ECSE 682 Assignment 3 Report

Yicheng Song 260763294

Xiangyun Wang 260771591

2022/11/22

Introduction

In this assignment, we developed a pedometer using thunderboard as a Bluetooth GATT server and smart phone as Bluetooth GATT client. The thunderboard will collect and process acceleration data to count steps. The application on smart phone will connect to thunderboard directly and receive the step number. Similar to assignment 2, in the application the user could set goals for step number, calories and distance. As shown in figure 1, the goal and current exercising data are recorded and showed in two columns. To set the goal, the user can press the “SET GOAL” button to access the view shown in figure 2. After typing in the goals, the user can press “SET” button to return, or press “RESET” button to clear the fill-in content. The step counting will automatically start. Once each goal is achieved, a notification will pop out. By pressing the “RESET” button, the user could clear the current exercise data.

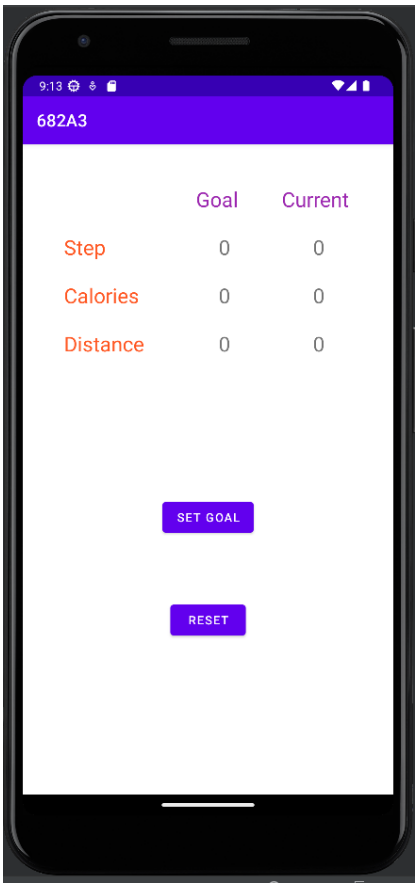

Figure 1. Main View

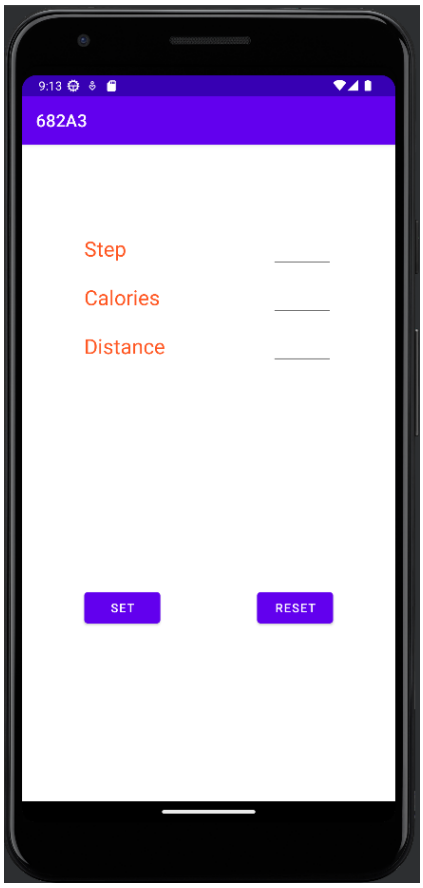

Figure 2. Set goal view

Implementation

This project consists of 4 main parts: thunderboard step-counting algorithm, Main activity, Set goal activity and BLE service.

Thunderboard step-counting algorithm

Assuming that the maximum walking/running speed for human beings is 4 step per second, we set the working frequency of accelerometer to be 10 milliseconds and update the step number every 1 second.

To identify a step, we only need to count the peak number within the 100 data, assuming that the acceleration on a single axis during walking is asymptotical to a trigonometric function. Since the difference between two data point would increase when approaching the peak, a peak could be recognized when the difference between two consecutive data points is larger than the threshold. The detailed implementation is shown below in figure 3.

```
static int16_t find_step(void){
    uint16_t peak_num = 0;
    int16_t counter_f = 1;
    int16_t previous = step_data[0];
    int16_t diff = 0;
    int16_t threshold = 300;
    int16_t min = step_data[0];
    int16_t max = step_data[0];

    while(counter_f < DATA_LEN){
        if(step_data[counter_f]>max){
            max = step_data[counter_f];
        }
        if(step_data[counter_f]<min){
            min = step_data[counter_f];
        }
        counter_f+=1;
    }
    if((max-min)>threshold){
        threshold = (max-min)*0.6;
    }

    counter_f = 1;

    while(counter_f < DATA_LEN){
          diff = sqrt((step_data[counter_f] - previous)*  (step_data[counter_f] - previous));
          if(diff > threshold){
              peak_num+=1;
          }
          previous = step_data[counter_f];
        counter_f += 1;
    }

    counter_f = 0;

    return (int16_t) (peak_num);
}
```

Figure 3. Implementation of step recognition.

Using the Bluetooth GATT Configurator provided by Simplicity Studio, a custom service “Pedometer Data” is added, which includes the GATT characteristics such as

“step_count” and ‘reset”, as shown in figure 4. The “step_count” characteristic is given the permission of read so that the GATT client could read the value within when connected to the GATT server. The “reset” characteristic is given the permission of write, which allows the GATT client to send data back to the thunderboard to reset the pedometer.

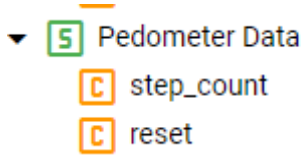

Figure 4. Custom GATT service “Pedometer Data”

Main activity

The Main Activity is mainly responsible for creating and binding to the BLE service. Function Set is linked to the “SET” button, which would bring up the set goal activity, as shown in figure 5. The goal will be transmitted back to main activity via an ActivityContract shown in figure 6.

```
public void Set(View v) {
    Intent intent = new Intent( packageContext: this, MainActivity2.class);
    ResultLauncher.launch(intent);
}
```

Figure 5. Launch Set Goal View

```
ActivityResultLauncher<Intent> ResultLauncher = registerForActivityResult(
        new ActivityResultContracts.StartActivityForResult(),
        new ActivityResultCallback<ActivityResult>() {
            @Override
            public void onActivityResult(ActivityResult result) {
                if (result.getResultCode() == Activity.RESULT_OK) {
                    Log.i( tag: "test", msg: "intent received");
                    Intent data = result.getData();
                    String message1 = data.getStringExtra( name: "StepGoal");
                    String message2 = data.getStringExtra( name: "CaloriesGoal");
                    String message3 = data.getStringExtra( name: "DistanceGoal");
                    TextView textView1 = findViewById(R.id.ShowStepGoal);
                    TextView textView2 = findViewById(R.id.ShowCGoal);
                    TextView textView3 = findViewById(R.id.ShowDGoal);
                    textView1.setText(message1);
                    textView2.setText(message2);
                    textView3.setText(message3);
                }
            }
        }
);
```

Figure 6. ActivityContract for fetching data from the Set Goal Activity

As shown in figure 7, the function updateViews will continuously update the current step number if the thunderboard is updating and transmitting the counted step number.

```
private void updateViews(Intent intent){
    countedStep = intent.getIntExtra( name: "stepcount", defaultValue: 0);

    Log.d( tag: "main", String.format("step in main: %d", countedStep));

    TextView CurrentSteps = findViewById(R.id.CurrentSteps);
    TextView CurrentC = findViewById(R.id.CurrentC);
    TextView CurrentD = findViewById(R.id.CurrentD);
    CurrentSteps.setText(String.valueOf(countedStep-oldStep));
    CurrentC.setText(String.valueOf(Integer.valueOf(countedStep-oldStep)*0.04));
    CurrentD.setText(String.format("%.2f", Integer.valueOf(countedStep-oldStep)*0.7));

    TextView textView1 = findViewById(R.id.ShowStepGoal);
    TextView textView2 = findViewById(R.id.ShowCGoal);
    TextView textView3 = findViewById(R.id.ShowDGoal);
    try{
        sgoal = Integer.valueOf(textView1.getText().toString());
    }catch(NumberFormatException e){
        sgoal = 0;
    }
    try{
        cgoal = Integer.valueOf(textView2.getText().toString());
    }catch(NumberFormatException e){
        cgoal = 0;
    }
    try{
        dgoal = Integer.valueOf(textView3.getText().toString());
    }catch(NumberFormatException e){
        dgoal = 0;
    }

    if(sgoal!=0 && Integer.valueOf(countedStep-oldStep) >= sgoal && !StepAchieved){
        Toast.makeText(getApplicationContext(), text: "Step Goal Achieved! Well done.", Toast.LENGTH_LONG).show();
        StepAchieved = true;
    }

    if(cgoal!=0 && Integer.valueOf(countedStep-oldStep)*0.04 >= cgoal && !CAchieved){
        Toast.makeText(getApplicationContext(), text: "Calories Goal Achieved! Well done.", Toast.LENGTH_LONG).show();
        CAchieved = true;
    }

    if(dgoal!=0 && Integer.valueOf(countedStep-oldStep)*0.7 >= dgoal && !DAchieved){
        Toast.makeText(getApplicationContext(), text: "Distance Goal Achieved! Well done.", Toast.LENGTH_LONG).show();
        DAchieved = true;
    }

}
```

Figure 7. Function updateViews

Another important function is Reset, which is responsible for setting the current step number to zero. The implementation of reset is realized by recording the instant step number as old step number when user press the reset button. The new step number shows on the screen in the main activity would automatically minus the old step number.

Set goal activity

In this activity, two methods are created corresponding to the two buttons “SET” and “RESET” shown in figure 8. The number typed in by users will be recorded and transmit back to the main activity through intent.

```
public void setgoal(View v){
    Intent back = new Intent( packageContext: this, MainActivity.class);
    EditText edit1 = findViewById(R.id.StepGoalTypeIn);
    EditText edit2 = findViewById(R.id.CaloriesGoalTypeIn);
    EditText edit3 = findViewById(R.id.DistanceGoalTypeIn);
    String sendback1 = edit1.getText().toString();
    String sendback2 = edit2.getText().toString();
    String sendback3 = edit3.getText().toString();
    back.putExtra( name: "StepGoal",sendback1);
    back.putExtra( name: "CaloriesGoal",sendback2);
    back.putExtra( name: "DistanceGoal",sendback3);
    setResult(Activity.RESULT_OK,back);
    finish();
}

public void resetgoal(View v){
    EditText edit1 = findViewById(R.id.StepGoalTypeIn);
    EditText edit2 = findViewById(R.id.CaloriesGoalTypeIn);
    EditText edit3 = findViewById(R.id.DistanceGoalTypeIn);
    edit1.setText(null);
    edit2.setText(null);
    edit3.setText(null);
}
```

Figure 8. SetGoal and ResetGoal button functions

BLE service

When starting the application, BLE service would automatically be created. Figure 9 shows the detailed implementation of initialization of Bluetooth adapter and connection to the GATT server of a BLE device via MAC address. The operation of connecting to GATT server will call the BluetoothGATTCallback,

As shown in figure 10, the BluetoothGATTCallback contains 3 functions: onConnectionStateChange, onServiceDiscovered and onCharacteristicRead. These functions would be invoked asynchronously when relative operation is finished. For example, when the smartphone successfully connected to the GATT server, onConnectionStateChanged will be invoked, and the application would try to discover all the services provided by the GATT server. After all services are discovered, onServicesDiscovered will be invoked, and the application would try to find the “Pedometer Data” service as well as the “step_count” characteristic using the 32-bit UUID. Every time readCharacteristic is called, onCharacteristicRead would be called and try to broadcast the counted step number to the main activity through the broadcast function.

```
public void onCreate(){
    super.onCreate();
    intent = new Intent(ACTION_BROADCAST);
    initialize();
    connect(deviceAddress);

    serviceStopped = false;
    Log.d( tag: "debug", msg: "on");

    handler.removeCallbacks(updateBroadcastData);
    handler.post(updateBroadcastData);

}

public void initialize() { bluetoothAdapter = BluetoothAdapter.getDefaultAdapter(); }

public void connect(final String address) {
    final BluetoothDevice device = bluetoothAdapter.getRemoteDevice(address);
    if (device == null) {
        Log.d( tag: "debug",  msg: "Device not found.  Unable to connect.");
    }
    if (ActivityCompat.checkSelfPermission( context: this, Manifest.permission.BLUETOOTH_CONNECT) != PackageManager.PERMISSION_GRANTED) {

    }
    bluetoothGatt = device.connectGatt( context: this,  autoConnect: true, bluetoothGattCallback);
    Log.d( tag: "debug",  msg: "Trying to create a new connection.");
}
```

Figure 9. Initialization of Bluetooth adapter and connection to Bluetooth GATT server

```
private BluetoothGattCallback bluetoothGattCallback = new BluetoothGattCallback() {
    @Override
    public void onConnectionStateChange(BluetoothGatt gatt, int status, int newState) {
        if (newState == BluetoothProfile.STATE_CONNECTED) {
            bluetoothGatt.discoverServices();
        } else if (newState == BluetoothProfile.STATE_DISCONNECTED) {}
    }

    @Override
    public void onServicesDiscovered(BluetoothGatt gatt, int status) {
        if (status == BluetoothGatt.GATT_SUCCESS) {
            bluetoothGattService = bluetoothGatt.getService(step_service_UUID);
            Log.d( tag: "debug",  msg: "discover service");
            Step_data = bluetoothGattService.getCharacteristic(step_char_UUID);
            Log.d( tag: "debug",  msg: "access characteristic");

            bluetoothGatt.readCharacteristic(Step_data);

            connected = true;

        } else {
            Log.d( tag: "debug",  msg: "not discover service");
        }
    }

    @Override
    public void onCharacteristicRead(BluetoothGatt gatt,
                                     BluetoothGattCharacteristic characteristic,
                                     int status) {
        if (status == BluetoothGatt.GATT_SUCCESS) {
            broadcast(characteristic);
        }
    }
};
```

Figure 10. Implementation of BluetoothGattCallback.