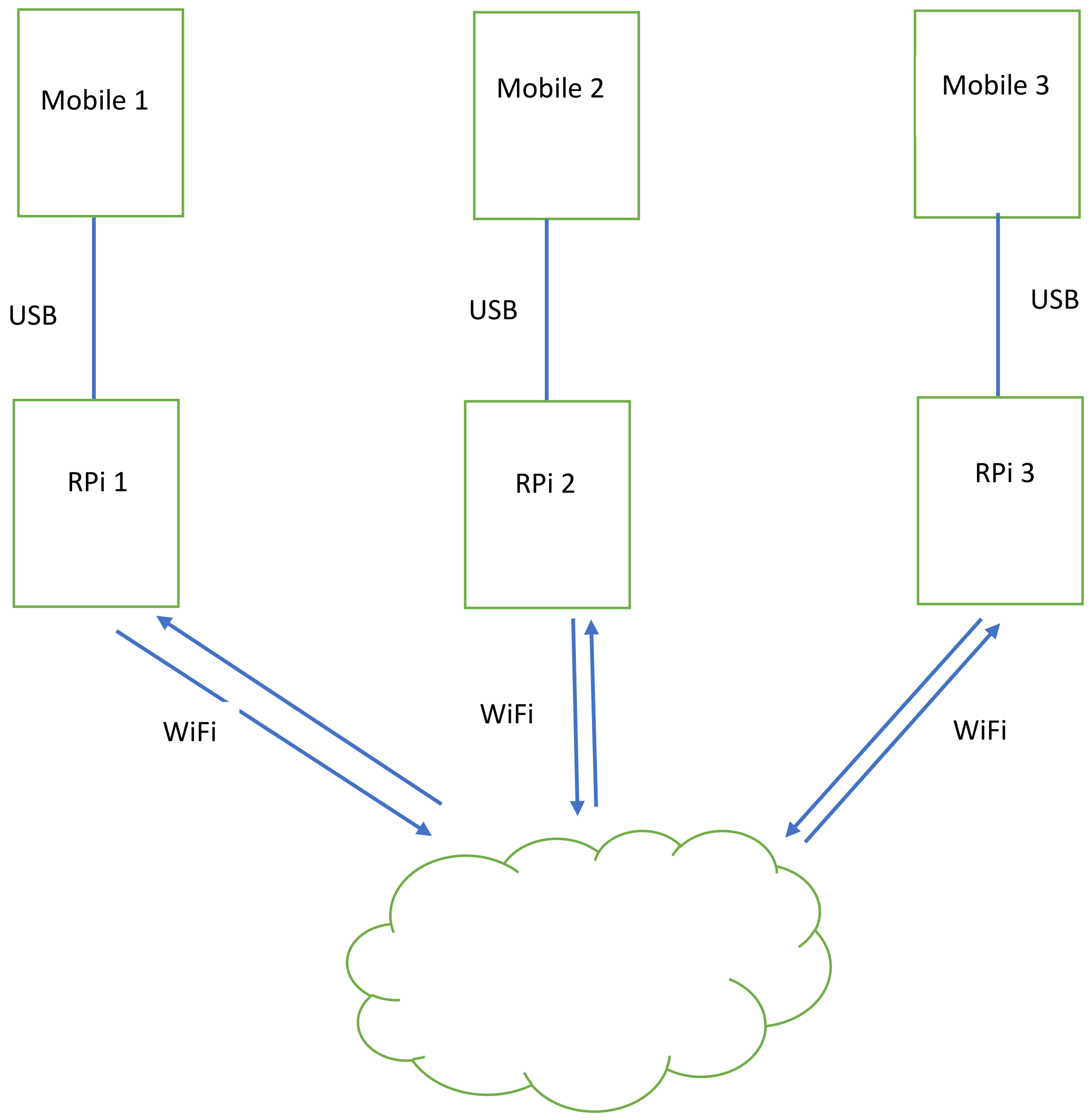
IoT – Fall 2017
Android Car Simulator

Mukul Anil Gosavi
Praveen Ganapathy Narasimhan
Supraja Mahadevan

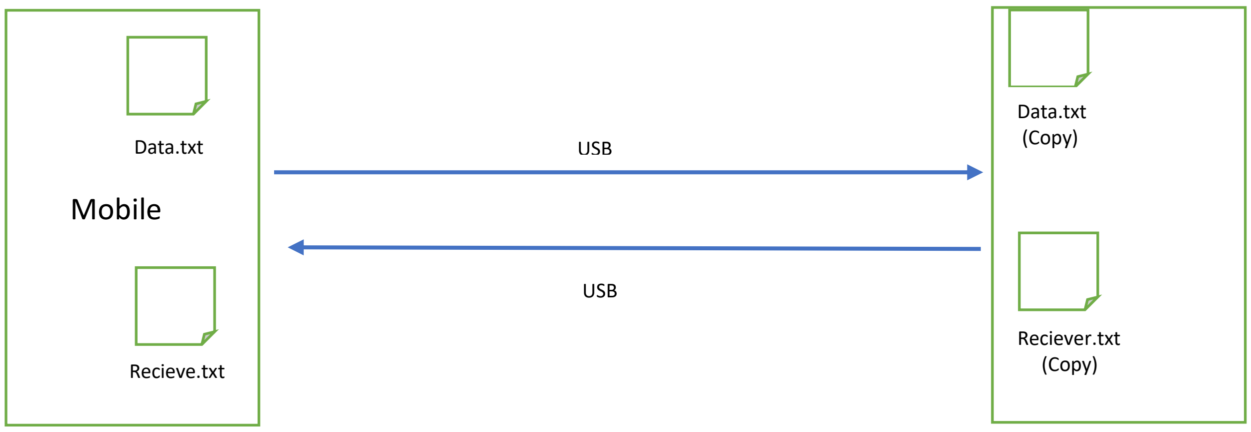
Aim:

The project aims at building a Car simulation Android application which displays the controlled motion of the car on a track. The computation engine is designed to be implemented on RPi, while the interfacing between the mobile device and the RPi and between various RPis are established using USB and Wifi.

Block Diagram:



USB Interfacing:

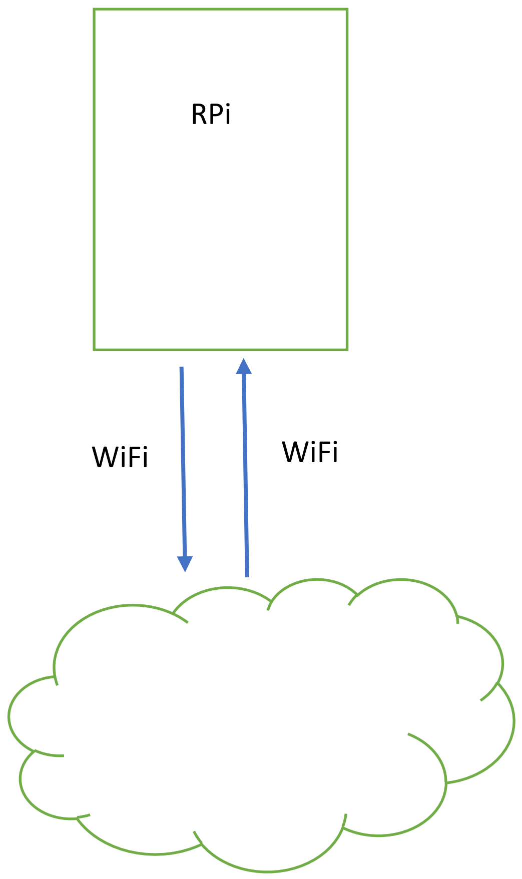


- When user controls the position of the car, the corresponding Acceleration, Steering and Ignition values are given to the file Data.txt and this will be written to the copy of the file created on RPi via USB communication and then the computation is performed in the RPi.
- Similarly, the details (X-Y Coordinates and Angle) of other cars will be communicated to the Rpi and these values will be updated onto the Receive.txt copy file in the Rpi along with the details of the current user's car and then data will be written onto the Receive.txt file in the mobile via USB.
- The mobile then reads the Receive.txt and accordingly updates the position of the remaining cars on the application.

Data.txt:
This text file holds three values of the current user's car information.
1) Ignition
2) Acceleration
3) Steering

Receive.txt
This text file holds nine parameters and the sequence will remain the same for all the three cases. Each of the data set will hold the X Coordinate, Y Coordinate, and Angle.
a) The data set of first user
b) The data set of second user
c) The data set of third user

<u>WiFi Interfacing</u>:



The RPi communicates with the other RPis via WiFi. There exists two modules SendDataPackets and ReceiveDataPackets running on each of the RPi simultanously.

a) The SendDataPackets broadcasts the X-Y coordinates and the angle of the current user.
b) The ReceiveDataPackets listens to all the data arriving at the corresponding ports from the rest of the RPis and transfers them to the mobile.

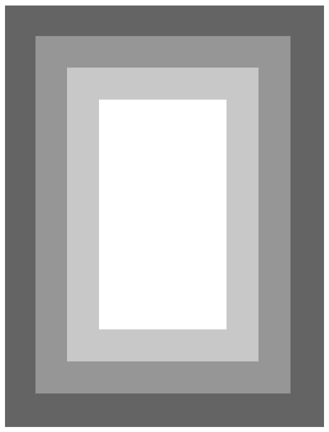
Application:

The project includes two Java classes each having its own functionality, four image files and one layout design file.

Image Files:
The drawable folder contains four image files, involving images of car and track. These image files are made use in the layout design to demonstrate the track view.

Car Images:

Track Image:



Java Classes:
A) <u>CarFunctions</u> :
The purpose of this class includes defining the boundaries of the car and that of the track and initialization.
CarTrackInit() Initializes all the parameters such as car dimensions (height, width etc.) and the track information from the images included in the layout.
FindCarVertices()The vertices of the corresponding car is estimated from the height, width and rotation information of the car.
FindCarVertices_12() Finds the vertices of the other two cars in the track
CheckBounds()This method checks the boundary conditions of the corresponding car, whether it stays within the permissible track.
CheckCarCrash() Checks if corresponding car crashes with the other cars.

B) MainActivity:

Methods

Buttons declaration and definition:

- Oncreate: This method contains the functionality of each of the button press in the layout. Two types of listeners are being used here
- OnTouchListener(): This listens to the button when pressed and withdraws when the button press is released. The buttons Accelerate, Brake, Steer Left and Right using this ontouchlistener logic.
- CompoundButton.OnCheckedChangeListener(): The Ignition ON button is used for starting the car engine and this is set as compound button which remains ON throughout the execution until turned OFF by the user.

There are significantly three threads running synchronously.

1) CarMoveForward():
   This is the thread which facilitates the translational motion of the car. When the carInMotionFlag is set and that the car is within the boundary, then the car is moved with a specific speed.
   When the accelerator is stressed, the speed increments accordingly and the car image in the track tends to move with the updated speed with the use of time delay concept. The brake operation functions the same except for the speed decreases.
   The speed of the car is set to be maintained within a bound.

2) CarTurn():
   This function works when the car is steered left or right by the click of the button and based on the carTurnFlag value, we can determine which turn the car makes, the left or the right.
   The carsteer angle increases until it reaches a 45 degree maximum on either left or right side and then reverts back to its original position, for that's how the steering process works in any car.
   This thread thus implements a proper right or left steer.

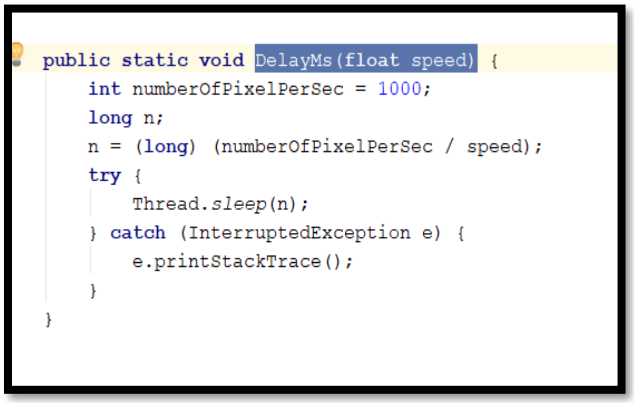
3) updateTextViews():
   This thread calculates and updates all the values and metrics to be displayed on the application.
   This application includes the display of the following metrics
   a) Position: This is the position of the car in the XY plane. The coordinates denote the position.
   b) Velocity: The current speed with which the car moves in the track. The initial speed of the car is set as 10 miles/hr and the speed changes based on the stress on accelerator or brake button in the application. The car pauses when the velocity drops down to 0 miles/hr.
      Velocity = The number of pixels moved / Time consumed to cover those pixels distance.

c) Acceleration: This is the parameter which changes only when the accelerator or brake buttons are pressed. The value goes positive when accelerated and negative when damped. The acceleration is assumed to increase by 0.1 on each touch.

d) Steer angle: This gives the angle in which the steering angle is steered when pressed steer left or right buttons. The steer angle can hold a range of -45 to 45 degrees.

e) Braking distance: This gives the distance in pixels from the point of application of brake and the point of actual car pause. This is computed by the product of Speed at which it traverses when braked and time taken to pause.

DelayMs(**float** speed): Tis function is used to convert the input speed to time delay by using the following code. More the speed lesser will be the delay.



```
public static void DelayMs(float speed) {
    int numberOfPixelPerSec = 1000;
    long n;
    n = (long) (numberOfPixelPerSec / speed);
    try {
        Thread.sleep(n);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

Computation Part:

This Module resides in the RPi and thus computes the required data.

a) Velocity: The current speed with which the car moves in the track. The initial speed of the car is set as 10 miles/hr and the speed changes based on the stress on accelerator or brake button in the application. The car pauses when the velocity drops down to 0 miles/hr.
Velocity = The number of pixels moved / Time consumed to cover those pixels distance
Velocity is computed using the formula
vt= (Speed of the car)/(Scheduled Speed) [Scheduled speed is set 100]

b) Position: This is the position of the car in the XY plane. The coordinates denote the position.
Coordinates of the car:
The following formulae can be used to estimate the car position from the previous position. Here, (X_0, Y_0) is the previous position and the angle theta is estimated from the steering values.
$X_1= X_0+(vt) \text{Sin}(theta)$
$Y_1=Y_0+(vt) \text{Cos}(theta)$

c) Steer angle: This gives the angle in which the steering angle is steered when pressed steer left or right buttons. The steer angle can hold a range of -45 to 45 degrees. This is estimated from the

d) Braking distance: This gives the distance in pixels from the point of application of brake and the point of actual car pause. This is computed by the product of Speed at which it traverses when braked and time taken to pause.

Contribution:

Application development : Mukul Anil Gosavi
USB interfacing : Praveen Ganapathy Narasimhan
Wifi Interfacing: Supraja Mahadevan