Lab 2: Report

Sukrit Ganesh

Self-Driving Car - Wireless Technologies:

A self-driving car will make use of a wide variety of wireless technologies, as it is a highly-complicated computer on wheels. It must connect to a variety of systems using a wide variety of methods to ensure resiliency, low latency, and high performance.

Every self-driving car will require bluetooth to connect to a gadget owned by the driver. Most modern cars already have something like this in the form of a bluetooth phone connection. As self-driving cars are effectively rolling computers, smartphones end up being the primary interface to their systems, requiring a lightweight but reliable communication protocol. Power is not too much of a concern, but bluetooth is an efficient, reliable protocol for short-range communication, and it will not cause much interference with other car bluetooth systems given its relatively short range. Using Wi-Fi or another heavyweight protocol would be unnecessary as only data will be streamed, not live video or audio. Wearable technology such as smartwatches and even smart clothing are rising in popularity, and it isn't unreasonable to believe these technologies will be integrated with the car (much like how home IoT works). For instance, a smartwatch or smart pacemaker measuring patient health could automatically connect to the car and alert the car if the driver falls asleep, falls unconscious, etc. A robust bluetooth connection will be essential for the car to interface with these various gadgets.

If self-driving cars become more prolific, and people become accustomed to using their car for auxiliary activities, a specific protocol for a "smart car" system could be developed. Just like Zigbee works for home automation, a protocol that enables relatively fast communication with car data and telemetrics, including low-resolution video streaming, could become very useful. Self-driving cars will enable people to spend time working, relaxing, etc. in their cars, and they may want to connect their own gadgets and add-ons to the car for convenience, safety, etc. A protocol designed for this purpose, optimized for a moving metal box and having a slightly higher bandwidth than bluetooth to enable more intense communication (such as streaming the car's camera feeds), would be useful.

Self-driving cars will also need to make use of GPS. This is obvious - the car needs to know where it is! GPS 3.0 will be accurate to 1-3 meters, allowing it to be precisely placed in not just specific roads but also in specific lanes, parking spots, and driveways. It will suffice for the vast majority of localization. Of course, GPS will cut out inside tunnels, and it will be weaker in cities, but local GPS "satellites" placed in these areas, along with repeaters, can ameliorate these issues.

Of course, self-driving cars will need to make use of broadband cellular communications like 5G or Starlink. Whether streaming telemetry to a remote server, receiving text messages and emails (cars are literally computers these days, after all), streaming a YouTube video (you can actually do this on a Tesla), or providing a WiFi hotspot for the passengers, self-driving cars will need to connect to the internet somehow. Although 5G is the default method to do this, Tesla's Starlink is a promising new technology that could enable relatively strong internet connection (download speeds of 50 Mbps) across the globe regardless of location. The internet connection can alert central command centers in the event of an accident and stream and upload valuable dashcam data along with other telemetry. As stated before, self-driving cars allow users to perform a variety of tasks besides driving, so the hotspot provided by the internet connection will be very useful.

Speaking of satellites, self-driving cars should ideally maintain a direct satellite connection as well, similar to a satellite phone. Vehicles that get into accidents, break down, or run out of charge in remote locations without cell service must be able to communicate with the world. It is a useful feature that can save many lives in the long-run. This protocol need not be high bandwidth or high latency; it should simply be capable of forming a reliable radio link with a satellite and sending basic messages. These cars should also be equipped with conventional cell phone receivers and transmitters to contact emergency services when cell service is available.

Self-driving cars should make use of a contemporary protocol to interface with smart city gadgets like smart traffic lights, road sensors, etc. This will probably use a variety of protocols of varying bandwidths in order to interact with the various pieces of technology in the city. For instance, self-driving cars should be able to directly read whether a light is red or green not using visual images but rather by using a radio receiver that picks up transmissions from the traffic light. Having the car itself interact with other cars is also a good idea - these short-range communications can avert crashes and improve traffic flow, although it should be made clear that these systems should be largely passive, and self-driving cars should not be part of a vulnerable centralized network that can be hacked. They should be able to navigate by themselves, with the "smart city" connections simply used to improve performance rather than being essential to it.

Of course, a self-driving car, like a normal car, requires a radio. Who doesn't want to listen to music? Modern streaming services use the internet, but nothing beats the reliability of a simple radio, especially when internet connection isn't available.

Finally, self-driving cars make use of a variety of sensors to navigate. These include, but aren't limited to, infrared sensors, ultrasonic sensors, cameras, RADAR, and LIDAR. A combination of these sensors enables localization and mapping - the car can generate a 3D model of the environment around itself, detect objects, and accordingly make decisions.

Code analysis:

Raspberry pi IP: 10.0.0.191

In order to solve this problem, I extended the given code in wifi_server.py, index.html, and index.js. These files controlled the picar server, the frontend of the electron application, and the backend of the electron application, respectively.

The first thing I wanted to do was send simple messages to and from the server. The starter code already sent text, so I decided to up the ante and send json dictionaries instead. I also decided to read the CPU temperature on the raspberry pi and send it back to the client. This turned out to be somewhat complex, as on one end, I was using JS, but on the other end, I was using python. I decided to use UTF-8 encoding and convert the json dictionary to a string using json.dumps before sending it. Thankfully, both python and JS have easy-to-use UTF-8 decoders.

Once I got this working, I decided to actually modify the spans relating to temperature, speed, etc. on my frontend. This was relatively simple, as I just had to use document.getElementById and change the innerHTML. I created dummy values for speed and current_direction and send those back to the frontend as well. I successfully updated my frontend, and everything worked as expected.

My next challenge was figuring out how to send control data to the picar. The existing client function currently sent data in a textbox, so I decided to take advantage of the existing infrastructure and replace the textbox data with a function argument. The argument to the function would be the e.keycode value of the key pressed. I chose to use wasd for controlling the car, in addition to 4 to speed it up, 6 to slow it down, and spacebar to bring it to a halt. There were a few challenges. Console.log isn't available for electron, so I had to modify the innerHTML of a span on the screen in order to print out messages. This is how I found out the keycode of the keys I added - 4, 6, and spacebar. Every time a key was pressed, it would directly involve the client function (this is separate from the client function being automatically applied every 100 milliseconds to retrieve pi data). On the server side, a handler function would process the received data and take the appropriate action. When the client function was invoked to get data, the value it would send was "garbage" (thus the handler on the server side wouldn't modify the speed or direction of the car). The handler itself could change the speed or direction of the car depending on the key pressed. Another challenge was the fact that global speed and direction variables couldn't be accessed from my handler function, as apparently only the main function could access these variables. I got around this issue by setting the two variables in the main function and passing them into the handler function for updating (and the handler function would return the updated values as well).

Overall my car works well. The ssh connection made debugging easy, and the controller is easy to use and versatile due to the speed changing abilities. I am also able to read the speed, direction, and temperature of the pi.

Demo Links:

Folder containing report and demo videos:
https://drive.google.com/drive/folders/1tU3fzeZOn-qMFZ0tpd7ubQXQ5hgvT-5-?usp=share_link

Code Walkthrough Video:
https://drive.google.com/file/d/1Kwk50knrBfBxjS99Q_dJB1YPss8xeCPp/view?usp=share_link

Live Demo Video:
https://drive.google.com/file/d/1peLZu-gid4DI5Y_qXYWotgMqYMpbIaaX/view?usp=share_link