SDC Doc

Johar Samrao

- Hi, I am Johar Samrao. I am a senior at Leland High in San Jose, CA. Today I am going to share the self driving car that I have built.
- I started working on this project back in April of this year. Though I put together and assembled the hardware for the car, most of my effort was focused on developing the software that drives the car and fine tuning the algorithms.
- So here is an overview of what I am going to cover: First I am going to introduce the car hardware, then we will go over the software and algorithms, followed by the actual demo. Finally, I will go over the future directions.

- RASPBERRY PI
- Well here is the self-driving car.
- First lets go over the Raspberry Pi card. It’s version 3. This is the miniature computer that makes up the brain of the car. This is where all the software resides and runs.
- Raspberry Pi is based on this Broadcom SoC chip. It has quad ARM processor cores built in that run at 1.2 GHz. These cores run the Raspbian operating system, which is a version of the Linux operating system.
- It’s little hard to see, but here is an SD card which makes up the storage system of this computer. This is where the OS and the software that I developed resides.
- Broadcom SoC comes with in-built WiFi, which is really nice.
- Here are the USB ports. And here is the Ethernet port.

- GPIO
- Here is the GPIO connector, which stands for general purpose IO.
- Various components on the car are connected to Raspberry Pi via this connector.
- You can program the GPIO as either inputs or outputs.
- The software drivers use these GPIO pins to communicate with all the components, which include the DC motors and the servo motor.

- CAMERA
- Here is the camera module. It is facing down. Its connected to Raspberry Pi via the USB port. The resolution of the camera is 1080P, though I am using only 480x320 because anything more than that brings the performance down drastically. The camera has a 120 degree viewing angle. I learned the hard way that anything less than that is very hard to work with. This is actually the fourth camera that I have selected for this project, that finally did work.

- PWM DRIVER
- Here is 16 channel PWM driver. PWM stands for pulse width modulation. The servos and the DC motors are controlled via PWM. For example, by controlling the duty cycle of the pulse, you can move the servo in one direction or the other. Similarly, by adjusting the duty cycle of the pulse, you can change the speed of the car.
- There is one channel from the GPIO to the PWM driver. And from there you can select one of the 16 channels to drive at a time.

- DC MOTORS
- Here are the two DC motors that actually move the car.
- There are two types of controls to these motors: to move forward or backward which is done by programming the internal configuration registers. Here is the connection to the GPIO that is used to configure these registers.
- The second control to the motors is the speed control. This is done via PWM. Here is the connection from the PWM driver to the the DC motors.
- STEP DOWN DC-DC CONVERTOR
- This is the step down DC-DC convertor module. It's job is to step down the 7+ volts supply from the batteries down to 5 volts that is needed for the Raspberry Pi.

- SERVO MOTOR
- Here is the servo motor which is used to steer the car left and right.
- Servo motor provides very precise and controlled movement in each direction.
- It is connected here to the PWM driver, which means steering is controlled using PWM.

- LAPTOP
- So finally on how do I connect my laptop to the car. I need to connect my laptop to the Raspberry Pi so I can develop, run, and debug the code.
- First of all Raspberry Pi and my laptop are on the same WiFi network. So Raspberry Pi has an IP address assigned to it by the router. I have the VNC software on my laptop, which allows me to connect to the Raspberry Pi via its IP address.
- Once I start the VNC session and log in, I get the Raspbian windows interface. From here I can open a terminal window, which allows me to develop code directly on the Raspberry Pi. All the code is saved on the SD card in Raspberry Pi.

- Software - Python
- OK, lets switch gears to the software.
- As I mentioned, the operating system and all the code and the drivers that I developed, and all the libraries all sit on the SD card in the Raspberry Pi SD card.
- I used the Python programming language to do all the coding. The clear advantage for me was the convenience factor. I could make quick changes to the code and see their effect without having to compile. Same goes with debugging or if you make a mistake, you don’t have to go through the compile step.
- The drawback is that it is slower. If I used C or C++ or Java, a lot higher performance would be possible. The car has to process images of the track to find lanes. With higher performance, you can process more frames per second, which leads to a more reliable lane detection, car’s responses to the changes in the track or noise in the image are quicker, and therefore, you can run the car at a higher speed.

- Software – Flow 1
- Ok, lets go over how the car’s software operates.
- The program is written using object-oriented methodology. Vast majority of the code is contained in just one class: sdcar, which stands for self driving car. Public methods in this class are called repetitively in a loop to make the car run.
- So let’s go over what’s actually done in this loop.

- Software – Flow 2
- The first step is to capture a frame or image of the track in front of the car using the camera.
- Then this color frame is converted to gray scale with a resolution of 480x320.
- Following that, frame is analyzed to detect edges. Then edges are analyzed to detect lines. These functions are performed using OpenCV library.
- Following that, my algorithm kicks in to detect lanes, and drive and steer the car. Developing and perfecting this algorithm is where most of my time went into on this project.

- Software – Algorithm 1
- The algorithm has two parts:
- First part is Lane Detection. The challenge here is to extract lanes accurately out of a frame that contains all kinds of lines in all directions and noise. For example, shadows of objects around the track and glare all show up in the frame as lines. Main job here is to filter out all the noise – meaning anything that is not a lane – so you can zero in on the true lanes out of the maze. This filtering process uses multiple filters. Output of this phase of the algorithm are two lines that are supposed to be lanes.

- Software – Algorithm 2
- Second part of the algorithm is the Turn Vector Generation.
- This phase starts with the one or two lanes detected by the first phase. From these lanes, it generates a Turn Vector, which is a vector line that originates from the mid-bottom of the frame and points in the direction that the algorithm believes the lanes are going in. Frame by frame, this turn vector is adjusted based on how the track in front of the car looks like and the direction it is leading to.

- Software – Algorithm 3
- Third part of the algorithm is the Steering Control.
- Once the turn vector is updated for a frame, basic trigonometry is used to find out the angle it is pointing at, and then that angle is used to steer the car.
- One of the challenges here is to effectively deal with the instantaneous noise in the turn vector. A perfect shadow or a glare in the frame can look like real lanes and from time to time algorithm can be fooled. To minimize the effects of such errors, the algorithm uses hysteresis and thresholding. These techniques make sure that once in a while wild changes in the turn vector don’t take the car into the woods.
- Second challenge is that it is possible to detect only one lane in a frame because the car drove too close to one lane and the other lane dropped out. To prevent problems in such situations, the algorithm maintains state and remembers how the lanes used to look in the immediate past to use that as the guidance.

- Software Final
- So this loop to acquire the frame, process it, detect lanes, and then steer the car, is performed as fast as possible. Faster this loop is run, higher the frames processed per second, and higher the speed at which the car can respond to the changes in the track, and therefore, faster the car can run.

- Demo
- OK, now lets get to the demo.
- First phase of the project was developed on a straight stretch of track. Second phase was developed on an all-circular track.
- For this demo, I have chosen an oval track which has both the straight and circular stretches of the track.
- Here is the track. It is ?? feet long and ?? feet wide.
- I will put the car down on the track and initiate the execution of the program on the car from my laptop. The car will start running, drive itself for ?? seconds/minutes, and then stop.

- Lane Detection / Turn Vector Window
- OK, now let me give you a feel of how the car and the algorithm sees the track and how it is making sense of everything that it sees out there.

- Need to show: lanes in the image as captured by camera, ALL the lines shown by OpenCV, the lanes detected by algorithm, and the turn vector.
- Car should first be put on the straight part and then on the curvy part. Car should be moved to sides and the lane detection and the turn vector should be demo'ed.

- Lane Detection / Turn Vector Window
- Ok, so we have the car on a straight track.
- On my laptop window you are seeing a live video of what exactly the software is seeing and generating. This window is sort of the eye view of the software.
- Even though the car is not running, it is capturing the frames of the track and processing them.
- The black lines in the video are the actual lanes of the track.

- Lane Detection / Turn Vector Window
- The ragged red lines are the edges and lines found open CV. These red lines are fed to my algorithms.
- The green lines are the lanes detected by the algorithm. That’s where the algorithm thinks the lanes are.
- I have put a screwdriver on the track and you can see as red lines in the middle of the track. These red lines are being rejected by the algorithm as it doesn’t think that’s a lane.

- Lane Detection / Turn Vector Window
- The purple line in the middle is the turn vector. Based on where the green lanes are detected, the turn vector will point in the direction that car should be steered in.
- You can probably hear the car’s steering servo make a little noise as it is trying to make little steering adjustments.
- (Let’s remove the screwdriver from here.)

- Lane Detection / Turn Vector Window
- OK, I am going to turn the car to left. Watch out how the camera feed of the actual track lanes changes and how the software reacts and produces the detected lanes. The turn vector will also change.
- (TURN CAR LEFT. WAIT FOR 5 SECONDS)
- Now let’s turn the car to the right. Again, let’s pay attention to the how the software reacts.
- (TURN CAR LEFT. WAIT FOR 5 SECONDS)

- Lane Detection / Turn Vector Window
- Alright, now let’s move the car to the curved portion of the track. And let’s see if the car gets ready to take off from the curved section.
- Let’s pay attention to the laptop screen.
- (MOVE CAR TO CURVED PATH.)
- As you can see, the green lanes are now being detected at an angle. The turn vector is also pointing at an angle, indicating the direction of steering.
- If you look at the wheels of the car, they are turned in the direction of the turn vector.
- So the car is ready to go in the right direction.