Activity Recognition Using Google API
Activity Recognition

- Activity Recognition? Detect what user is doing?
  - Part of user’s context
- Examples: sitting, running, driving, walking
- Why? App can adapt it’s behavior based on user behavior
- E.g. If user is driving, don’t send notifications

https://www.youtube.com/watch?v=S8sugXgUVEI
Google Activity Recognition API

- API to detect smartphone user’s current activity
- Programmable, can be used by your Android app
- Currently detects 6 states:
  - In vehicle
  - On Bicycle
  - On Foot
  - Still
  - Tilting
  - Unknown
Google Activity Recognition API

- Deployed as part of Google Play Services
Activity Recognition Using Google Fit
Ref: How to Recognize User Activity with Activity Recognition by Paul Trebilcox-Ruiz on Tutsplus.com tutorials

- Example code for this tutorial on GitHub:
  https://github.com/tutsplus/Android-ActivityRecognition

- Google Activity Recognition can:
  - Recognize user’s current activity (Running, walking, in a vehicle or still)

- Project Setup:
  - Create Android Studio project with blank Activity (minimum SDK 14)
  - In `build.gradle` file, define latest Google Play services (8.4) as dependency

  ```
  compile 'com.google.android.gms:play-services:8.4.0'
  ```
Create new class `ActivityRecognizedService` which extends `IntentService`

**IntentService**: type of service, asynchronously handles work off main thread as Intent requests.

Throughout user’s day, **Activity Recognition API** sends user’s activity to this IntentService in the background.

Need to program this Intent to handle incoming user activity.

```
public class ActivityRecognizedService extends IntentService {

    public ActivityRecognizedService() {
        super("ActivityRecognizedService");
    }

    public ActivityRecognizedService(String name) {
        super(name);
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        // Called to deliver User’s activity
    }
```
Activity Recognition Using Google Fit

Ref: How to Recognize User Activity with Activity Recognition by Paul Trebilco-Ruiz on Tutsplus.com tutorials

- Modify **AndroidManifest.xml** to
  - Declare **ActivityRecognizedService**
  - Add `com.google.android.gms.permission.ACTIVITY_RECOGNITION` permission

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.tutsplus.activityrecognition">
  <uses-permission android:name="com.google.android.gms.permission.ACTIVITY_RECOGNITION" />

  <application android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>

    <service android:name=".ActivityRecognizedService" />
  </application>
</manifest>
```
Requesting Activity Recognition

- In `MainActivity.java`, To connect to Google Play Services:
  - Provide `GoogleApiClient` variable type + implement callbacks

```java
public class MainActivity extends AppCompatActivity implements GoogleApiClient.ConnectionCallbacks,
    GoogleApiClient.OnConnectionFailedListener {

    public GoogleApiClient mApiClient; // Handle to Google Activity Recognition client

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    @Override
    public void onConnected(@Nullable Bundle bundle) {
    }

    @Override
    public void onConnectionSuspended(int i) {
    }

    @Override
    public void onConnectionFailed(@NonNull ConnectionResult connectionResult) {
    }
}
```

- Called if sensor (accelerometer) connection
- Called if Google Play connection fails
Requesting Activity Recognition

- In `onCreate`, initialize client and connect to Google Play Services.

```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    mApiClient = new GoogleApiClient.Builder(this)
        .addApi(ActivityRecognition.API)
        .addConnectionCallbacks(this)
        .addOnConnectionFailedListener(this)
        .build();

    mApiClient.connect();
```
Requesting Activity Recognition

- Once `GoogleApiClient` has connected, `onConnected()` is called
- Need to create a `PendingIntent` that goes to our `IntentService`
- Also set how often API should check user’s activity in milliseconds

```java
@Override
public void onConnected(@Nullable Bundle bundle) {
    Intent intent = new Intent( this, ActivityRecognizedService.class );
    PendingIntent pendingIntent = PendingIntent.getService( this, 0, intent, PendingIntent.FLAG_UPDATE_CURRENT );
    ActivityRecognition.ActivityRecognitionApi.requestActivityUpdates( mApiClient, 3000, pendingIntent );
}
```
Handling Activity Recognition

- Our app tries to recognize the user’s activity every 3 seconds
- `onHandleIntent` called every 3 seconds, Intent delivered
- In `onHandleIntent()` method of `ActivityRecognizedService`
  - Validate that received intent contains activity recognition data
  - If so, extract `ActivityRecognitionResult` from the Intent
  - Retrieve list of possible activities by calling `getProbableActivities()` on `ActivityRecognitionResult` object

```java
@Override
protected void onHandleIntent(Intent intent) {
    if(ActivityRecognitionResult.hasResult(intent)) {
        ActivityRecognitionResult result = ActivityRecognitionResult.extractResult(intent);
        handleDetectedActivities(result.getProbableActivities());
    }
}
```
Handling Activity Recognition

- Simply log each detected activity and display how confident Google Play services is that user is performing this activity.

```java
private void handleDetectedActivities(List<DetectedActivity> probableActivities) {
    for (DetectedActivity activity : probableActivities) {
        switch (activity.getType()) {
            case DetectedActivity.IN_VEHICLE: {
                Log.e( "ActivityRecognition", "In Vehicle: " + activity.getConfidence() );
                break;
            }
            case DetectedActivity.ON_BICYCLE: {
                Log.e( "ActivityRecognition", "On Bicycle: " + activity.getConfidence() );
                break;
            }
            case DetectedActivity.ON_FOOT: {
                Log.e( "ActivityRecognition", "On Foot: " + activity.getConfidence() );
                break;
            }
            case DetectedActivity.RUNNING: {
                Log.e( "ActivityRecognition", "Running: " + activity.getConfidence() );
                break;
            }
            case DetectedActivity.STILL: {
                Log.e( "ActivityRecognition", "Still: " + activity.getConfidence() );
                break;
            }
            case DetectedActivity.TILTING: {
                Log.e( "ActivityRecognition", "Tilting: " + activity.getConfidence() );
                break;
            }
        }
    }
}
```

Sample output:
```
1 E/ActivityRecognition: On Foot: 92
2 E/ActivityRecognition: Running: 87
3 E/ActivityRecognition: On Bicycle: 8
4 E/ActivityRecognition: Walking: 5
```
Handling Activity Recognition

- If confidence is > 75, activity detection is probably accurate
- If user is walking, ask “Are you walking?”

```java
case DetectedActivity.WALKING: {
    Log.e( "ActivityRecogition", "Walking: " + activity.getConfidence() );
    if( activity.getConfidence() >= 75 ) {
        NotificationCompat.Builder builder = new NotificationCompat.Builder(this);
        builder.setContentText( "Are you walking?" );
        builder.setSmallIcon( R.mipmap.ic_launcher );
        builder.setContentTitle( getString( R.string.app_name ) );
        NotificationManagerCompat.from(this).notify(0, builder.build());
    }
    break;
}
case DetectedActivity.UNKNOWN: {
    Log.e( "ActivityRecogition", "Unknown: " + activity.getConfidence() );
    break;
}
```
Sample Output of Program

- Sample displayed on development console

<table>
<thead>
<tr>
<th></th>
<th>E/ActivityRecognition: On Foot: 92</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>E/ActivityRecognition: Running: 87</td>
</tr>
<tr>
<td>3</td>
<td>E/ActivityRecognition: On Bicycle: 8</td>
</tr>
<tr>
<td>4</td>
<td>E/ActivityRecognition: Walking: 5</td>
</tr>
</tbody>
</table>

- Full code at: https://github.com/tutsplus/Android-ActivityRecognition
How Activity Recognition Works
Activity Recognition

- **Goal**: Want our app to detect what activity the user is doing?
- **Classification task**: which of these 6 activities is user doing?
  - Walking,
  - Jogging,
  - Ascending stairs,
  - Descending stairs,
  - Sitting,
  - Standing

- Typically, use machine learning classifiers to classify user’s accelerometer signals
Example Accelerometer Data for Activities

(a) Walking

(b) Jogging

(e) Sitting

(f) Standing
Example Accelerometer Data for Activities

(c) Ascending Stairs

(d) Descending Stairs
Alternate Implementation Options
AppInventor (http://appinventor.mit.edu/)

- MIT project, previously Google
- Use lego blocks to build app, easy to learn
- **Pro:** Quick UI development
- **Con:** sensor access, use third party modules restricted
PhoneGap

- Develop Apps using HTML, CSS, javascript
- **Pro**: Access to most native APIs, sensors, UI
- **Con**: Need to know HTML, CSS javascript
More?

- Multi-platform development tools
- iOS?
References

- Head First Android
- Android Nerd Ranch, 2nd edition
- Busy Coder’s guide to Android version 6.3
- CS 65/165 slides, Dartmouth College, Spring 2014
- CS 371M slides, U of Texas Austin, Spring 2014