frame in camera.capture_continuous(rawCapture, format="bgr", use_video_port=True):
        frame = frame.array
        # handle the frame from VideoCapture or VideoStream
        #frame = frame[1] if args.get("video", False) else frame
        # if we are viewing a video and we did not grab a frame,
        # then we have reached the end of the video
        if frame is None:
        print "broke while loop"
        break

        # resize the frame, blur it, and convert it to the HSV
        # color space
        #frame = imutils.resize(frame, width=600)
        blurred = cv2.GaussianBlur(frame, (11, 11), 0)
        hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)
        # construct a mask for the color "green", then perform
        # a series of dilations and erosions to remove any small
        # blobs left in the mask
        mask = cv2.inRange(hsv, greenLower, greenUpper)
        mask = cv2.erode(mask, None, iterations=2)
        mask = cv2.dilate(mask, None, iterations=2)
        # find contours in the mask and initialize the current
        # (x, y) center of the ball
        cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
        cv2.CHAIN_APPROX_SIMPLE)
        cnts = cnts[0] if imutils.is_cv2() else cnts[1]
        center = None
        # only proceed if at least one contour was found
        if len(cnts) > 0:
        # find the largest contour in the mask, then use
        # it to compute the minimum enclosing circle and
        # centroid
        c = max(cnts, key=cv2.contourArea)
        ((x, y), radius) = cv2.minEnclosingCircle(c)
```

```
		M = cv2.moments(c)
		center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))
		# only proceed if the radius meets a minimum size
		if radius > args["size"]:
		# draw the circle and centroid on the frame,
		# then update the list of tracked poi
		centerx = int(x)
		centery = int(y)
		if(centerx > (screenx*2/3)):
			print("right")
			horiz = "right"
		elif((centerx >= (screenx/3)) and (centerx <= (screenx*2/3))):
			print("horizontal center")
			horiz = "center"
		else:
			print("left")
			horiz = "left"
		if(centery > (screeny*2/3)):
			print("down")
			vert = "down"
		elif((centery >= (screeny/3)) and (centery <= (screeny*2/3))):
			print("center")
			vert = "center"
		else:
			print('up')
			vert = "up"
		arduino.write(horiz + "," + vert + "\n")

		# show the frame to our screen and increment the frame counter
		#cv2.imshow("Frame", frame)
		#key = cv2.waitKey(1) & 0xFF
		counter += 1
		# if the 'q' key is pressed, stop the loop
		#if key == ord("q"):
		#   break
		rawCapture.truncate(0)
# if we are not using a video file, stop the camera video stream
if not args.get("video", False):
		vs.stop()
# otherwise, release the camera
else:
		vs.release()
```

Next Steps and Future Improvements

The next steps for the project involve connecting the car to the existing object tracking camera and have the two pis be able to communicate. In addition, the object tracking camera will be able to record objects it sees and keep a record of them. Also, if the camera detects an object as it pans left and right, it will detect objects in front of the car, and depending on what the object is, the camera will send signals to the car and the car will perform various actions.

Another thing i want to do with the car is be able to train its autopilot. The car has the capability to collect data and process it into different “tubs.” Enough tubs of data can program an autopilot, which will follow the route quite accurately.

Lastly, I want to include more sensors to allow the car to track its position relative to when it started, likely using an Inertial Reference System or similar device. A GPS could also be viable, as well as sensors to avoid objects like LIDAR, similar to what Tesla uses for their self-driving car software.

Here is the future improvement plan for my rover V2: I would like to continue developing this car to integrate more sensors and cameras to enable more autonomous functionality:

- Stop sign and traffic light detection
    - Perform action if camera detects stop sign/traffic signal and is within 15 degrees of center
- Follows rules of traffic lights & stop signs
- Keep a record of objects tracked. Build “map” of what objects were seen where (relative to starting position (IRU))