EC535- VALAR MORGHULIS

Submitted by
Aravind Sridhar(U60780342)
Anmol Gupta(U58928729)

Introduction:

Our project for EC535 is a live MBTA tracker that tracks the current positions of MBTA trains. We track the positions of trains in all the Lines inside Boston and in all directions.

We chose this problem because we realized that every time we use the public transportation in Boston we have to look at the current positions of trains using apps on our phones which becomes inconvenient. So we decided to make a dedicated Embedded system to track the positions of trains at that current moment.
This way, if the system is mounted on the wall or in a similar position, all we have to do is to look at the map and we'll know where the trains are.

We were able to get the Gumstix to put a very interactive UI on the LCD using QT and we managed to establish Asynchronous communication between the Gumstix and a Raspberry Pi which accesses the MBTA servers to update the positions of trains periodically. These are then plotted on a map and then sent to the Gumstix. The Gumstix displays this map on the LCD.

The default ssh client on the Gumstix was bear which was very slow. It's encryption algorithms were also very old and so we could not transfer data back and forth through Ethernet between the Raspberry pi and the Gumstix. So we updated it to openSSH which was much more advanced.

Design Flow:

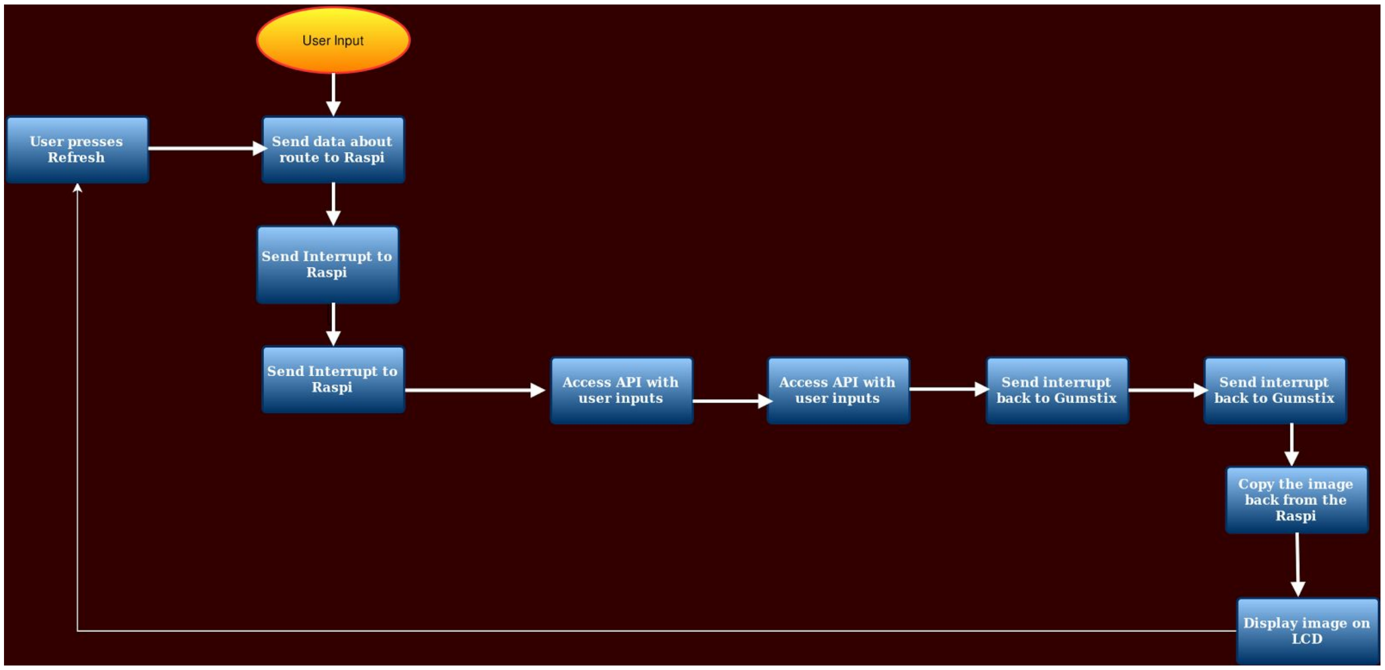
The design of this system is based on the communication between the Gumstix and the Raspberry Pi.

We used a Raspberry Pi to access data from the MBTA servers via a key we received from the MBTA IT service which we use to access data through an API that they provide.

This data is then stored locally and the positions of the trains are then plotted on a Google map using the static Google maps API for which we received a key as well.

When a user selects a particular Route and Direction, that data is sent to the Raspberry Pi via the Ethernet. Also an interrupt is generated to tell the Raspberry Pi it should then access the MBTA servers to get the current data. The Raspi then uses the data sent by the Gumstix to get appropriate data from the MBTA servers. The server returns data in several formats based on the request. We primarily requested xml format to be returned. An example response from the Server is included in the Report.

The Raspi then processes this information and then sends an interrupt to the Gumstix which then copies over the image of the map from the Raspi. This is then displayed on the LCD.
The following is the Design Flowchart for the module:


User Input
User presses Refresh
Send data about route to Raspi
Send Interrupt to Raspi
Send Interrupt to Raspi
Access API with user inputs
Access API with user inputs
Send interrupt back to Gumstix
Send interrupt back to Gumstix
Copy the image back from the Raspi
Display image on LCD

For the overall design, We split it up this way:

Internet access and Communication establishment: **Aravind Sridhar**
UI and Interrupt handler for Gumstix: **Anmol Gupta**

Project Details:

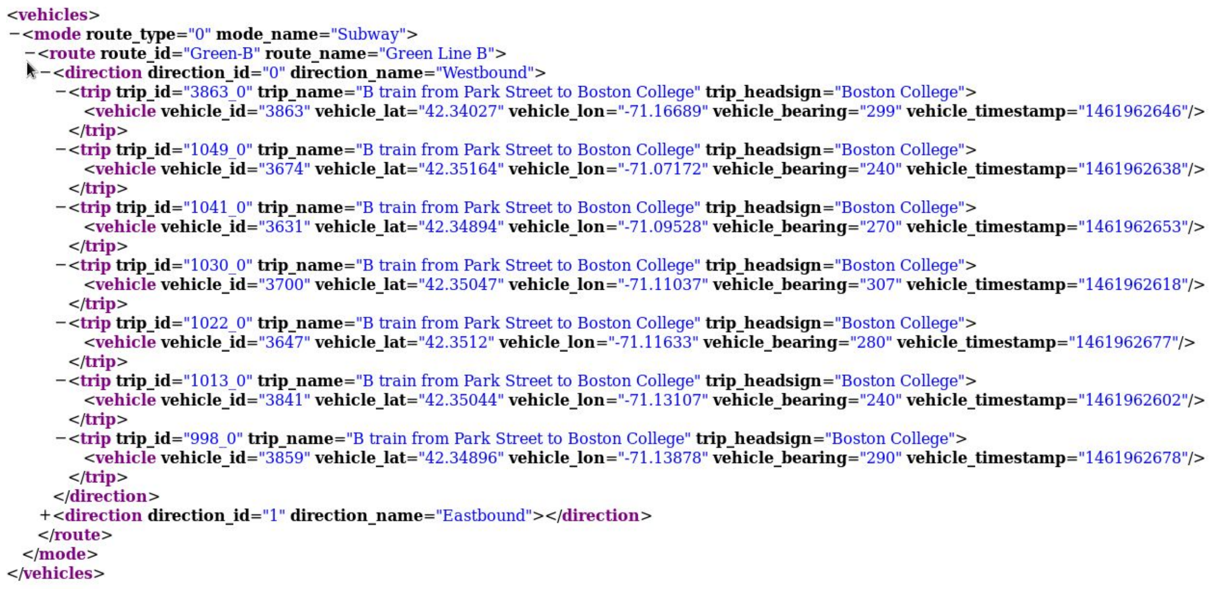
As mentioned before, the overall system depends upon the communication between the Raspi and the Gumstix. The Gumstix when, when it has received data about which route the user wants info about, it stores the data in a file called .input and copies it over to the Raspi. Then it sends an interrupt to the Raspi telling it that it has sent the file over. This is implemented via a Kernel Module. When a Write is performed on the Kernel module, it makes a GPIO pin high and then low.

The GPIO pin on the Gumstix is the connected to the Raspi which then detects the interrupt. So it reads the file that was transferred over. The file is in the following format:
<Route number>
<Direction>

Route number:
0 --> Red
1 --> Orange
2 --> Blue
3 --> Green-B
4 --> Green-C
5 --> Green-D
6 --> Green-E

Direction:
Eastbound
Westbound
Northbound
Southbound

There are several API queries that can be made. For our application, we use the Vehiclesbyroute which query which returns the locations of all the trains in a particular route. An example response is listed below:



```
<vehicles>
−<mode route_type="0" mode_name="Subway">
  −<route route_id="Green-B" route_name="Green Line B">
    −<direction direction_id="0" direction_name="Westbound">
      −<trip trip_id="3863_0" trip_name="B train from Park Street to Boston College" trip_headsign="Boston College">
          <vehicle vehicle_id="3863" vehicle_lat="42.34027" vehicle_lon="-71.16689" vehicle_bearing="299" vehicle_timestamp="1461962646"/>
        </trip>
      −<trip trip_id="1049_0" trip_name="B train from Park Street to Boston College" trip_headsign="Boston College">
          <vehicle vehicle_id="3674" vehicle_lat="42.35164" vehicle_lon="-71.07172" vehicle_bearing="240" vehicle_timestamp="1461962638"/>
        </trip>
      −<trip trip_id="1041_0" trip_name="B train from Park Street to Boston College" trip_headsign="Boston College">
          <vehicle vehicle_id="3631" vehicle_lat="42.34894" vehicle_lon="-71.09528" vehicle_bearing="270" vehicle_timestamp="1461962653"/>
        </trip>
      −<trip trip_id="1030_0" trip_name="B train from Park Street to Boston College" trip_headsign="Boston College">
          <vehicle vehicle_id="3700" vehicle_lat="42.35047" vehicle_lon="-71.11037" vehicle_bearing="307" vehicle_timestamp="1461962618"/>
        </trip>
      −<trip trip_id="1022_0" trip_name="B train from Park Street to Boston College" trip_headsign="Boston College">
          <vehicle vehicle_id="3647" vehicle_lat="42.3512" vehicle_lon="-71.11633" vehicle_bearing="280" vehicle_timestamp="1461962677"/>
        </trip>
      −<trip trip_id="1013_0" trip_name="B train from Park Street to Boston College" trip_headsign="Boston College">
          <vehicle vehicle_id="3841" vehicle_lat="42.35044" vehicle_lon="-71.13107" vehicle_bearing="240" vehicle_timestamp="1461962602"/>
        </trip>
      −<trip trip_id="998_0" trip_name="B train from Park Street to Boston College" trip_headsign="Boston College">
          <vehicle vehicle_id="3859" vehicle_lat="42.34896" vehicle_lon="-71.13878" vehicle_bearing="290" vehicle_timestamp="1461962678"/>
        </trip>
      </direction>
    +<direction direction_id="1" direction_name="Eastbound"></direction>
    </route>
  </mode>
</vehicles>
```

The Latitude and Longitude locations are extracted from this and are then stored in the two files .lat_file and .lon_file in the same folder.
It then write into the file .control.

The file map_access.py is waiting till the time the latitude and longitude data are available. To know if they are available, it keeps reading the file .control to see if the get_lat.py has written into the file. When it sees that get_lat.py has written into the file, it sees that the latitude and longitude data are available, and then constructs a Google maps API with the locations of the trains in the API. This then returns a static Map.
As soon as this map is downloaded, it sends an interrupt to the Gumstix.

On the Gumstix, when the interrupt is received, for the first time the device file is read, it returns a value of 1. All the other times, it returns a value of 0. The QT code, as soon as the user has selected all the options, Writes into the device file as mentioned above and then keeps reading the device file to see if a 1 is received. As soon as it sees that the device file returns a 1, it knows that the map is now available. It copies over the map from the Raspi using the scp command embedded within the c++ file.

We do all the scp from the Gumstix because we noticed that the scp is faster from the gumstix. So we have designed our program flow so that the gumstix does all the scp.

When the image is copied over, the Gumstix then displays the image on the LCD.

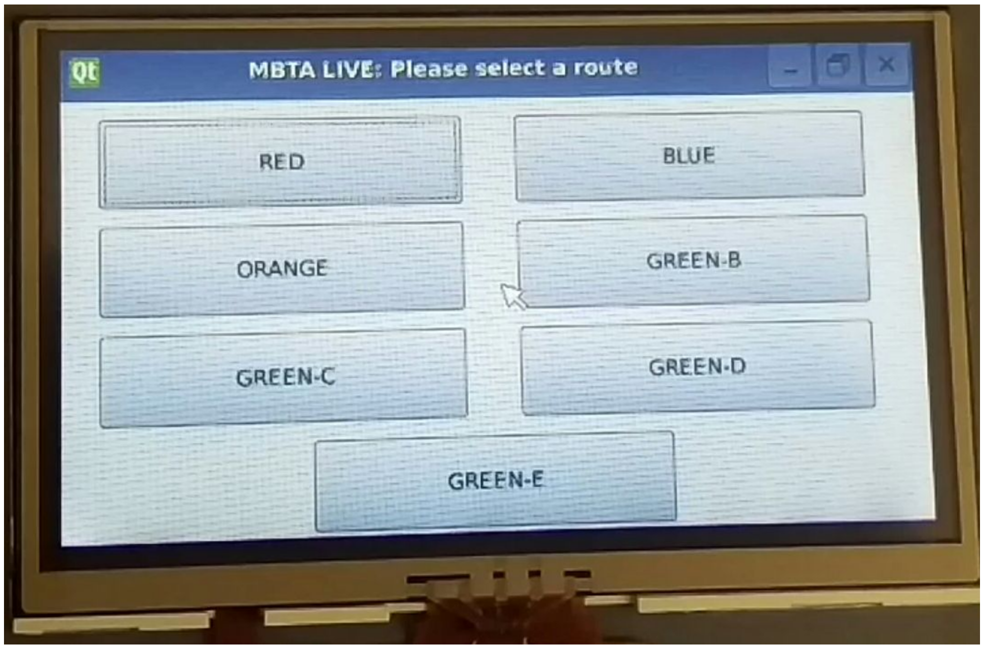
It also displays a REFRESH button on the top of the screen. When this button is pressed, the Gumstix sends an interrupt again to the Raspi to access fresh data from the servers.

QT:

The QT module which is the other major part of our system, is structured as follows:

Window 1:

This is the first window that comes up as soon as the code is run.

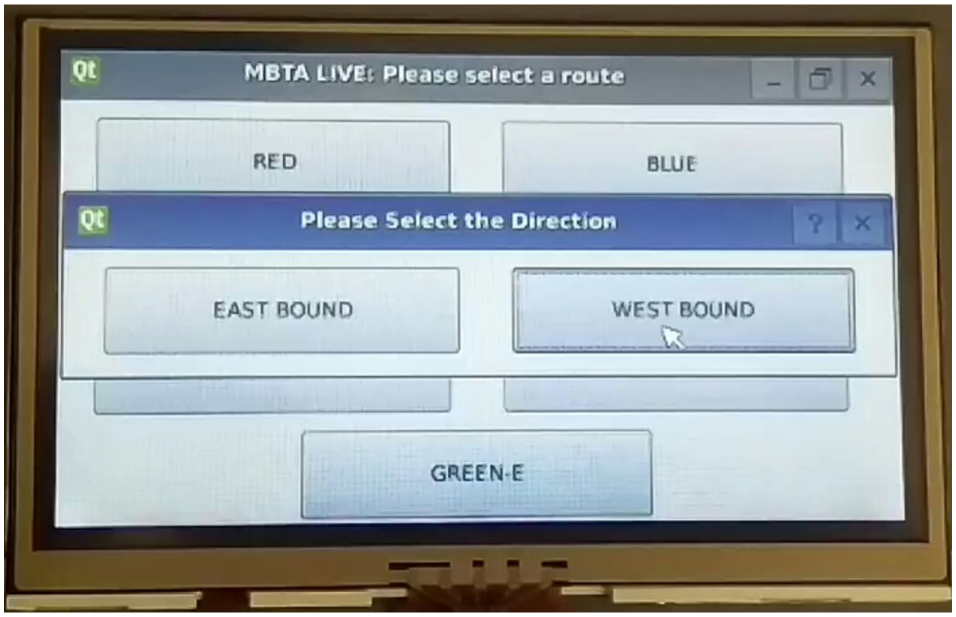


When the user selects the Route that He/She wants to travel on, a Dialog-box is displayed on this window. That effectively makes it the second window.

Window 2:

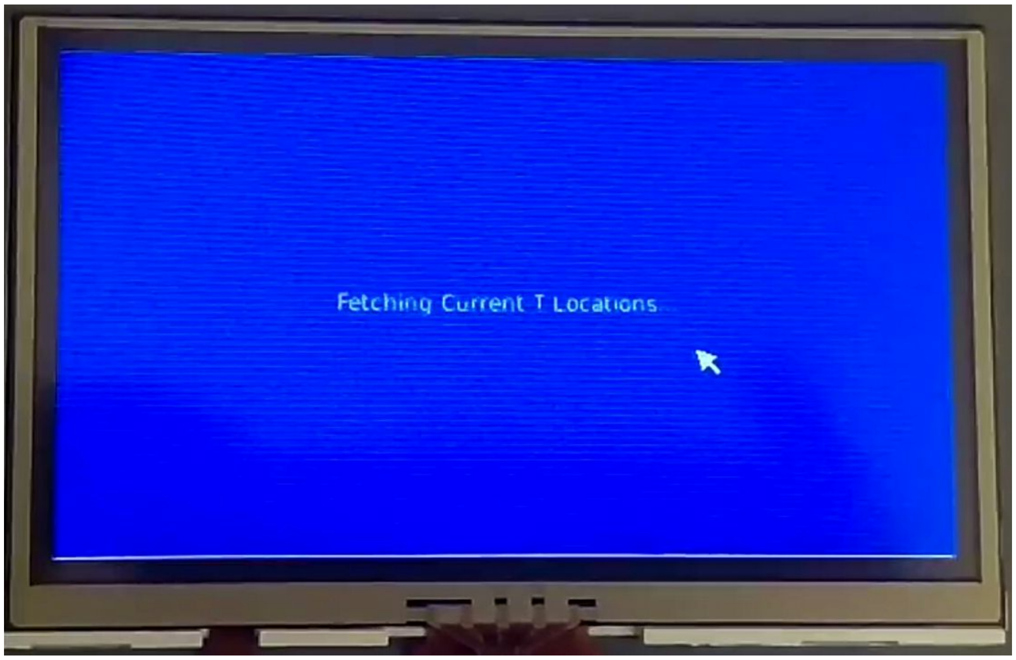
This window prompts the user to select the Direction in which they want to travel. Depending on the line, the options are either East Bound/West Bound or North Bound/South Bound.

Based on the direction selected, appropriate information is sent to the Raspberry pi via scp.
A screenshot of the screen is included below.



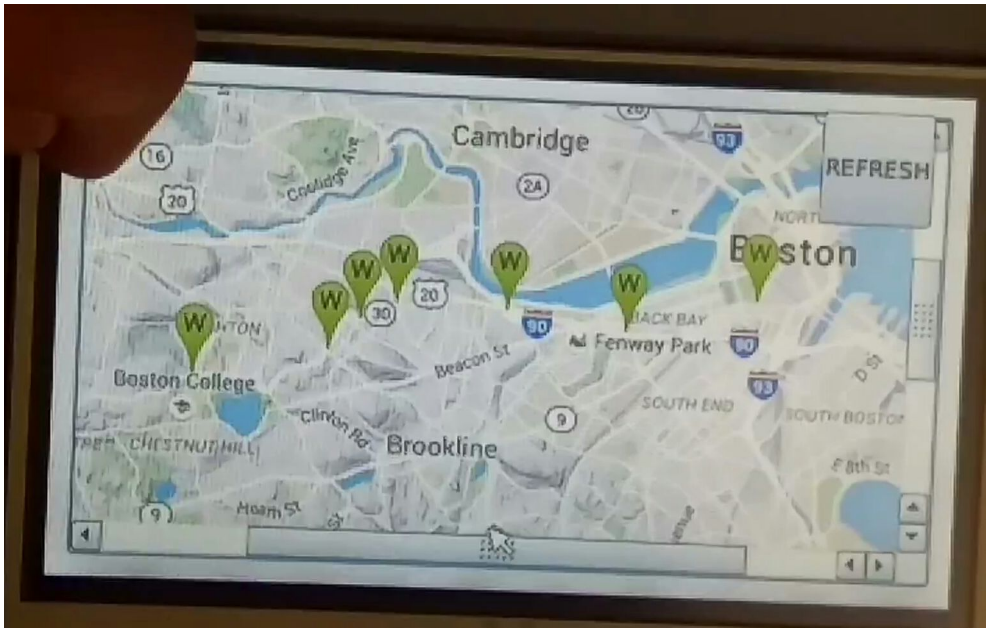
<u>Window3:</u>

When the data is sent to the Raspi, the LCD goes into a loading screen. It looks as follows:



<u>Window4:</u>

This is the window that displays the map. It has a Three widgets on it.
They are the scrollbar, Zoom button and Refresh button. Based on the value of the zoom, the scrollbar is also adjusted.
A screenshot of this window is attached below:



A zoomed in version:

The refresh button fetches new data from the MBTA servers. This is how the Live MBTA tracker was implemented.

For the circuitry, the pins 28 and 101 of the Gumstix were connected to pins 23 and 24 of the Raspberry pi for implementing the interrupts. This way the communication between the two are initialized and implemented.

Summary:

Hence as described above, the Live MBTA tracker was implemented. The difficulties faced were mainly with respect to setting up the networking on the Gumstix and synchronizing communication between the Gumstix and the Raspberry Pi.

References:

1. http://raspi.tv/2013/how-to-use-interrupts-with-python-on-the-raspberry-pi-and-rpi-gpio