

Wireless Autonomous Car

COURSE PROJECT ENGR 844



San Francisco State University, SF, CA, 94132

Wireless Autonomous Robot (WAR)

Luis L. Bill¹, Ravi Teja Mamidipaka², Siddharth Jankar³
San Francisco State University, SF, CA, 94132

As part of our proposed project solution, our group will modify an RC toy car and convert it into our test platform for the project. The RC Toy Car will be modify to use an off shelf motor driver, which will allow us to control the direction of the movement of the car, as well as the speed. Then, the Tiva C Series LaunchPad will be installed on the RC Car Toy, and it will be used as the main brain of the RC Toy Car. The communication will then be obtained through wireless connectivity. The group proposes wireless connectivity because a GUI (Graphical User Interface) is going to be used for both, control the car manually, as well as receive and visualize data coming from the LaunchPad. For this task, our group will use Kivy, a Python based GUI library. It allows for easy implementation of TCP/IP or XML-RPC communication (communication protocols that can be used with Wi-Fi).

I. Introduction

We proposed a project to operate an RC car wirelessly using a local wireless network with a router. The idea comes into the mind after research done on wireless and autonomous driving technology. Recently the news about Google self-driving car motivated us to work on a car having an autonomous drive mode and a wireless communication feature. Autonomous technology on cars will help further increase driving safety. And yet, wireless communication for data transfer between cars is one important feature for these cars. It'll allow cars to communicate between each other, and transfer meaningful data while driving autonomously.

The LaunchPad that we are using is from TI family TM4C1294 and to make the robot wireless we are using SimpleLink Wi-Fi BoosterPack cc3000. The car has two operating modes: manual and autonomous. Autonomous mode is achieved using IR sensors on the front left and right side of the robot. The sensors are measuring the distance from the robot to a detected obstacle. Depending of the proximity to an object the car will turn left or right, or drive straight. The Graphical User Interface (GUI) to the car is a python based library named Kivy. The car uses the FuelTank BoosterPack to power the MCU, sensors, and Wi-Fi module. We are using one more battery which is 9V to drive the motors. The RC car with 3 DC motors has two motors for driving forward/backwards, and one motor to turn left/right. The motors are driven by the motor driver circuit with 9V supply to it.

II. Design of the project

A. Specifications

For the implementation of this project we have several implementation requirements, not only as part of the class, but also as part of the objective to be accomplished. Robotics systems are different from other software related projects. Specifically, robots must have the ability to obtain data from its environment, analyze data in real time, and proceed to react (generally) quick enough. That is the reason why for this project we needed several core components, including:

1. ARM Cortex-M4 (TivaWare LaunchPad 1294x1) for fast processing.
2. A test platform (such as a modifiable RC car toy).
3. A medium to communicate with the robot to debug and log data (WI-FI).
4. A way to control the robot manually (we can use a Graphical User Interface).
5. A power source, mainly rechargeable and portable, so that the robot is free to move (FuelTank).
6. And sensors, which allow the robot to freely interact with an often dynamic environment (IR sensors).

¹ Robotics Engineer, Electrical Engineering, Embedded Electrical and Computer Systems, SFSU.

² Controls Engineer, Electrical Engineering, Embedded Electrical and Computer Systems, SFSU.

³ Systems Engineer, Electrical Engineering, Embedded Electrical and Computer Systems, SFSU.

The following figure demonstrates the system architecture:

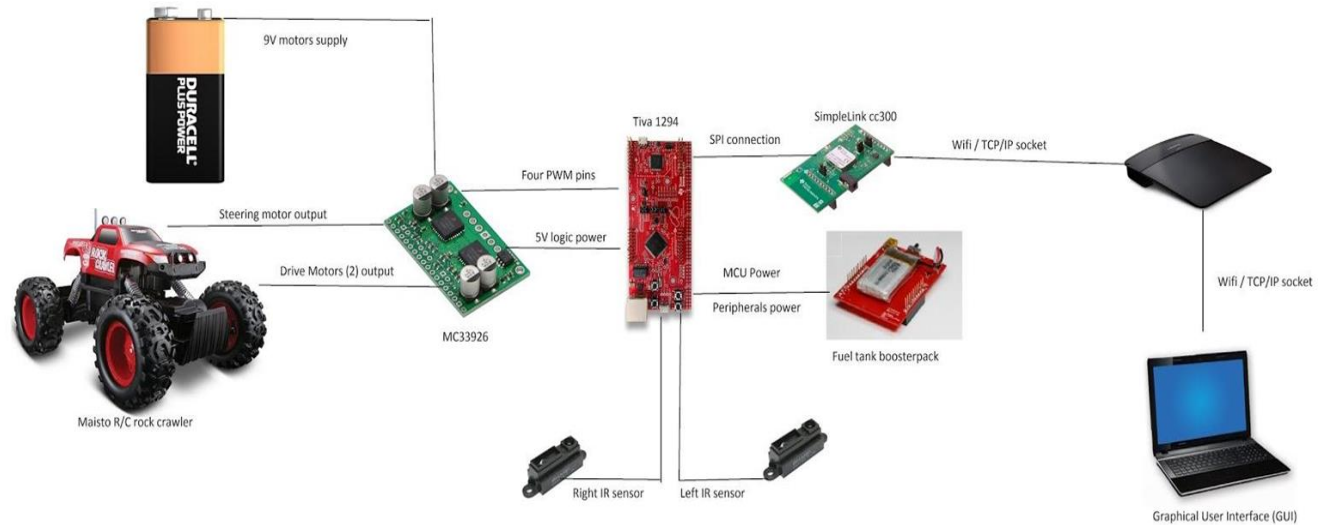


Figure 1: System architecture of the Robot

The system architecture includes a motor controller which allows the MCU to send PWM commands to the motors. Moreover, there are two power sources: a 9V battery for the RC car connected through the motor controller, and a FuelTank BoosterPack to power the rest of the components (MCU, sensors, etc.). Also, a Wi-Fi router is used as the main communication hub between a client (Graphical User Interface) and the server (MCU).

B. Graphical User Interface (GUI)

The project involved developing a GUI that would allow a user to control the car manually (move forward, backwards, etc.), and switch the car to autonomous mode as well. Since the only mean of communication for the robot is the SimpleLink BoosterPack, a GUI is necessary in order to control the robot in case of an emergency, and it's also necessary for debugging purposes (data logging and robot monitoring). The GUI was developed using the Kivy open-source GUI library. Kivy is written in python, and the fact that it is written in python allows Kivy to run in different operating systems (including Android and IOS). In this project the operating systems of choice are Windows and Ubuntu.

The GUI allows a user to perform the following functions:

- A. Initiate a TCP/IP socket to communicate with the car through Wi-Fi.
- B. Once connection is started, the user can manually move the car.
- C. There are three “speed values” to control the PWM of the car to make it move faster or slower.
 1. Velocity 0.
 2. Velocity 1.
 3. Velocity 2.
- D. The GUI displays data being transmitted from the car (e.g. distance sensor data in centimeters).

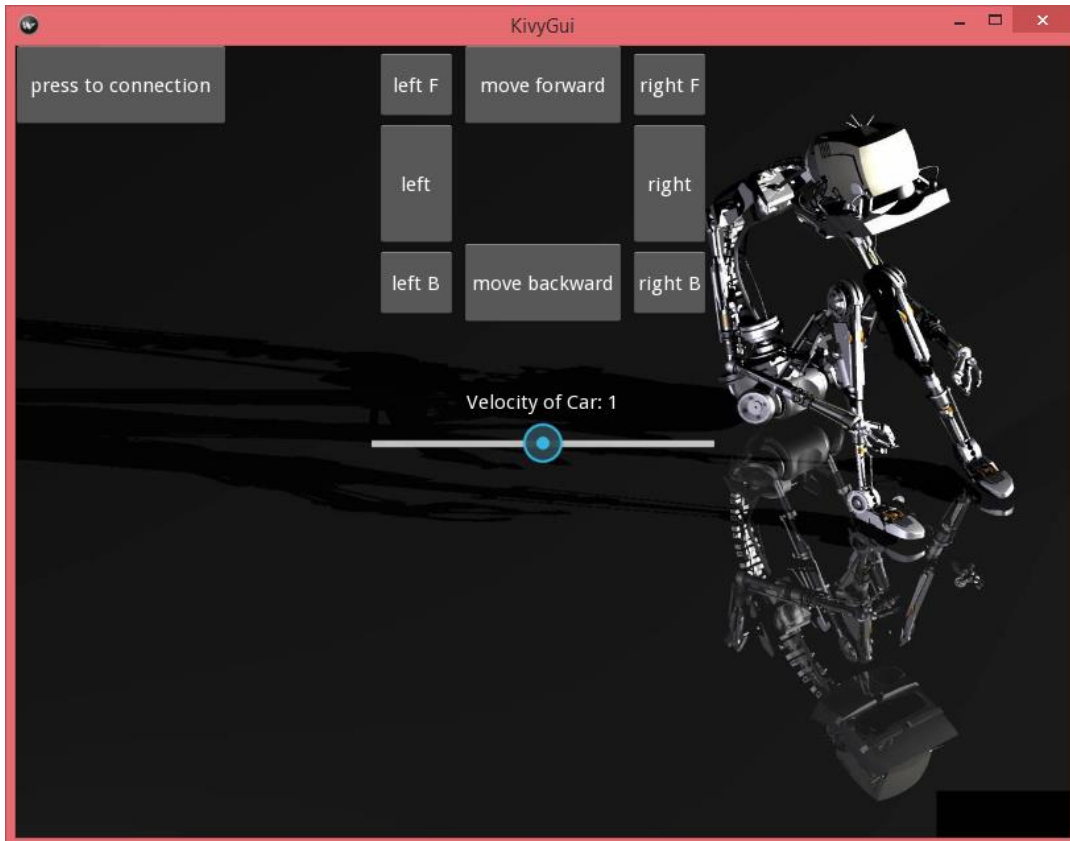


Figure 2. Graphical User Interface (GUI). *This GUI has widgets that allow a client to connect to a robot, control the robot manually or turn on autonomous mode, receive and display data.*

Below is the flow diagram of the GUI:

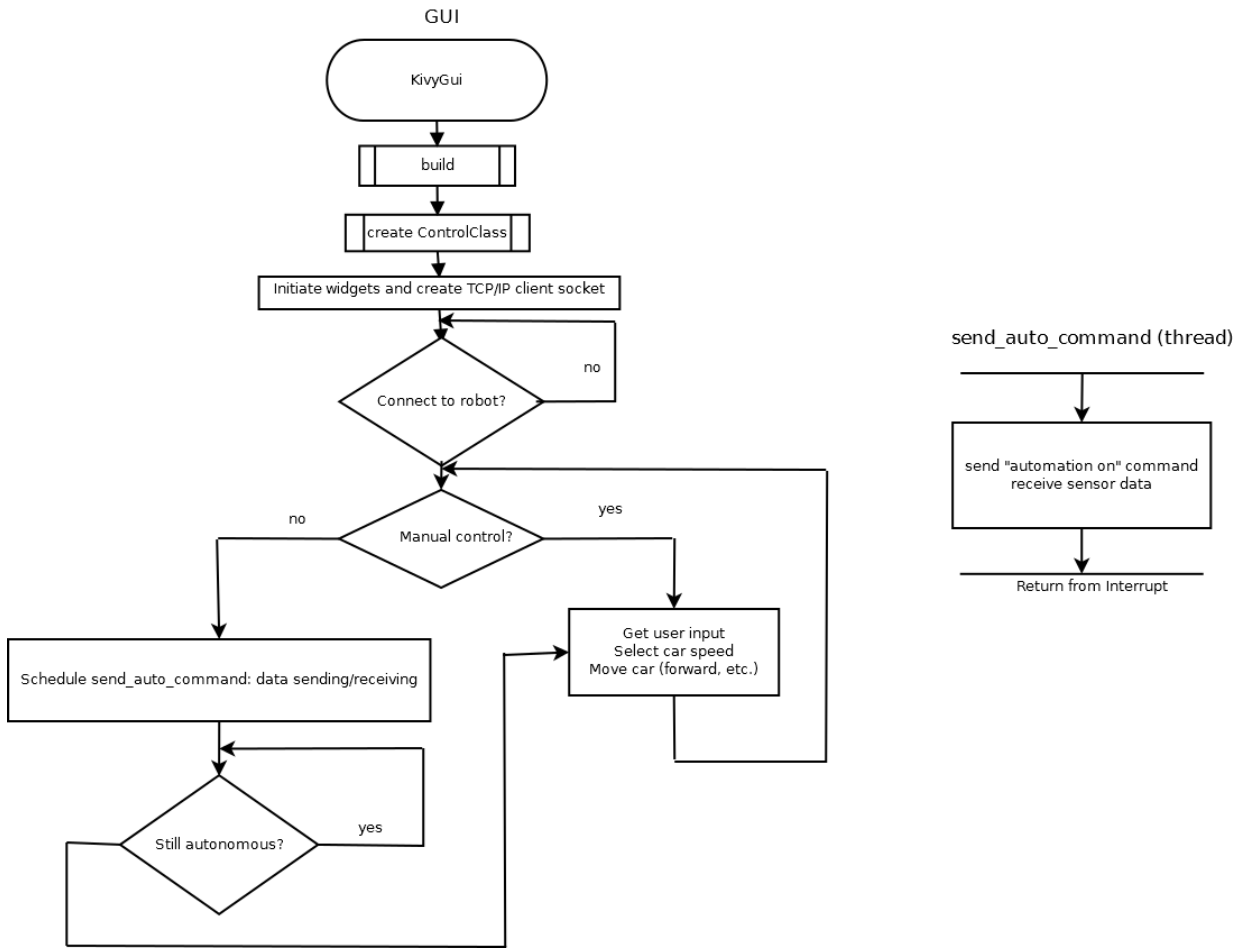


Figure 3. GUI Flowchart. The flowchart depicts the way the GUI works, from connecting to a socket, to communicating with the robot, and also running a thread that sends and receives commands to/from the robot.

When the GUI starts, the user won't be able to send nor receive commands. To initiate the two-way communication, the user must press the "press to connect" button to connect the socket on the client side (laptop) to the socket on the server side (Tiva C LaunchPad MCU). The connection is made through a predefined server IP address and port. The IP address of the server is generally assigned through DHCP, but could also be static. Once this connection is opened, the user can send a data stream of up to 255 bytes at a time (each character on a stream is equivalent to 1 byte). This data stream can be anything from movement commands to car velocity and sensor data.

On the GUI there is also the "autonomous mode" button which allows a user to switch between autonomous mode and manual mode. In autonomous mode, the ADC conversion is use in order to obtain data from the IR sensors. This data is then sent to the GUI for displaying. The data stream sent represents the distance in cm to an object detected near the car.

C. ADC with Sensors and Obstacles Avoidance

The autonomous car built contains an array of Infra-Red (IR) sensors used in order to detect obstacles close the robot. The robot uses two 2Y0A21 F 43 SHARP IR sensors set 90 degrees apart from each other (45 degrees from the robot's center line). These sensors can transmit data at a rate of ~25Hz, but for the current project data is being sampled at only 10Hz. The IR sensors can be powered with 3.3V up to 5V.



Figure 4. Infra-Red sensor. *Sample IR sensor.*

The IR sensors provide a voltage reading based on the time of flight of the light return after an obstacle has been detected. The farther an obstacle is, the lower the voltage reading. Yet, the sensors the car has are only accurate between 4cm to 80cm. Outside of this range the data is not useful (e.g. too much noise and inaccurate readings appear).

The distance-to-voltage relationship was found empirically by manually recording the voltage reading when an obstacle was positioned in front of a sample sensor. The data is not linear, and it reaches a maximum voltage reading at a distance of 5cm.

Now, even though this is a very accurate description of the voltage as a function of distance, it is quite hard to obtain the exact equation of this graph. Therefore, we can linearize the data by plotting voltage as a function of inverse distance ($1/\text{distance}$) as on Figure 5.

The distance has been inverted, which allows the robot to obtain a linear approximation of the data. The linear approximation works well enough for short distance (e.g. detecting objects within 50cm of the robot). Also, since we are mostly interested in obtaining the distance reading, we can plot the inverse of the distance as a function of voltage. In this case, the voltage is the input, and we obtain the distance as the output.

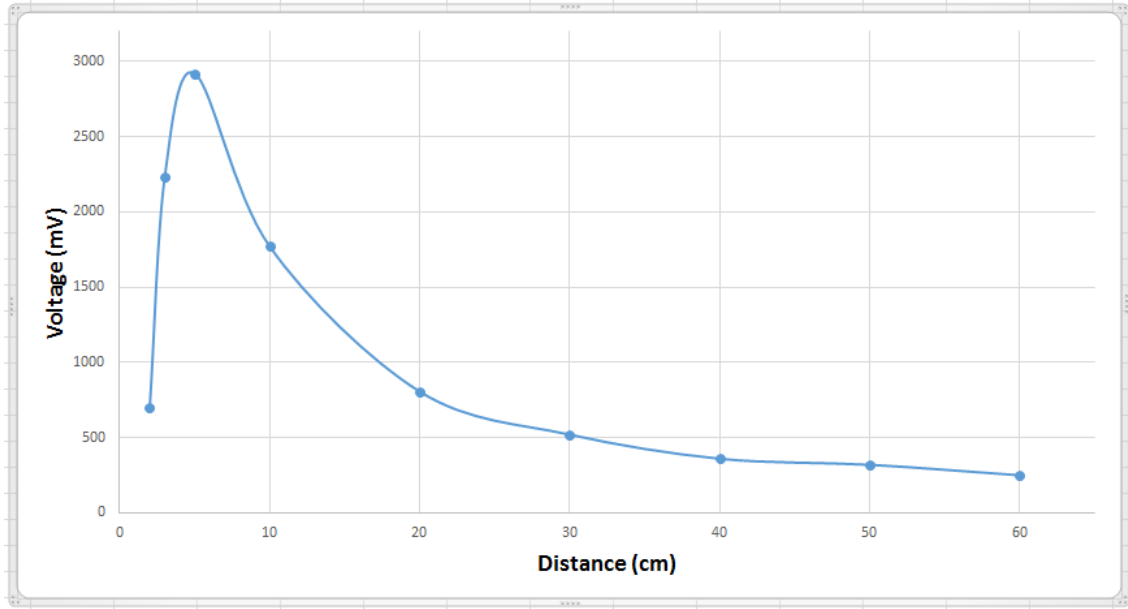


Figure 5. Voltage as a function of measured distance to obstacle. Below 5cm the data is not useful, so we measure distances starting from 5cm and up.

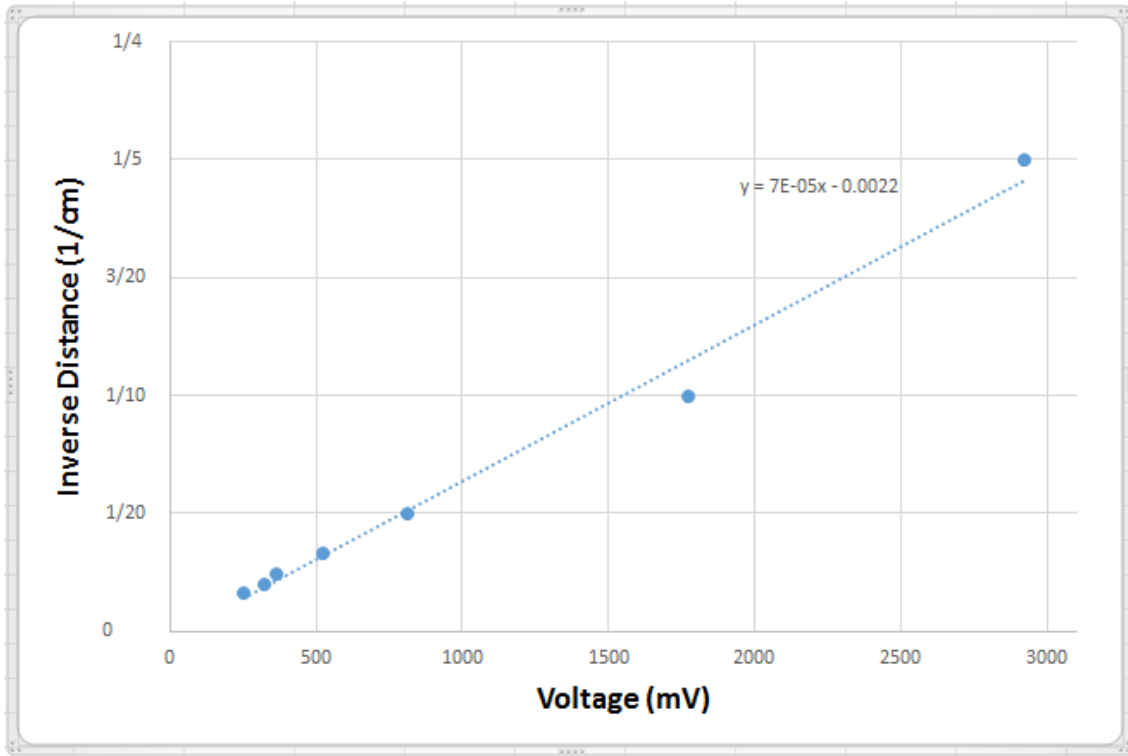


Figure 6. Inverse of distance as a function of Voltage. BA linearized version of the graph is easier to work with.

The robot is using ADC0 as the main module to obtain the sensor readings, along with Sample Sequencer 0. Only two channels are being used (a channel per sensor). The data is being sampled at a speed of 10Hz, which is sufficient data to detect and avoid obstacles. Timer 0 is used as the trigger for the ADC conversion. After this data is converted, it is grabbed from the FIFO0, and store in an array. Each element in the array is corresponds to the data sample from each sensor (each channel). After gathering the data, a difference between sensors is obtained:

```

Grab data from FIFO 0 and store data into adc_conversion [2]
LSensor = adc_conversion [0];
RSensor = adc_conversion [1];
if (LSensor < 30 or RSensor < 30)
    Difference = RSensor - LSensor
    if Difference < 0
        Turn Robot left
    if Difference > 0
        Turn robot right
    else
        Move robot straight
else
    Difference = 0
    
```

It is a very simple algorithm. If the difference it's zero, the robot shall just move straight. The following flowchart and picture show how the ADC conversion software was architected, as well as a visual interpretation of the car while avoiding obstacles:

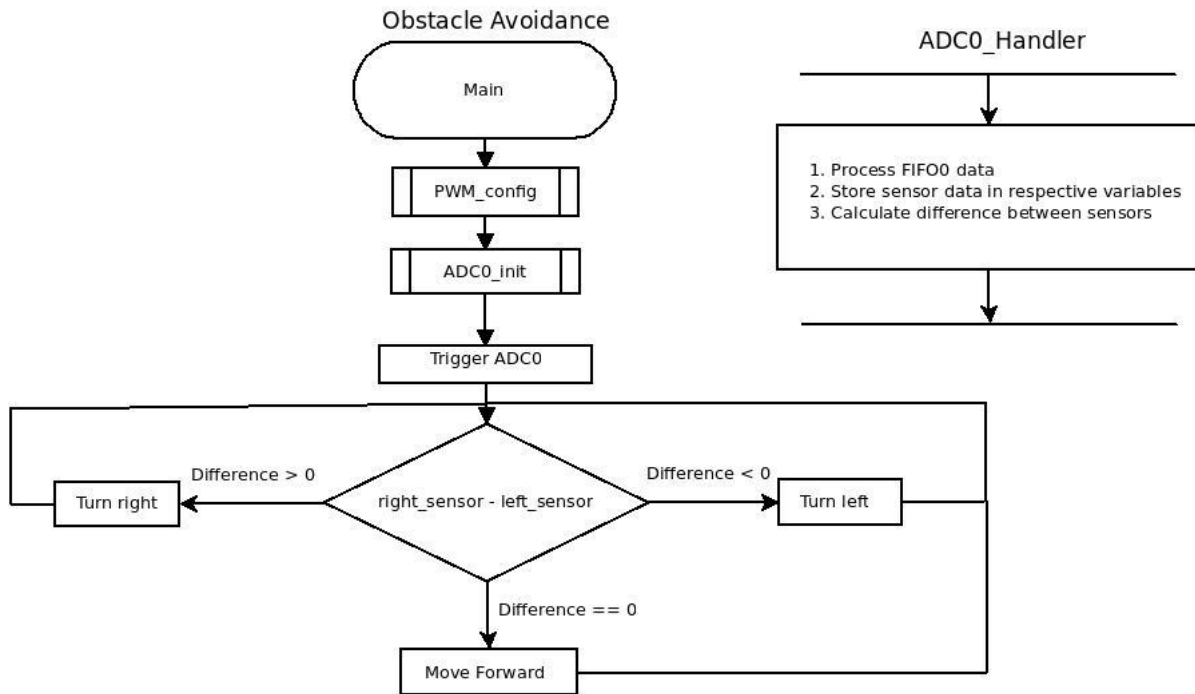


Figure 7. ADC of IR sensor data. ADC0 is used, and ADC is timer triggered, which is the most precise way to trigger the ADC.

The following image shows a simulation where if an obstacle is in front of the car, the light from the sensor obtains a shorter time of flight. This data is then analyzed to decide in which direction the car shall turn.

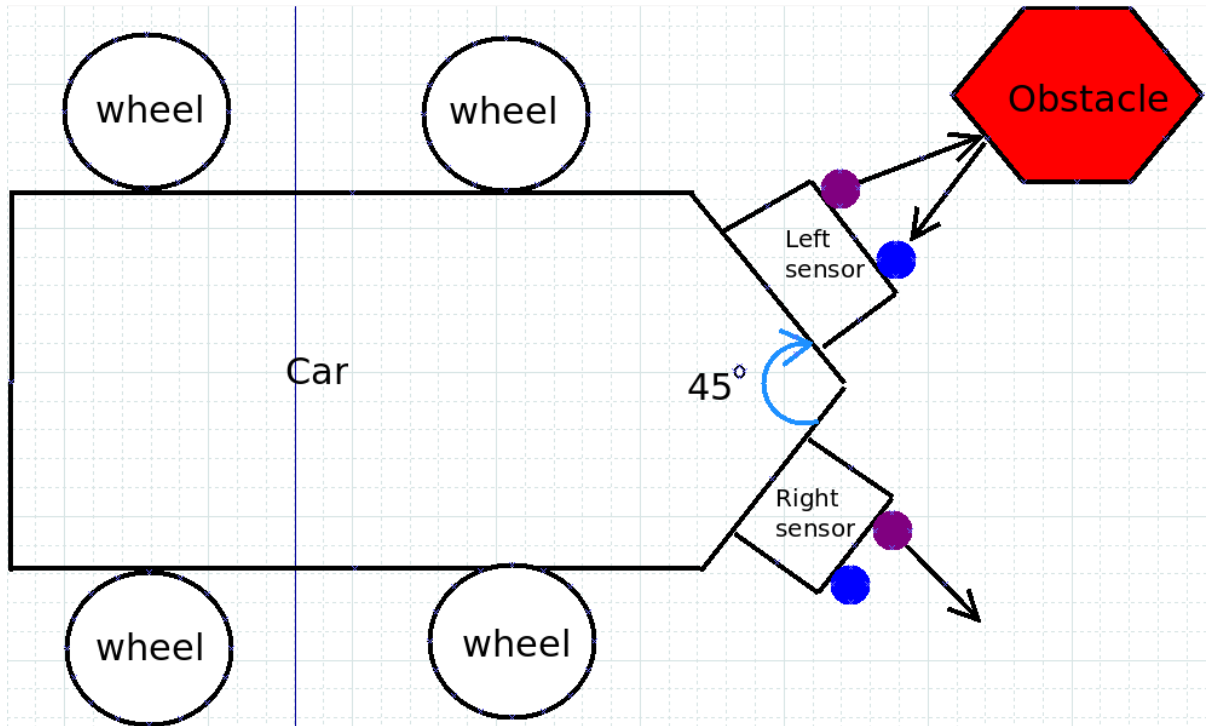


Figure 8. Car detecting obstacle. *The car will turn right, since the difference between both sensors is positive.*

D. SimpleLink Wi-Fi module cc3000

The project uses Wi-Fi communication protocol, for this we have used a Wi-Fi Booster Pack from TI family. This BoosterPack is mounted on the slot number 2 for BoosterPacks on the TM4C1294. The Wi-Fi module is self-contained wireless network processor that simplifies the implementation of the internet connectivity. It is IEEE 802.11 b/g and uses IPv4 TCP/IP protocol for the data communication by opening the sockets. Its low weight, small size and less power consumption makes it suitable to use in this project.

The cc3000 performs the following functions once the system is turned on

- A. It gets connected to the initially configured router for the communication
- B. Once connected it opens a socket and binds to the port 5005.
- C. Then it is ready to receive the data from router generated by the user through GUI.
 - a. Velocity 0.
 - b. Velocity 1.
 - c. Velocity 2.
- D. It sends the data to the GUI (the distance measured in centimeters is sent).

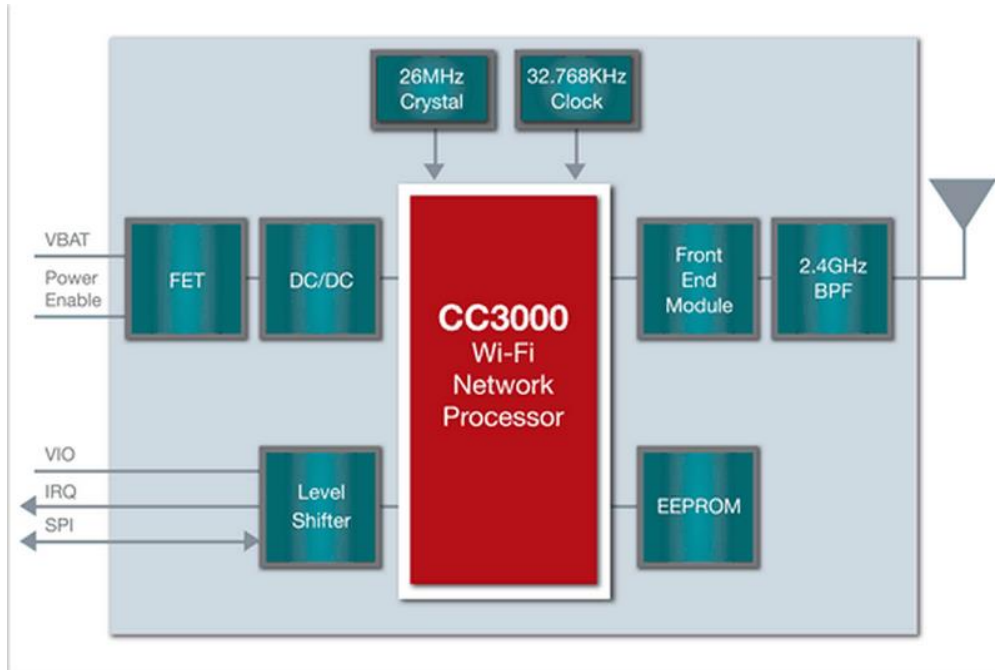


Figure 9 Showing the system architecture of SimpleLink Wi-Fi

Below is the flow chart of SimpleLink Wi-Fi communication:

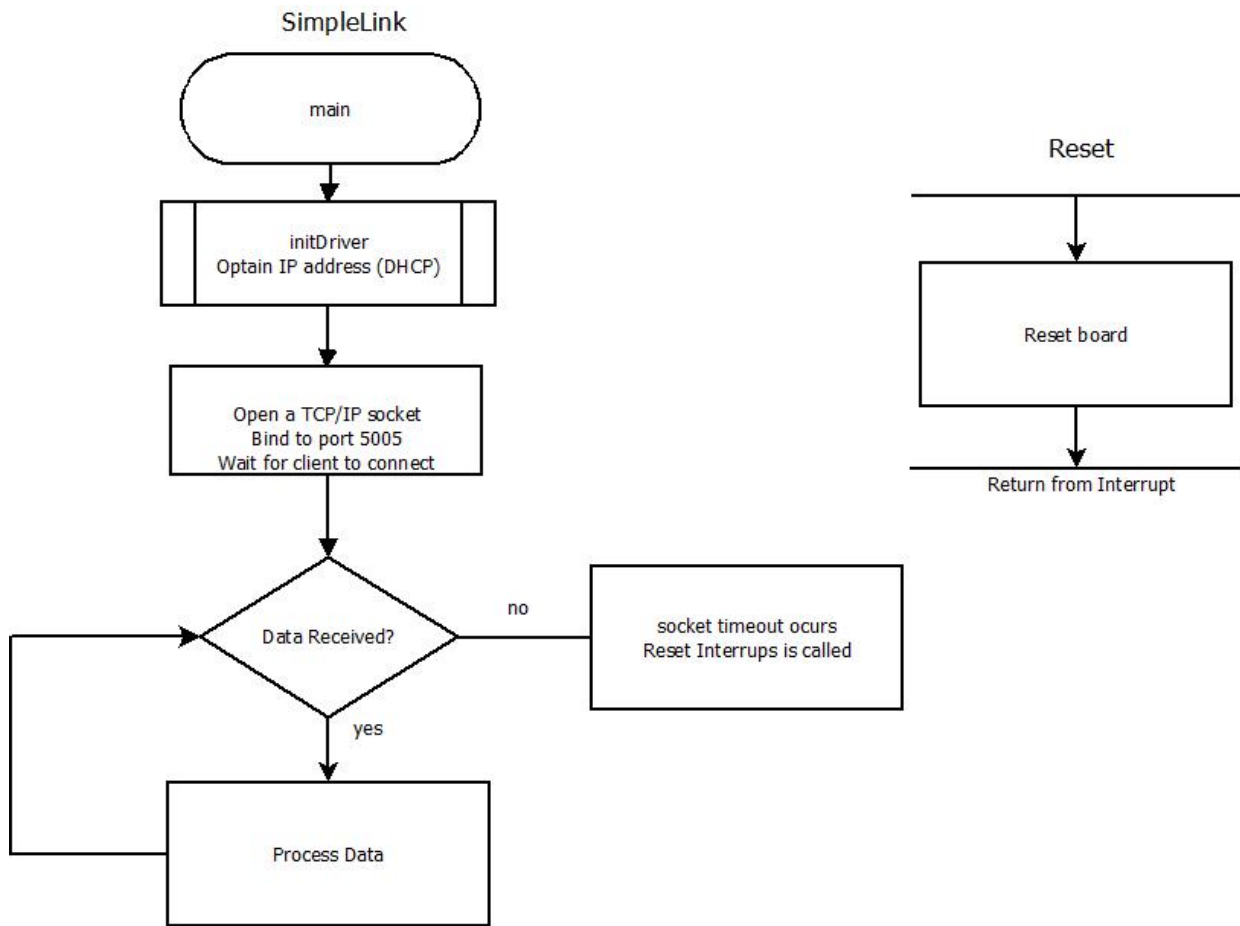


Figure 10. SimpleLink Wi-Fi Flow chart. The Flowchart depicts the way the SimpleLink works by connecting to the router, gets an IP address from router, opens a socket and binds to the port 5005. Receives the data and passes it to the Microcontroller.

When the robot is started the SimpleLink connects itself to the router. The module connects to the router to which it is configured initially. The router assigns an IP address to the SimpleLink for the TCP/IP connection.

Following figure shows the UI of the iOS application for smart config:

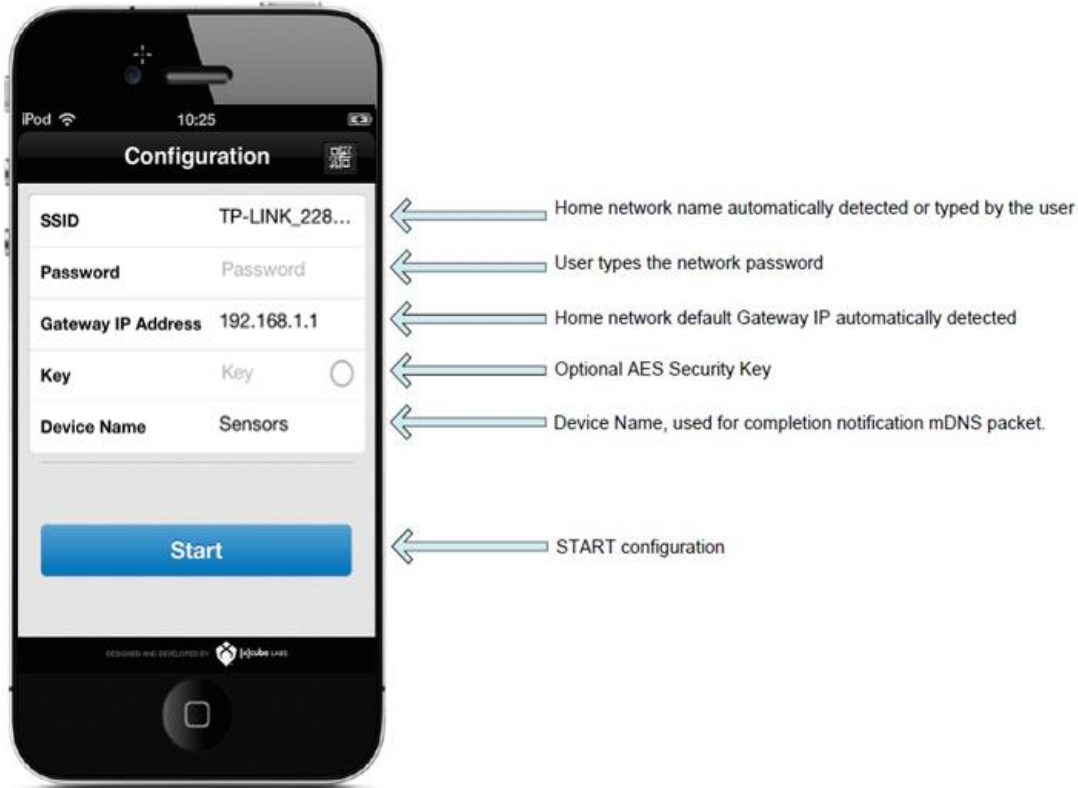


Figure 11. iOS application for smart config. *The image is taken at the time when the robot is turned on and the SimpleLink gets connected to the wireless network.*

1. Smart configuration

The TI cc3000 Wi-Fi chip has a special smart config mode, this is to allow the initial configuration of the Wi-Fi access details.

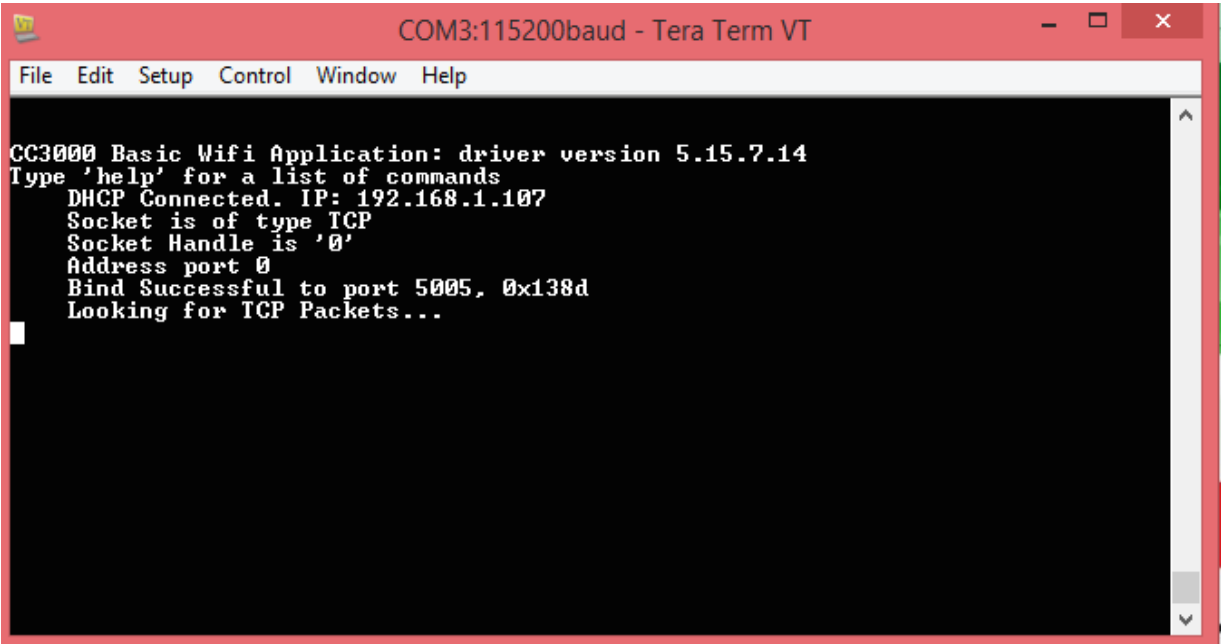
The cc3000 can be smart configured as follows,

1. The chip enters a smart config "listen" mode
2. Application on smart phone sends a "UDP" packet with access point settings
3. The chip captures this data and configures itself

There is a first time ritual which have to be performed to activate Wi-Fi module.[10]

1. Download the iOS app called smart config[11]
2. App connects itself to SimpleLink.
3. Configure the application with required fields (SSID, IP, Length, etc)
4. As the first time config is done we can see the screen below

Following is the image which shows the TeraTerm displaying the Wi-Fi operations:



```

COM3:115200baud - Tera Term VT
File Edit Setup Control Window Help
CC3000 Basic Wifi Application: driver version 5.15.7.14
Type 'help' for a list of commands
DHCP Connected. IP: 192.168.1.107
Socket is of type TCP
Socket Handle is '0'
Address port 0
Bind Successful to port 5005, 0x138d
Looking for TCP Packets...

```

Figure 12. Wi-Fi operations displayed on the Tera Term. The image is taken at the time when the robot is turned on and the SimpleLink gets connected to the wireless network.

2. Socket open

Simple opens a socket once it is connected to the wireless network. Sockets can be configured to act as a server and listen for incoming messages, or connect to other applications as a client. After both ends of a TCP/IP socket are connected, communication is bi-directional.

3. Bind port

Once the socket is opened it binds to the port 5005. The port 5005 is TCP/IP communication on the SimpleLink. The router connects to the same port for the communication between PC and the robot. Then **bind()** is used to associate the socket with the server address. In this case, the address is localhost, referring to the current server, and the port number is 10000.

4. Receive data

After binding to TCP/IP port 5005 the SimpleLink waits and gets ready to receive the data from router. When the command is sent through GUI router gives it to the robot through the port.

5. Send data

The robot is made autonomous by using two IR sensors, which measure the distance from obstacles present if any. The distance is measured in centimeters in the microcontroller. The measured distance is sent to the PC through router to be displayed on the PC.

6. Socket close

When communication with a client is finished, the connection needs to be cleaned up using **close**. Socket is closed if no commands are received by the robot for timeout period.

E. Pulse Width Modulation

7. Introduction

A variable power supply tends to use a variable resistor to control the power supply which dissipates the power as heat, which is a critical waste of the energy. In fact, PWM is an intelligent way to use electricity. As we want to control the speed of the motor one way is to use a resistor but holding back an amount of current tends to generate heat it means the system is wasting power.

A motor takes a little bit of time to respond to changes, so powering with a pulse wave which switches in between on and off limits the power to the motor, i.e. 0V and 9V; at a fast rate the motor behaves as it gets a steady voltage in between 0V-9V. We can determine this voltage by taking the amount of time our pulse is on. The rate which it is on is determined by wave's duty cycle. Duty cycle is the rate of on time of the wave.

$$\text{Duty cycle} * \text{high level} = \text{steady voltage}$$

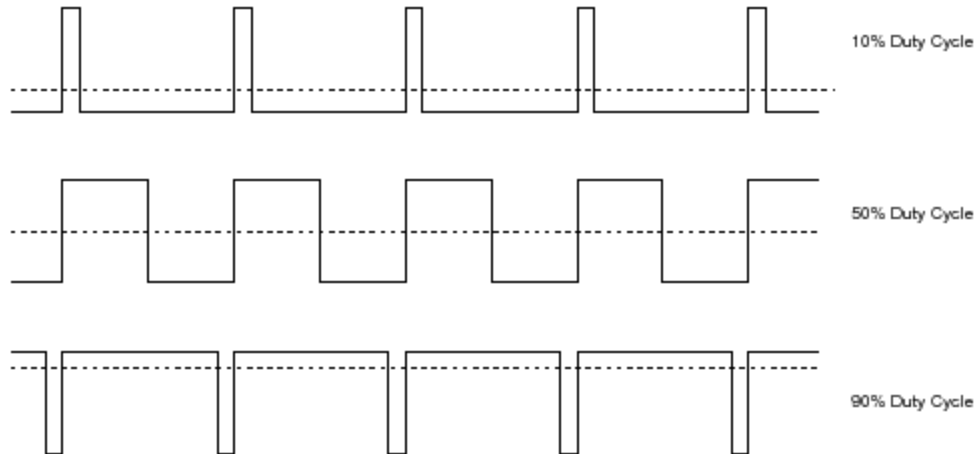


Figure 13. *Different duty cycles of PWM signal*



Figure 14. *Final robot prototype.*

8. PWM module

The Launchpad 1294XL has 4 PWM generators which generate 8 independent PWM signals. The Launchpad contains one PWM module, with four PWM generator blocks and a control block, for a total of 8 PWM outputs. It contains high resolution counters to generate square waves. And the duty cycle of the square wave is modulated to encode an analog signal [5].

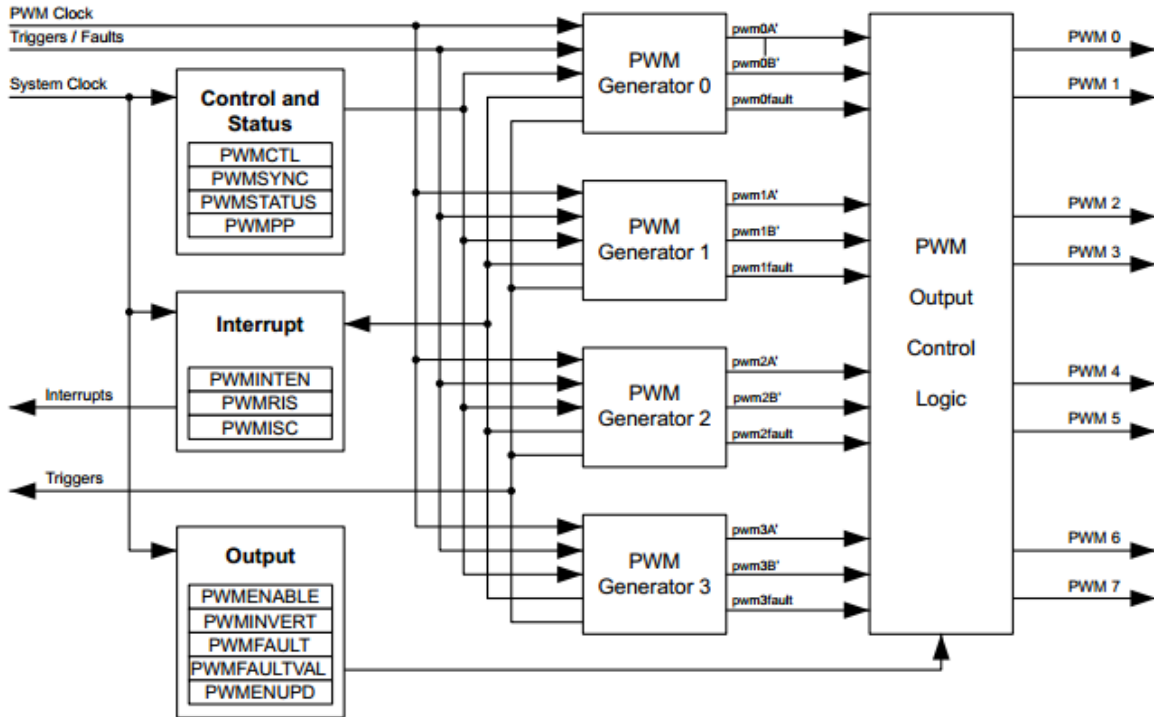


Figure15. Showing the PWM module and output signals (image TivaWare TM4C1294XL datasheet)

To use PWM signal we have to initialize the required pins using TivaWare functions

- Configure the generator used to down mode to align all the edges (PWMGenConfigure)
- Set the period to generator (time * clock it is 5KHz for this project) (PWMGenPeriodSet)
- Set the duty cycle (PWMPulseWidthSet)
- Enable generator and its timer (PWMGenEnable)
- Set the output bit to true (PWMOutputState)

There are three motors to control the robot: two for the car to move forward/backward and the other one for direction. There are two functional speeds used in the project velocity 1, 2. The speeds of motors were determined using trial and error method.

The microcontroller receives bytes of data from user which is compared to run the predefined functions. Each byte of data received is compared and the function is called. The lines of code were found in the this location `../cc3000_basic_wifi_application.c`

```

else if(i32ReturnValue==3)
{
    if(g_pui8CC3000_Rx_Buffer[0] == 'L' && g_pui8CC3000_Rx_Buffer[1] == 'F' &&
        g_pui8CC3000_Rx_Buffer[2] == '1')
    {
        UARTprintf(" LF1 checked\n");
        LForward1();
    }
}

```

Figure 16: Code which compares every byte of information sent to microcontroller to call the function *Lforward1()*

Each PWM Generator is configured to 5 KHz frequency (8000 cycles). Now, these speed functions were defined as forward1, forward2, backward1, backward2 etc. which were forwarded from GUI to Wi-Fi SimpleLink cc3000 and to microcontroller. The File in ../src/PWM_10-30.c has all the required functions written.

F. DUAL MOTOR CONTROLLER MC33926

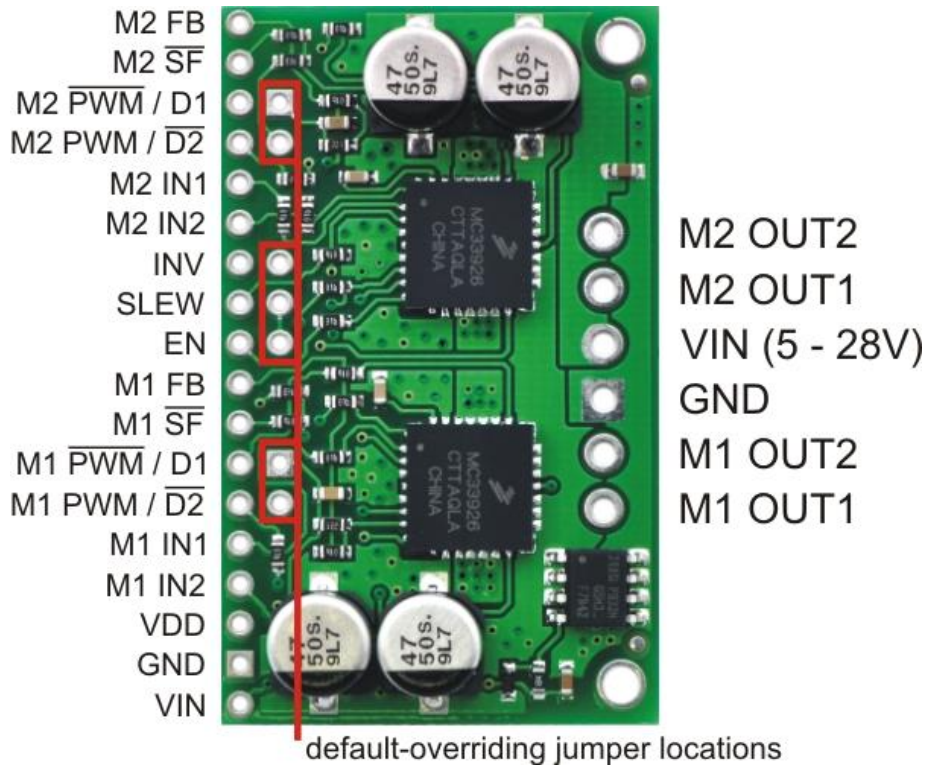


Figure17. Showing MC33926 Dual motor controller [8]

The dual MC33926 motor driver carrier is a breakout board featuring two Freescale MC33926 H-bridge ICs. It can control two dc motors at 5V-28V and it can tolerate peak currents up to 5A per channel. The polarity and voltage from this driver can be controlled using PWM signal. We have chosen 5 KHz frequency rate. The IN1 and IN2 are fed with PWM signals to control the OUT1 and OUT 2.

IN1(PWM-Pin)	IN2	Result
High	Low	Forward
High	High	STOP
Low	High	Backward
Low	Low	STOP

Table 1: Showing the logic followed by motor controller

9. Supply:

A 9V battery is used as supply and velocity 1 delivers 3V to motors and velocity 2 delivers 4V to motors. The motor which controls the heading is supplied with a constant voltage of 4V to make it move towards left or right. Note all these voltages are finalized by trial and error method to use least power possible.

10. Implementation

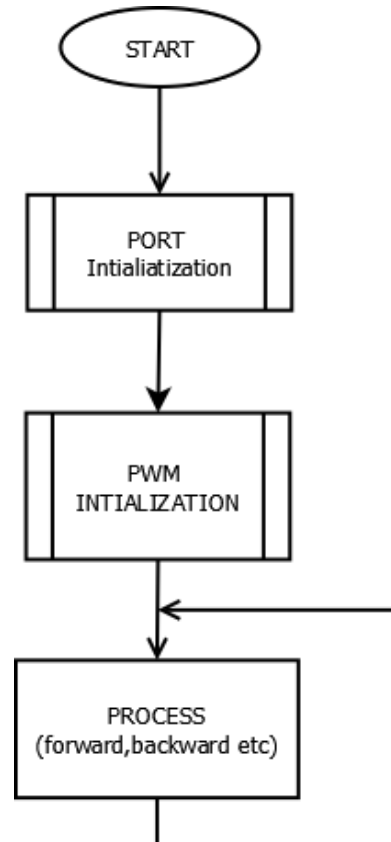


Figure18. Flow chart showing the response of PWM module

The GUI sends bits of information via Wi-Fi SimpleLink to microcontroller the resulting the functions execute predefined functions PWM_10-30.c.

III. Conclusion and Results

We successfully implemented the proposed modules and an extra feature of making it an autonomous car by using the IR sensors. Implemented PWM using the motor controller and Wi-Fi connectivity with the SimpleLink cc3000 BoosterPack. IR sensors send analog signal to the controller, using two ADC channels to the MCU which are processed at 10Hz. And a reset interrupt to reset the robot when the connection is lost between PC and the robot.

Battery BoosterPack gives enough power to operate the TI Launchpad and the sensors. Please find the final code in this link https://github.com/RoboEvangelist/TivaWare_C_Series-2.1.0.12573.



Figure 19: QR code of the result in github

IV. Future Research & Discussions:

The robot is quite stable at this moment, but a PCB will be made to make the circuit more stable. Also, instead of using two IR sensors the robot will use only one IR sensor mounted on a servo motor which can sweep and detect (e.g. radar) the obstacles. This will provide a much more accurate obstacle detection mechanism. Finally, the robot's software will be combined with ROS (Robot Operating System), providing a more robots and scalable software architecture.

V. Acknowledgments

Luis Bill, Ravi Teja, and Siddharth Jankar thank Dr. Xiaorong Zhang for teaching ENGR 844, for allowing us to work on this project, and for providing us with sensors and other parts for our project.

VI. References

Books

¹ *Volume 1, Introduction to ARM Cortex-M Microcontrollers* (fifth edition, 2nd printing- June 2014), 2014, ISBN: 978-1477508992

Reports

² Tiva TM4C1294NCPDT Microcontroller Data Sheet File

³ CLP Workbook File

⁴ TivaWare Peripheral Driver Library User Guide File

Computer Software

⁵ μ Vision V4.74.0.22

⁶ Kivy-1.8.0-py2.7

⁷ Tiva™ C Series MCUs PinMux Utility

Private Communications and Web Sites

⁸ <http://www.pololu.com/product/1213>

⁹ <http://www.parallax.com/product/28995>

¹⁰ <https://www.youtube.com/watch?v=fxP9hnZysgo>

¹¹ <https://itunes.apple.com/us/app/ti-wifi-smartconfig/id580969322?mt=8>

¹² http://www.sharpsma.com/webfm_send/1205

Result

¹³ https://github.com/RoboEvangelist/TivaWare_C_Series-2.1.0.12573