

# CS6650: Smart Sensing for Internet of Things

## IoT enabled Bicycle Speedometer Report

Gorre Venkata Satya Praveen CS18B017  
Viswanath Tadi CS18B047  
Sikhakollu Venkata Pavan Sumanth EE18B064  
Poosala Sreenivasulu Sainath EE18B026

November 23, 2021

# Contents

<b>1</b>	<b>Problem Statement</b>	<b>3</b>
<b>2</b>	<b>Overview</b>	<b>3</b>
<b>3</b>	<b>Implementation Details</b>	<b>4</b>
3.1	Working principle . . . . .	4
3.2	Hardware used . . . . .	4
3.3	Schematic . . . . .	4
3.4	Cycle setup . . . . .	5
3.5	Working . . . . .	5
3.5.1	Detection . . . . .	5
3.5.2	Pulse capturing using interrupts . . . . .	5
3.5.3	Estimation of RPM . . . . .	6
3.6	Problems and their fixes . . . . .	6
3.6.1	Debouncing . . . . .	6
3.6.2	Halting problem . . . . .	7
3.7	Bluetooth Connection . . . . .	7
3.7.1	Bluetooth Device Selection . . . . .	7
3.7.2	Bluetooth Data Transfer . . . . .	8
3.7.3	Pairing & Data Transfer Part . . . . .	8
3.8	Android Application . . . . .	9
3.8.1	Obtaining GPS data . . . . .	9
3.8.2	Displaying Color-Coded Path . . . . .	9
3.8.3	Saving & Loading Tours . . . . .	10
<b>4</b>	<b>Demo / Application Walkthrough</b>	<b>11</b>
<b>5</b>	<b>Contributions</b>	<b>12</b>
<b>6</b>	<b>References</b>	<b>12</b>

# 1 Problem Statement

The problem at hand is to implement a microcontroller based system to estimate the speed of a bicycle. The bicycle speed is communicated over bluetooth to an Android smartphone for real-time display, storage, and possibly further metrics and analysis. A magnetic reed switch that turns on or off based on the presence of a magnet may be used to measure the RPM of the cycle. The Switch is to be fixed above the rear wheel and a magnet affixed to the rear wheel rim. The wheel's rotation will take the magnet near and far from the reed switch (resulting in an OFF/ON pulse) which can be used to directly estimate it's RPM. The speed can be calibrated from the RPM information.

Additionally, the GPS data is collected from the Android smartphone and mapped with the RPM data received using bluetooth. The RPM data can be visualized in real-time as a speed based color-coded map. RPM and GPS data is stored by the application, and it can be viewed later on.

# 2 Overview

The Speedometer system uses an Arduino microcontroller and a Android smartphone. The main functionalities of the system are as follows:

- The magnet(s) attached to the cycle wheel periodically activates the reed switch. The Arduino keeps track of the reed switch activations and estimates the RPM of the cycle (and thus its speed) using the time interval between consecutive pulses from the reed switch.
- The program will estimate the linear speed of the cycle. This value is calibrated to get the actual speed of the cycle and this quantity will be transferred to the Android application via bluetooth.
- Arduino starts up the HC-05 module and performs the necessary startup checks. The Arduino then goes into a continuous loop of calculating the RPM and sending the data to the smartphone.
- On start up, the Android application and the Arduino microcontroller are paired via bluetooth. The calibrated speed is transferred by the Arduino to the Android application using this connection.
- The Android application collects the GPS data from the smartphone in parallel to receiving the Speed samples. Each Speed sample is paired with corresponding GPS coordinates. The current view of the map is updated to add the new Speed-GPS datapoint synchronously.
- On completion of a tour, all the data collected is saved. All saved tours can be visualized later in the application.

## 3 Implementation Details

### 3.1 Working principle

A magnet is attached to the rim of cycle. The RPM/Speed of a cycle can be determined from the angular velocity of the magnet. The magnet on the wheel can be detected by a reed switch. Microcontroller can keep track of the time of triggering. The rpm can be estimated using a microcontroller from the time intervals between the detections.

### 3.2 Hardware used

- Arduino Uno or Mega microcontroller
- HC-05 module: For extending arduino to have bluetooth connectivity
- Reed switch and magnets
- Smartphone with the custom app
- Additional things like power supply, cables etc

### 3.3 Schematic

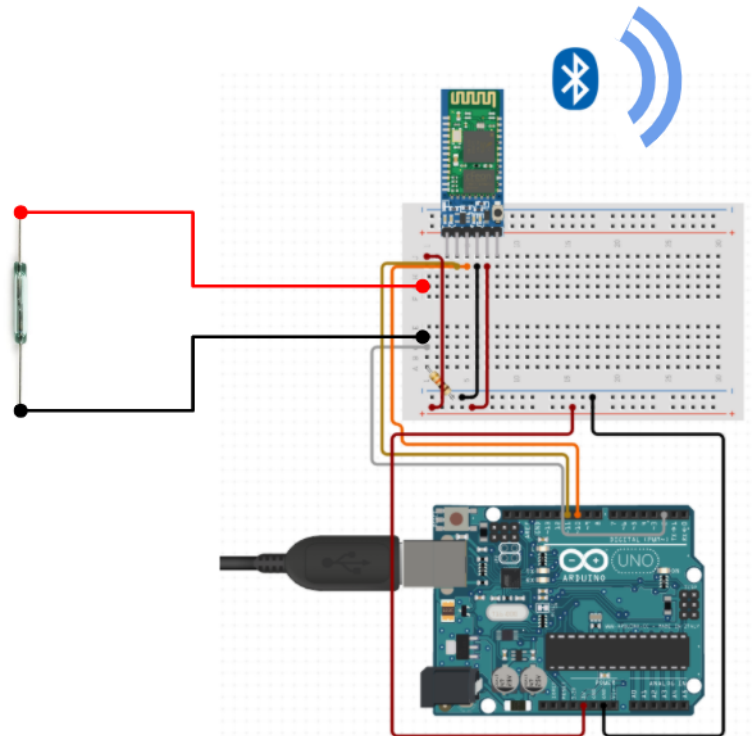


Figure 1: Schematic

### 3.4 Cycle setup

The resolution of the measurement can be increased by using more magnets. For simplicity, we consider the magnets are uniformly angularly positioned. If there are  $M$  magnets, then the angle between them are  $2\pi/M$ . More the number of magnets, more the samples per rotation. For better accuracy at lower speeds, more magnets are recommended. The arduino setup can be placed under the seat.

### 3.5 Working

#### 3.5.1 Detection

The reed switch should be placed close enough to the rim of the wheel but it should be attached to the stationary part of the cycle. The choice of number of magnets and placement of reed switch must be such that the reed switch will only get activated when a magnet is nearby and is deactivated rest of the time.

- If strong magnets are close enough, their magnetic field may combine and reed switch might get activated at unintended times.
- If reed switch is too far from the magnet, it may not pickup all the signals well

#### 3.5.2 Pulse capturing using interrupts

The microcontroller can keep track of the incoming pulses in two ways:

- Polling
- Using interrupts

In polling, the microcontroller goes into a loop waiting for the pulses. This can lead to missing some the samples when the microcontroller is performing other tasks. So the best choice would be to use interrupts for capturing. Interrupts are given highest priority so that any current task is paused and the control transfers to the interrupt service routine. Interrupts are associated with special class of functions called "Interrupt Service Routine (ISR)".

```
// Interrupt Service Routine
void interval_capture() {
    old_time = new_time;
    temp_time = millis();

    // sampling time more than 30ms
    // to eliminate false positives (Reducing Debouncing)
    // We constrain speed to an upper limit of 60kmph
    if ( temp_time - old_time < 30 ) {
        return;
    }
}
```

```

    new_time = temp_time;
    if ( index >= ARR_LEN ) {
        start_flag = 1;
        index = 0;
    }

    // updating the array (in seconds)
    time_interval[ index % ARR_LEN ] = ( new_time - old_time ) / 1000.0;
    index+=1;
}

```

### 3.5.3 Estimation of RPM

When the ISR is triggered, it stores the time interval between the current sample and previous sample into an array. If two pulses come within a very short span of time, this could be due to debouncing problem or false positive triggering of the reed switch. This problem is avoided by placing a minimum constraint on the time interval. This problem is explained later in this report.

This array contains the last N time intervals (equal to length of array). Moving average is calculated on the time intervals of the samples. This gives an average value which gives a better and stable estimate than just using the current time interval.

The microcontroller, when there are no interrupts, is continuously in a loop of calculating the RPM and sending the result to the mobile via HC-05. The array is updated only in the ISR. So when there are no interrupts, the array is not updated. If speed is too low or stopped completely, then we run into halting problem where the array is not updated and will not correspond to the current speed. This halting problem is explained later in this report.

Instead of sending the result to the phone in every iteration of the loop, we can control the rate at which data is sent by sending once every K iterations.

$$\begin{aligned}
 avg\_rpm &= m \times 60 / avg\_time; \\
 avg\_speed(kmph) &= avg\_rpm \times (2\pi / 60) \times radius\_tyre \times (18/5)
 \end{aligned}$$

where radius\_tyre is preconfigured in arduino

## 3.6 Problems and their fixes

### 3.6.1 Debouncing

If two pulses come within a very short span of time, this could be due to debouncing problem or false positive triggering of the reed switch. This problem is avoided by placing a minimum constraint on the time interval. This will lead to a maximum constraint on measurable speed.

We need to wisely select the maximum speed as selecting higher values can make the minimum time interval too low that samples due to debouncing will be also captured.

In our case, we observed that debouncing samples have a time interval of around 20 ms. So we have assumed 60 kmph as maximum speed whose time interval is well above 40ms. We implemented a simple filter which allows consecutive samples whose time interval is above 30ms.

```

old_time = new_time;
temp_time = millis();

// sampling time more than 30ms
// to eliminate false positives (Reducing Debouncing)
// We constrain speed to an upper limit of 60kmph
if ( temp_time - old_time < 30 ) {
    return;
}

```

### 3.6.2 Halting problem

The array is updated only in the ISR. So when there are no interrupts, the array is not updated. If speed is too low or stopped completely, then we run into halting problem where the array is not updated and will not correspond to the current speed.

We implemented our code such that if there are no interrupts for 5 seconds, we will make the speed to be 0 and time interval to a high values.

## 3.7 Bluetooth Connection

### 3.7.1 Bluetooth Device Selection

The Bluetooth Selector has been implemented to allow the user to select a paired bluetooth device and fetch its Mac address. There is a Floating Action Button (with a bluetooth symbol) in the bottom right of the application. Upon clicking this button, a popup window is displayed with a list of paired bluetooth devices. The user can select the Arduino device from this list (The Arduino must be manually paired with the Android phone in advance). Upon clicking on a device from the list, the defaultBluetoothAdapter is used to fetch the Mac address of the bluetooth device. The popup window is dismissed automatically after selection, and an attempt is made to establish a connection.

```

// Mac address fetched and Popup window dismissed.
MainActivity.getInstance().deviceAddress = bluetoothDevice.getAddress();
MainActivity.getInstance().tryToConnect();
popupWindow.dismiss();

// Try to establish a connection thread
BluetoothAdapter bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
createConnectThread = new CreateConnectThread(bluetoothAdapter, deviceAddress);
createConnectThread.start();

```

### 3.7.2 Bluetooth Data Transfer

A `ConnectedThread` class (extends `Thread`) is implemented to handle the bluetooth connection. The class instance runs in a background thread and continuously tries to read data from the Bluetooth Input Stream. When some data is available, it is parsed into a `Integer` (this value corresponds to speed in kmph) and the `getGPSLocation` function is called. This function calls the `addDataPoint` method of `MapDraw` with the current Latitude, Longitude and speed values. The new datapoint is rendered onto the canvas by invalidating the `MapDraw` canvas.

```
private void getGPSLocation(int speed){
    mapDrawView.addDataPoint(currentLatitude, currentLongitude, speed);
    mapDrawView.invalidate();
}
```

### 3.7.3 Pairing & Data Transfer Part

This system uses the HC-05 bluetooth module to connect to the smartphone wirelessly and to effectively relay the speed data to the application. Before the ride, arduino starts up the HC-05 module into pairing mode. The user/ application connects to the module and the arduino then goes into a continuous loop of catching the pulses, calculating the speed and sending the data to the app.



## 3.8 Android Application

### 3.8.1 Obtaining GPS data

Collection of GPS data from the Android phone is an asynchronous process that is triggered by the 'Start' and 'Save' buttons on the App. When the 'Start' button is pressed, the `requestLocationUpdates` method of the `fusedLocationClient` is called with the following Location request settings. The callback (the function that is called upon a GPS location update) sets the global variables `currentLatitude` and `currentLongitude` to the most recent coordinates. Hence, the collection of GPS data is independent of the collection speed samples from the Arduino.

```
locationRequest = LocationRequest.create();
locationRequest.setInterval(200);
locationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);

fusedLocationClient.requestLocationUpdates(locationRequest,
                                           locationCallback,
                                           Looper.getMainLooper());
```

When the 'Save' button is pressed, the `removeLocationUpdates` method of the `fusedLocationClient` is called. This deletes the existing `locationRequest` that is running asynchronously. This stops further updates and optimizes battery life. Hence, GPS location is used only when the tour is in progress.

```
fusedLocationClient.removeLocationUpdates(locationCallback);
```

### 3.8.2 Displaying Color-Coded Path

The latitude and longitude coordinates along with the speed samples are stored in an array in the `MapDraw` class. This data is visualized and displayed in the application in the form of a color-coded path. The color of a datapoint is determined based on the magnitude of speed. The speeds are mapped to colors in a Red-Yellow-Green color spectrum, with the higher speeds (>20 kmph) displayed in green, and the lower speeds (<10 kmph) displayed in red on the path.

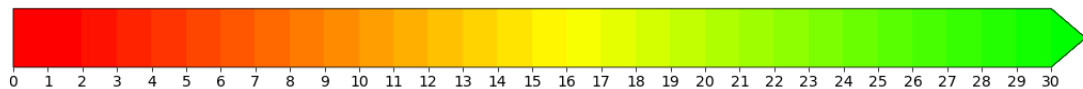


Figure 2: Speed(kmph) to Color mapping

The `MapDraw` class (extends `View`) handles all activities related to the Path display lifecycle. The Android `View` class has a `Canvas` object. Drawing onto the `Canvas` using the `onDraw` method creates the path on the `View`. The path is drawn as discrete line segments between consecutive datapoints. The path is displayed on the `Canvas` using `drawLine` method to render the path. The ends of the line segment are determined by the latitude and longitude scaled onto the `Canvas`, and the color determined by the value of speed. Upon addition of each new datapoint, `View.invalidate` method is used to clear and redraw the updated path.

The displayed path is dynamically adjusted to occupy about 90% of the canvas. When a new coordinate is added to the array, a check is in place to ensure that the datapoint does not go beyond 450px (90%) from the center of the canvas. In the case that the datapoint is further away than 450px, the value of SCALE is recalibrated and all the existing datapoints are rescaled. This causes the entire path to still be visible by reducing in size.

```
double stretch = max(topMostLatitude - x_center,
                     x_center - bottomMostLatitude,
                     rightMostLongitude - y_center,
                     y_center - leftMostLongitude);
int newScale = (int) (450.0 / stretch);
if(newScale != currentSCALE){
    currentSCALE = newScale;
    recalibratePreviousDataPoints();
}
```

### 3.8.3 Saving & Loading Tours

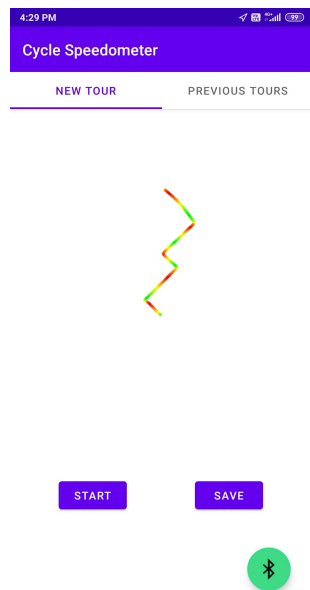
When the 'Save' button is clicked, the GPS updates are turned off, the MapDraw View is reset, and the running tour is ended. Each datapoint (Latitude, Longitude, Speed) is written to a new line in the file in a comma-separated format. The CSV file is saved in the private file directory of the application with the name derived from the current timestamp.

```
File file = new File(getActivity().getFilesDir(), "saved_routes");
writer.append("latitude,longitude,speed(kmph)\n");
for(int i = 0; i < min(x_coords.size(), speeds.size()); i++) {
    writer.append( Double.toString(y_coords.get(i)) + ","
                  + Double.toString(x_coords.get(i)) + ","
                  + Double.toString(speeds.get(i)) + "\n");
}
```

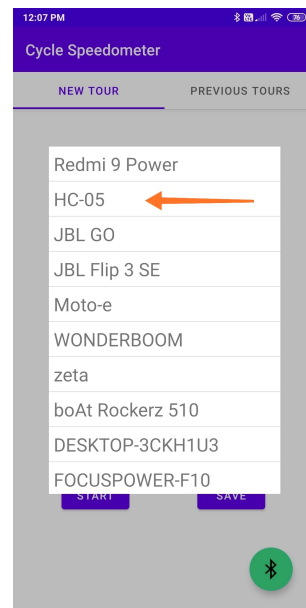
A list of saved tours can be viewed in the 'Previous Tours' tab of the application. Upon clicking any of the tours, a popup window containing the saved route appears. The route is visualized as a Color-coded path using the MapDraw class. The saved CSV file can be shared to external apps using the 'Share' button in the popup window.

## 4 Demo / Application Walkthrough

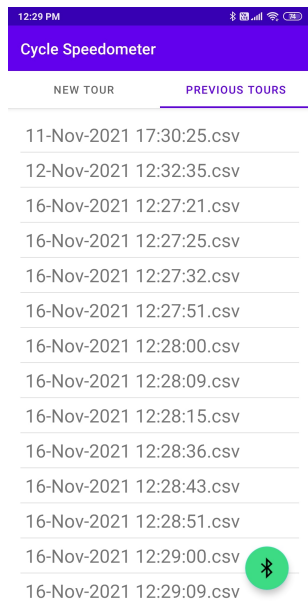
- Install and open the Cycle Speedometer app. Grant GPS permission to the app.
- Turn on bluetooth and make sure that the phone is already paired with the Arduino.
- Click the Bluetooth button on the bottom right and select the Arduino device from the paired device list.
- If the bluetooth connection failed to establish, make sure that the Arduino is turned on and is within a few feet.
- Click the 'Start' button to begin the tour. Now your path will be visible in real time on the map as the tour progresses.
- Click the 'Save' button to end the tour and save the path to storage.
- Saved tours can be accessed in the 'Previous Tours' tab. A list of previous tours will be visible.
- A saved tour can be visualized by clicking on it. The CSV file can be shared to external applications by clicking the 'Share' button.



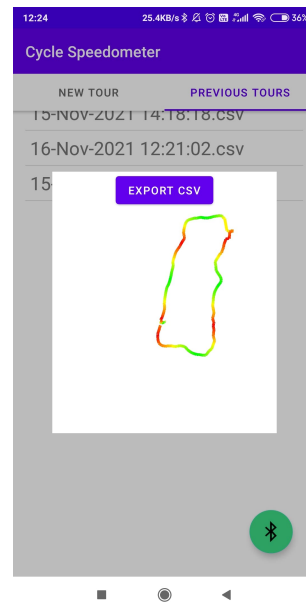
(a) Ongoing Tour



(b) Bluetooth Device selection



(c) Saved Tour list




(d) Saved Tour view

## 5 Contributions

	CS18B017	CS18B047	EE18B026	EE18B064
<b>Android Code</b>	50%	50%	0%	0%
<b>Arduino Code</b>	0%	0%	50%	50%
<b>Design Choices</b>	25%	25%	25%	25%
<b>Code Discussion</b>	25%	25%	25%	25%
<b>Report</b>	25%	25%	25%	25%
<b>PPT</b>	25%	25%	25%	25%
<b>Presentation</b>	25%	25%	25%	25%

## 6 References

- [Android Developer documentation](#) 
- [Establishing bluetooth connection from Android to Arduino](#) 