ILR 05 – Progress Review 4

Dorothy Kirlew

Team Daedalus Members: Mohak Bhardwaj, Shivam Gautam, Pranav Maheshwari, and Richa Varma

November 24, 2015

1. Individual Progress

For the progress review on November 24, I helped assemble the mobile platform and worked on setting up the Odroid-XU4 SBCs. I also worked on bidirectional Bluetooth serial communication between an Android phone and a laptop, which was split into three sections.

a. Mobile Platform

When the mobile platform finally arrived, I worked with Mohak and Shivam to perform the basic assembly of the mobile platform and charging station.

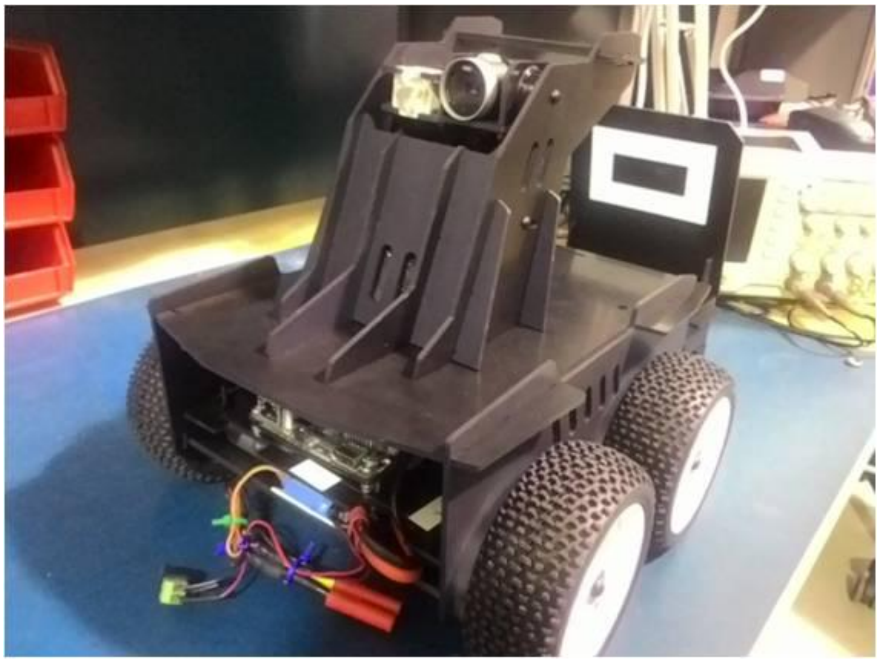


Figure 1: Assembled Oculus Prime, photo by Richa Varma

b. ROS on Odroid-XU4 Ubuntu ARM

Mohak installed Ubuntu ARM on the three Odroid-XU4s. I then installed ROS Indigo and the Oculus Prime ROS packages on the Odroids. This presented a few challenges, which I will discuss further on in this review.

c. Sending Bluetooth Serial Messages

For the last progress review, Mohak created a Python script that received serial messages over Bluetooth and displayed the received message on the terminal. Pranav and I modified this script to do two things. First, upon receiving a message, the script now accepts user input from the terminal. Second, the script sends this user input to the connected device, which, in this case, is an Android phone. This exchange can be seen in Figure 2.

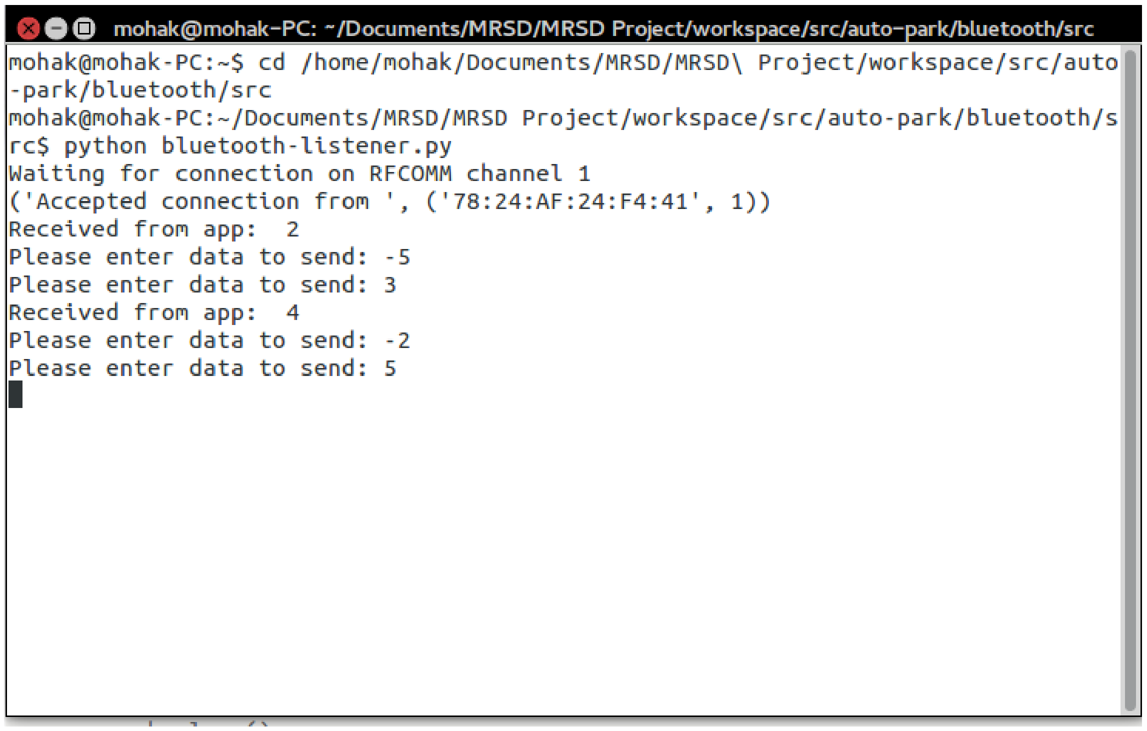

```
mohak@mohak-PC: ~/Documents/MRSD/MRSD Project/workspace/src/auto-park/bluetooth/src
mohak@mohak-PC:~$ cd /home/mohak/Documents/MRSD/MRSD\ Project/workspace/src/auto
-park/bluetooth/src
mohak@mohak-PC:~/Documents/MRSD/MRSD Project/workspace/src/auto-park/bluetooth/s
rc$ python bluetooth-listener.py
Waiting for connection on RFCOMM channel 1
('Accepted connection from ', ('78:24:AF:24:F4:41', 1))
Received from app:  2
Please enter data to send: -5
Please enter data to send: 3
Received from app:  4
Please enter data to send: -2
Please enter data to send: 5
```

Figure 2: Bidirectional Bluetooth Serial Communication

These messages should follow the protocol set forth by Pranav:

- 2 = Parking
- 3 = Parked
- 4 = Returning
- 5 = Returned
- Any negative integer = one fifth of the seconds it will take to reach the destination

Note: “1” is for “Free” and is the default state of the vehicle. There is no need to include it in a message.

d. Receiving and Parsing Bluetooth Serial Messages on App

Android provides BluetoothChatService.java, which I used in the last progress review to send serial messages over Bluetooth. BluetoothChatService.java is also set up to receive Bluetooth serial messages. I modified the method that received messages to only use messages that match the message format above. Additionally, I added logic to ensure that any messages sent out of turn will be ignored. For example, if “Return” is sent while the vehicle is parking, some error has occurred and no action should be taken on the part of the app.

e. Updating the Android App User Interface

The app follows the same base functionality displayed in the last progress review; when the app is first opened, only the Park button is enabled. When the Park button is pressed, it is disabled, the Return button is enabled, and the Status and ETP update (**Error! Reference source not found.**). When the

Return button is pressed, it is disabled, the Park button is enabled, and the Status and ETP update (**Error! Reference source not found.**). The ETA and ETP now say "calculating" until the app receives an estimate of the time needed.

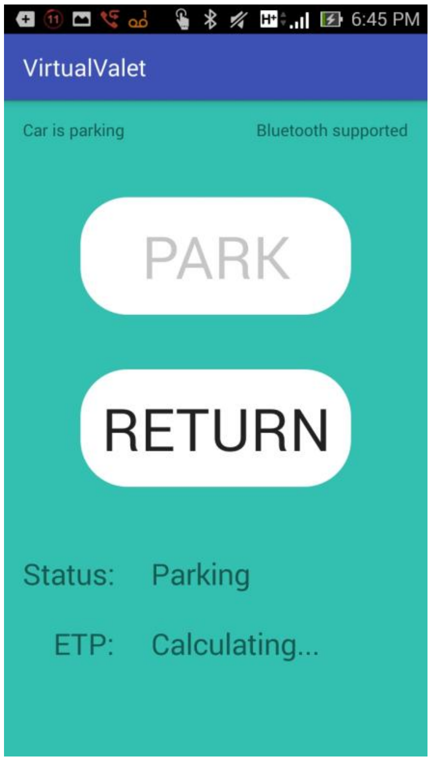


Figure 3: Park Button Pressed

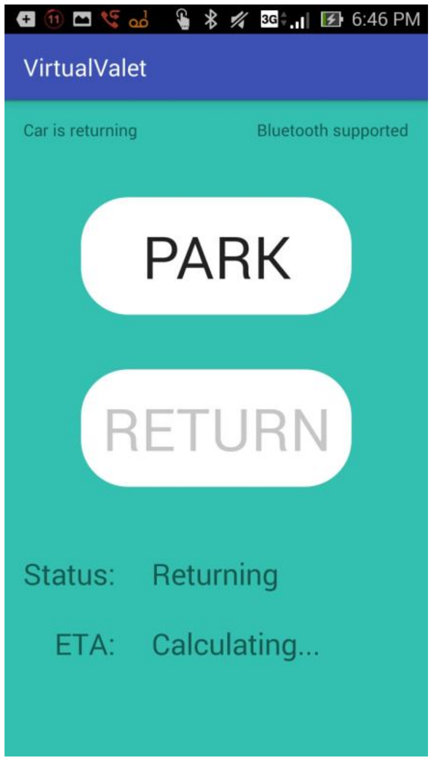
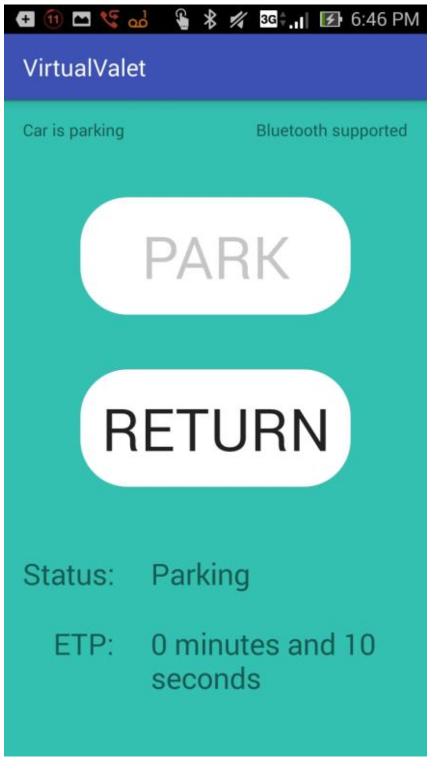


Figure 4: Return Button Pressed

Due to incoming serial messages, the app can now display a timer that counts down the time until the vehicle reaches its destination, as seen in Figure 5 and Figure 6. Additionally, the status will update when the car is parked and when it is returned, as seen in Figure 7 and Figure 8.

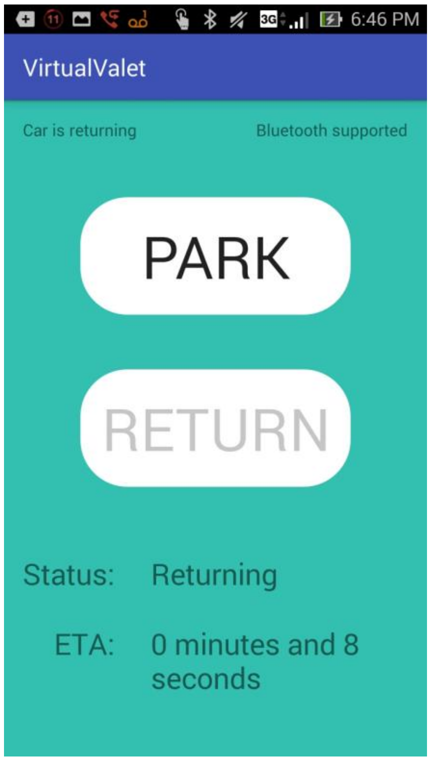


Car is parking Bluetooth supported

Status: Parking

ETP: 0 minutes and 10 seconds

Figure 5: Estimated Time to Park

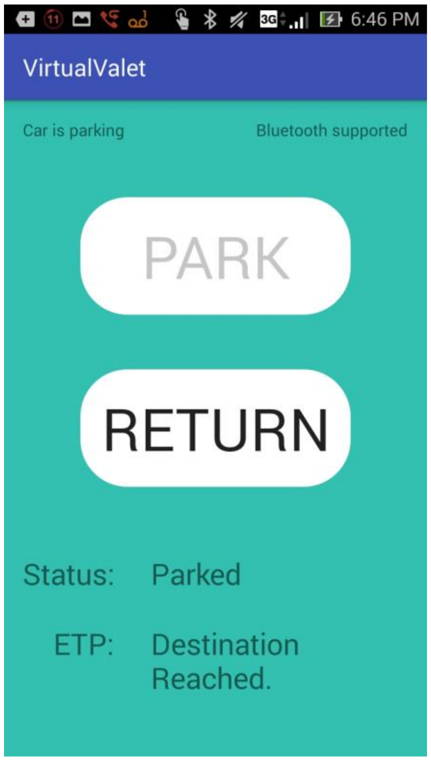


Car is returning Bluetooth supported

Status: Returning

ETA: 0 minutes and 8 seconds

Figure 6: Estimated Time to Arrive at Exit

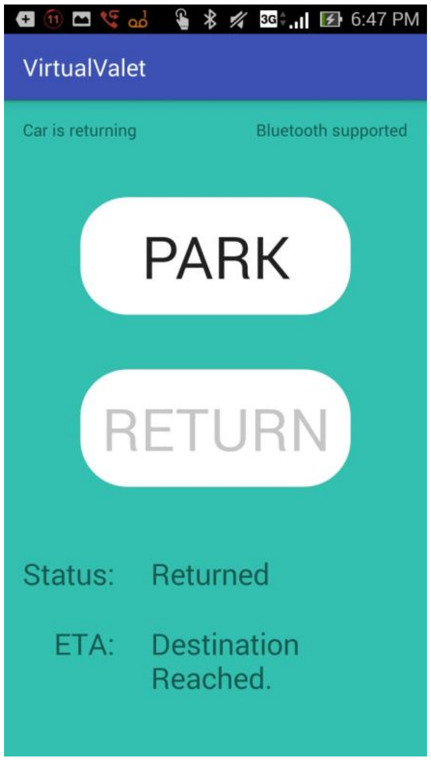


Car is parking Bluetooth supported

Status: Parked

ETP: Destination Reached.

Figure 7: Car is Parked



Car is returning Bluetooth supported

Status: Returned

ETA: Destination Reached.

Figure 8: Car is Returned

2. Challenges

a. Setting up the Odroid-XU4s

I installed ROS on three Odroid-XU4s. The installation on the first Odroid took several hours to complete, as the instructions were not clear on what to do in the case of any difficulties. The third Odroid was relatively quick, as I had discovered solutions to most of the problems when performing the installation on the first Odroid. The problem occurred on the second Odroid. To set up the sources list, ros-latest.list had to be modified. The permissions had to be changed on this file in order to do this. I accidently changed the permissions on the main folder, etc, instead of the specific file. Changing the permissions on the etc folder is one of the worst things to do on Ubuntu, because it makes it impossible to use “sudo”. The only way to correct this is to re-install Ubuntu, which takes several hours. However, this mistake is easy to avoid in the future, which is why the installation on the third Odroid was so quick.

Of course, all of these issues became moot when we found that the Odroids operate on a 32-bit system and the Oculus Prime requires a 64-bit system. However, I learned a lot about what to never, ever do on Ubuntu.

b. Android and Laptop Bluetooth Communication

In addition to not having an Android phone or a secondary laptop, Pranav and I were having difficulty getting Bluetooth serial communication to work between the Virtual Valet app installed on his phone and the Bluetooth listener running on his laptop. After many hours of struggling with this, changing the Bluetooth listener, reverting the app code to the most basic form, etc., we finally tried communicating between Mohak’s phone and laptop, which worked perfectly. We still have not discovered why it only works with these two devices, but it probably is due to the age of Parana’s devices. Regardless, testing the app was quite easy once communication was established. However, this means that Mohak is unable to use his laptop or phone while I am testing the app.

c. Parsing Bluetooth Serial Messages

Once we were able to establish bidirectional communication between the phone and the laptop, the next task was to parse the incoming messages. The BluetoothChatService.java was set up to send the incoming messages to a separate file, BluetoothChatFragment.java. This file has too much extra functionality to make it useful, so I decided to parse the messages within BluetoothChatService.java. Initially, it was just reading the length of the buffer, and not the buffer itself. I cast the buffer as a String, but it contained the sent message followed by many unrecognizable characters. I extracted either the first character of the string if it was numeric, or all the digits in the string if the first digit was “-“, indicating that the message was the time to reach the destination, formatted according to the serial protocol defined in section 1.c.

d. Updating the Android App User Interface

The logic behind updating the UI based on a received message is contained within the BluetoothChatService. For example, when the vehicle status is 2 (parking) and the app receives a message saying that the car is parked (3), the status on the UI needs to update to indicate this, as seen in Figure 7**: Car is Parked**. The problem with this is that BluetoothChatService cannot update the view hierarchy, because MainActivity.java is the owner of the view. The error of attempting to update the UI from the BluetoothChatService caused the app to crash. The workaround for this was to pass the MainActivity to the BluetoothChatService, and create a Handler to jump back in to the Main Activity and update the UI. The Handler is required because the BluetoothChatService is a static class and MainActivity is non-static, which means that BluetoothChatService cannot call any methods in MainActivity. Once I was able to update the UI from BluetoothChatService, I was able to proceed with testing.

3. Teamwork

We divided the work amongst our five team members as evenly as possible, with respect to our experience and skills. Pranav created the IR mount for the Oculus Prime and designed the Decision Unit central node to manage dataflow between the subsystems. Richa and Shivam worked on sending, receiving, and parsing information communicated between the XBees. Mohak and Shivam worked on obstacle detection using Kinect on a laptop. Mohak also worked on the MinnowBoard in regards to setting up the Kinect environment and running the Oculus Prime server. Shivam worked on the emergency node to interface with the Arduino Nano that is publishing data received from the IR sensors. I helped assemble the mobile platform, installed ROS Indigo on the Odroid-XU4 SBCs, and established bidirectional communication between the Android app and a laptop.

4. Plans

The team's goals for this progress review were to complete all of our subsystems. In the remaining time before FVE, we will be testing and validating our subsystems and integrating the subsystems. For the conceptual design review, our team created many tests to ensure successful subsystem integration. These will test Sensors, Communication, Android App Interface, Locomotion, Power, Localization, and Obstacle Detection. We will divide these tests amongst ourselves in order to make the most of our time and ensure that all subsystems are thoroughly tested and successfully integrate with each other.