POTHOLE MAPPER

Beta Release Document

Team Name: Droiders

Term: Spring 2014
Course: ECE 573 Software Engineering Concepts
University Of Arizona

Team Members:

Navin Chaganti
Email: nchaganti@email.arizona.edu
Graduate Student

Dhawal Srivatsava
Email: dhawalsrivastava@email.arizona.edu
Graduate Student

Developers	Release Version
Dhawal Srivastava, Navin Chaganti	1.1

Table of content

Contents

Preface

About the Documentation

The intended purpose of the documentation is to explain the main features of the beta release of the application “Pothole Mapper” and to provide the user with step by step information on installation and execution of the product. This documents also enlist the limitation of the beta release and the implementation that are yet to be incorporated before the final release.

Product Overview

The application utilizes the smart phones accelerometer sensor to check the conditions of road in a moving vehicle. The app describes poor road conditions, road hazards such as potholes, speed bumps, etc. and all the different scenarios can be differentiated through different icons. The resulting data is displayed as an overlay of colors (indicating road conditions) and icons indicating road hazards.

Main features:

1. Map of potholes,
2. Adding potholes,
3. Checking existing potholes,
4. Tracking changes for added potholes, checked potholes or potholes nearby.

Features Implemented in Beta Release

Following core features have been implemented in this Beta Release.

- Accelerometer data Fetch: The beta release of this product has the accelerometer functionality implemented with it. This application can render the data from accelerometer sensor for three co-ordinates along the length, breadth (plane parallel to the screen of the device) and height (Plane perpendicular to the screen of the device). The fetch implementation is controlled by button labeled "Start Detecting"/ "Stop Detecting". Pressing "Start Detecting" fetches the value from the accelerometer and displays to the user. Pressing "Stop Detecting" stop fetching the value from the accelerometer and displays to the user.

- Relative Speed Fetch: The beta release has a speed fetch functionality implemented with it. This application determines an approximation of the speed of motion of the device based on the data obtained from the accelerometer. In this release the speed is being calculated for even a slight variation in acceleration, hence the speed obtained will not be equal to the actual speed determined by speedometer or other gauges. This functionality when integrated with detection algorithm of the pothole will render further accuracy in the detection.

- GPS data Fetch: The beta release has the GPS data fetch functionality. The GPS functionality is being used to fetch the latitude and longitude values of the location of a road event. The selection of Road events causes the latitude longitude values to be stored as an array list. These latitude and longitude values are further mapped as per the kind of event captured.

- Signal Processor : The beta release has the Signal Processor functionality. The Signal Processor Functionality is used to process the accelerometer values of a particular road event. Different Algorithms used for the filtering out the noise.

- Road Event :Defines the kind of road Hazard which has occurred. The app give the user to select the different kind of Road hazard which took place during the drive and the end of signal processing of the accelerometer values.

User Interface of Beta Release

- The product's user interface is enhanced in beta release. On the product start-up, user is displayed with the main screen of the application. It contains the control button labeled "Start Detecting"/ "Stop Detecting" for controlling the fetching of the data. The Process button performs all the 5 signal processing algorithms on all the values. Test Initiate selects the data file containing the Accelerometer values. The Buttons Step one to Step five are the Algorithms implements and each algorithm can be executed selecting them. It also contains a control button labeled "Test Initiate" for displaying new intent which control the map plotting functionality.

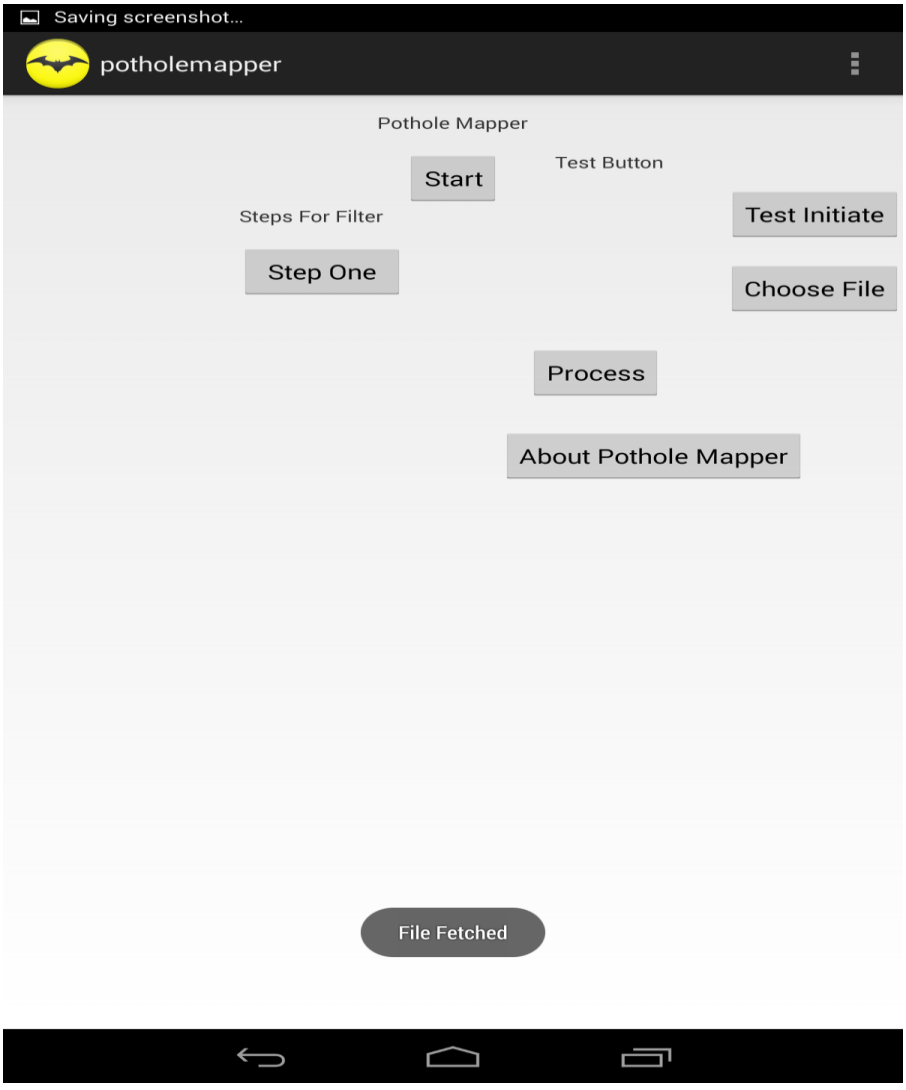

Figure 1 Screen displayed at Start-up

The above image indicates the step wise execution of all the algorithms.

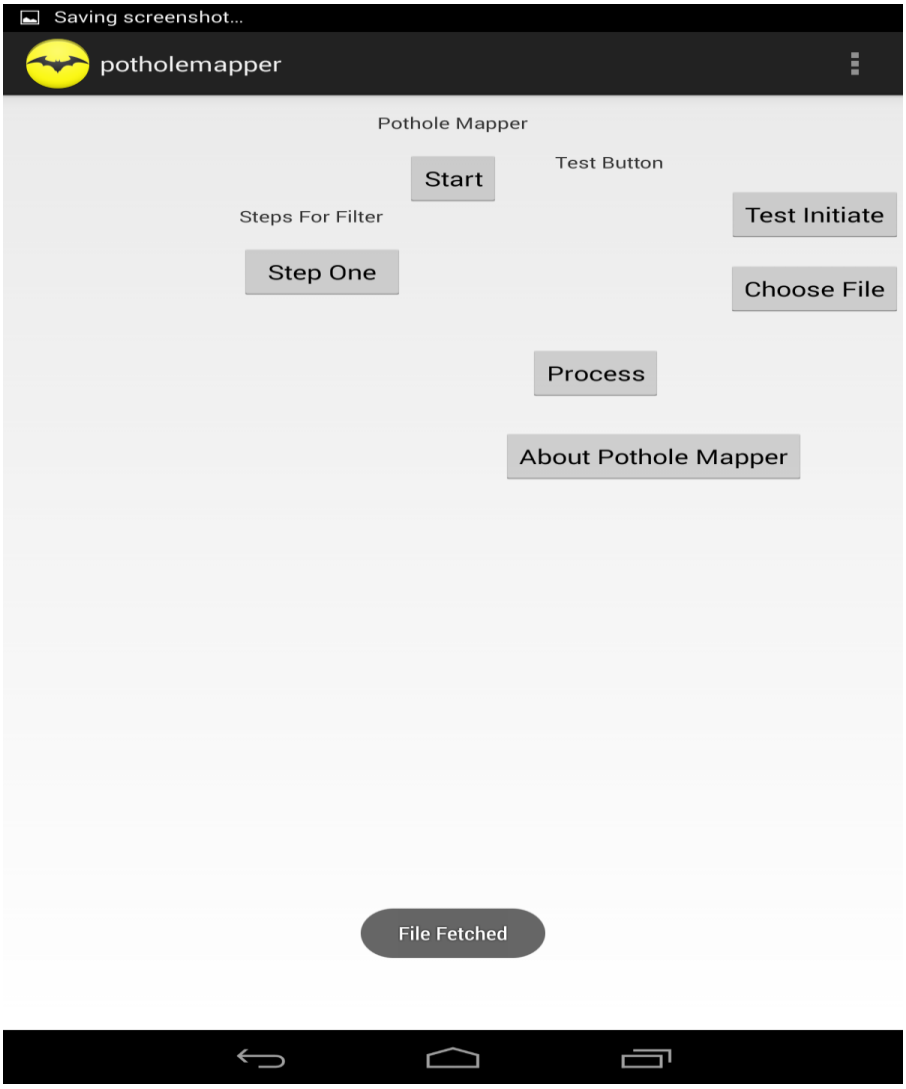

Figure 2 App after fetching the accelerometer data and waiting for analysis

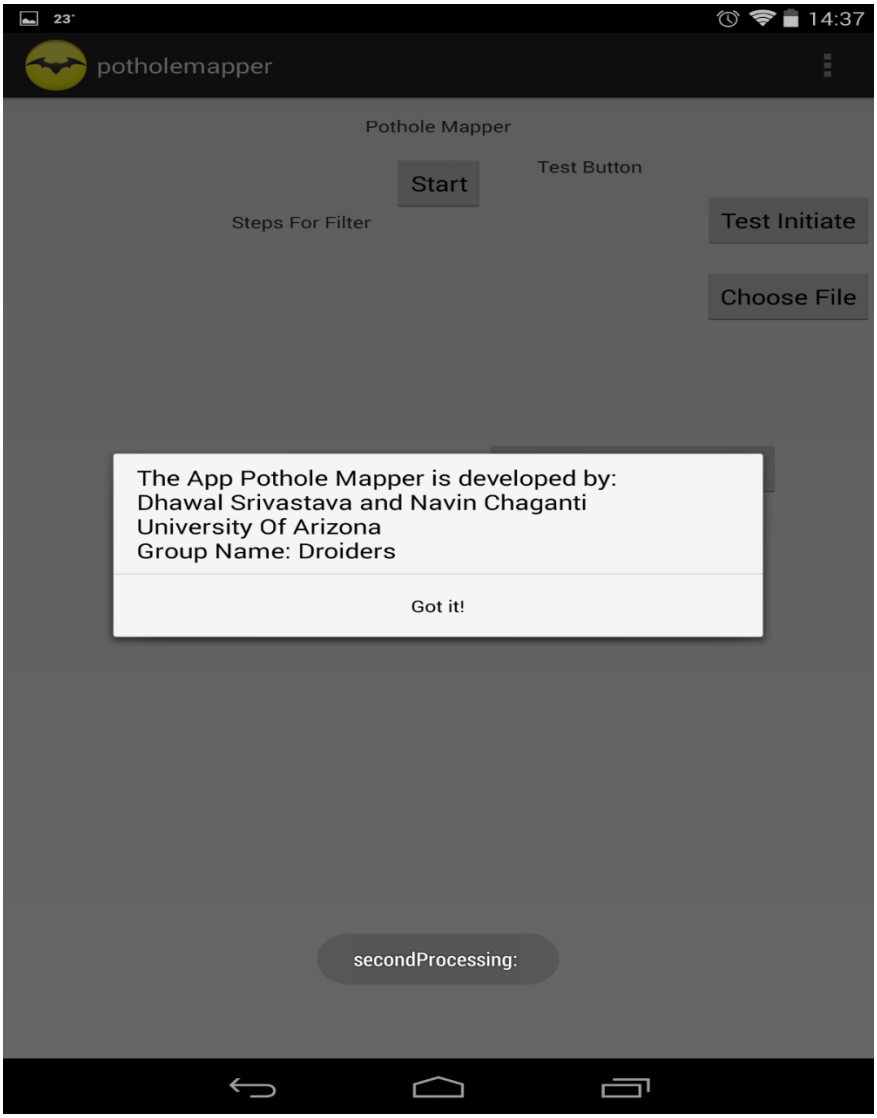

Figure 3 About function of the app (Credits)

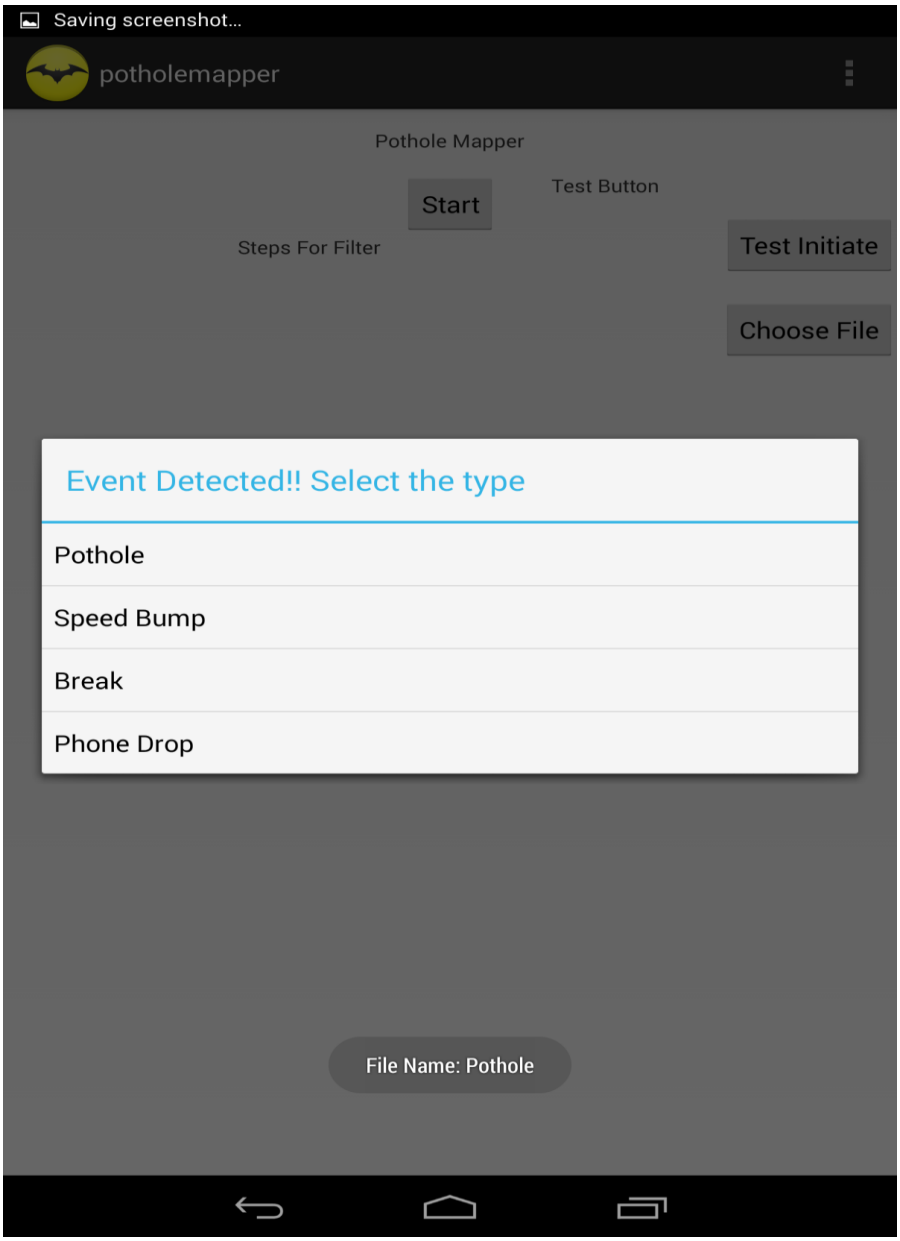

Figure 4 App showing RoadEvents Detected and allows user to manually chose them

The above image describes the Road event selection behavior of the app. After signal processing of the accelerometer data the values user is given the privilege to select the kind of road event to be selected.

Features to be Implemented post Beta Release

Following features will certainly be implemented after this Beta Release.

Map Plot: After the beta release the current API conflict will be resolved to generate proper Map interface with the plots of the data obtained from the GPS fetch.

Demo Mode: Future releases will also incorporate demo mode for user to simulate the road condition on the device and check the functionality if the application.

Application Installation

Following steps should be followed while installing the application on an android device

These steps are defined for Nexus 7 for android version: 4.4.2

1. Checkout the project folder Pothole Mapper from SVN location trunk/c2d/ to your local computer directory.
2. Open the checked out directory and locate PotholeMapper.apk at: trunk/c2d/src/android/PotholeMapper/bin/PotholeMapper.apk.
3. Connect the mobile device to the local computer.
4. Check the Unknown Sources box in the security settings of the device to ensure that project obtained from sources other than Play Store works on the deivce.
5. Copy the PotholeMapper.apk to the device.
6. Open FileManager on the device to install the PotholeMapper.apk. When asked for confirmation, Click Install.
7. The application is installed.

Application Execution

Following Steps should be performed for executing the beta release

1. Enable the GPS on the device.
2. Locate the app PotholeMapper on mobile browser.
3. Click on the app to start the application.
4. On the main screen locate the “Start Detecting” button.
5. Click on the “Start Detecting” button to start the fetching of accelerometer data.
6. Click on the “Stop Detecting” button to stop the fetching of accelerometer data.
7. Locate “Process” button on the screen.
8. Click on “Process” button to view second screen with list of options for road events.
9. Click on different road events Bump, Kerb, Pothole, Break.

Beta Testing

Requirement 1.1: High pass filter implemented for Horizontal acceleration on Matlab
Validation 1.1: Run test 1.1
Test 1.1:
(A)

1. Place the device on the dash board. Adjust the position of the device so direction of motion of vehicle be in line with the X-axis of the device. Start the app.
2. Drive at speed of 10 Km/hr on smooth road & make sure 'Low Speed' radio button is selected.
3. After 10 seconds. Increase speed to 40 Km/hr and Drive for 20 seconds. Make sure to select 'High Speed' radio button.
4. Save the data.

(B)

1. Implement Algorithm-I on Matlab.

Algorithm–I : $H_h(Z) = \left(\frac{1+a}{2}\right)\left(\frac{1-z^{-1}}{1-az^{-1}}\right)$

2. Filter the data obtained through Matlab implementation.

Result 1.1: The output should not have data with flag of Low Speed.

Requirement 1.2: High pass filter implemented for Vertical acceleration on Matlab
Validation 1.2: Run test 1.2
Test 1.2:
(A)

1. Place the device on the dash board. Adjust the position of the device so direction of motion of vehicle be in line with the X-axis of the device. Start the app.
2. Drive at the speed of 40 Km/hr on smooth road and make sure to press 'Smooth Road' button at regular intervals of 2 sec.
3. After 20 seconds. Drive at the speed of 40 Km/hr on rough patch of road and don't press 'Smooth Road' button.
4. Save the data.

(B)

1. Implement Algorithm-II on Matlab.

Algorithm–II: $H_v(Z) = \left(\frac{1+b}{2}\right)\left(\frac{1-z^{-1}}{1-bz^{-1}}\right)$

2. Filter the data obtained through Matlab implementation.

Result 1.2: The output should not have data with flag of 'Smooth Road'.

Requirement 1.3: Match filter with for Pothole event on Matlab
Validation 1.3: Run test 1.3
Test 1.3:
(A)

1. Place the device on the dash board. Adjust the position of the device so direction of motion of vehicle be in line with the X-axis of the device. Start the app.
2. Drive at the speed of 40 Km/hr over a pothole and make sure to press 'Pothole' button.
3. Save the data.

(B)

1. Implement Algorithm-III on Matlab.

Algorithm–III: $H_{z-axiz}(Z) = a_0 + a_1Z^{-1} + a_2Z^{-2} + a_3Z^{-3} \dots\dots (-a_{n-3})Z^{-(n-3)} + (-a_{n-2})Z^{-(n-2)} + (-a_{n-1})Z^{-(n-1)}$

2. Filter the data obtained through Matlab implementation.

Result 1.3: The output should have separate flag for a window of 400 samples around the manual pothole flag position.

Requirement 1.4: Match filter with for Speed pattern at Pothole event on Matlab
Validation 1.4: Run test 1.4
Test 1.4:

(A)

1. Place the device on the dash board. Adjust the position of the device so direction of motion of vehicle be in line with the X-axis of the device. Start the app.
2. Drive at the speed of 40 Km/hr over a pothole and make sure to press 'Pothole' button.
3. Save the data.

(B)

1. Implement Algorithm-IV on Matlab.

Algorithm–IV: $H_{x-axiz}(Z) = b_{p_z0} + b_{p_z1}Z^{-1} + b_{p_z2}Z^{-2} + b_{p_z3}Z^{-3} \dots\dots (-b_{p_zn-3})Z^{-(n-3)} + (-b_{p_zn-2})Z^{-(n-2)} + (-b_{p_zn-1})Z^{-(n-1)}$

2. Filter the data obtained through Matlab implementation.

Result 1.4: The output should have separate flag on a Horizontal acceleration array window of 40 samples around the manual pothole flag position.

Requirement 1.5: Window reject below threshold peak.
Validation 1.5: Run test 1.5
Test 1.5:

(A)

1. Place the device on the dash board. Adjust the position of the device so direction of motion of vehicle be in line with the X-axis of the device. Start the app.
2. Drive at the speed of 40 Km/hr over a deep pothole and make sure to press 'Pothole' button.
3. Save the data.
4. Drive at the speed of 80 Km/hr over a shallow pothole and make sure to press 'Pothole' button.
5. Save the data.

(B)

1. Implement peak reject filter on Matlab. Filter the data obtained through Matlab implementation.

Result 1.5: The output should have separate flag at the manual pothole flag position where the speed was 40 Km/hr but not at the position where the speed was 80 Km/hr and the pothole depth was shallow.

Requirement 2: All 5 Algorithm implemented in Java on device
Validation 2.1: Run test 1.1. Then run test 2.1
Test 2.1:
JUnit Test : test_Checkingalgorithmone()

1. Implement Algorithm-I in Java.

Algorithm–I: $H_h(Z) = \left(\frac{1+a}{2}\right)\left(\frac{1-z^{-1}}{1-az^{-1}}\right)$

2. Run Test 1.1 (A)
3. Perform first level processing by pressing 'Step-1' button and calling firstProcessing() function (According to the sequence model 5.1).
4. Save the output data.
5. Upon completion, compare output file of Test 5.1 with out.bin, which was obtained from a reference implementation in MATLAB in Test 4.1 (B).

Result 5.1: The output of implementation in Java should match the output obtained from reference implementation in Matlab.

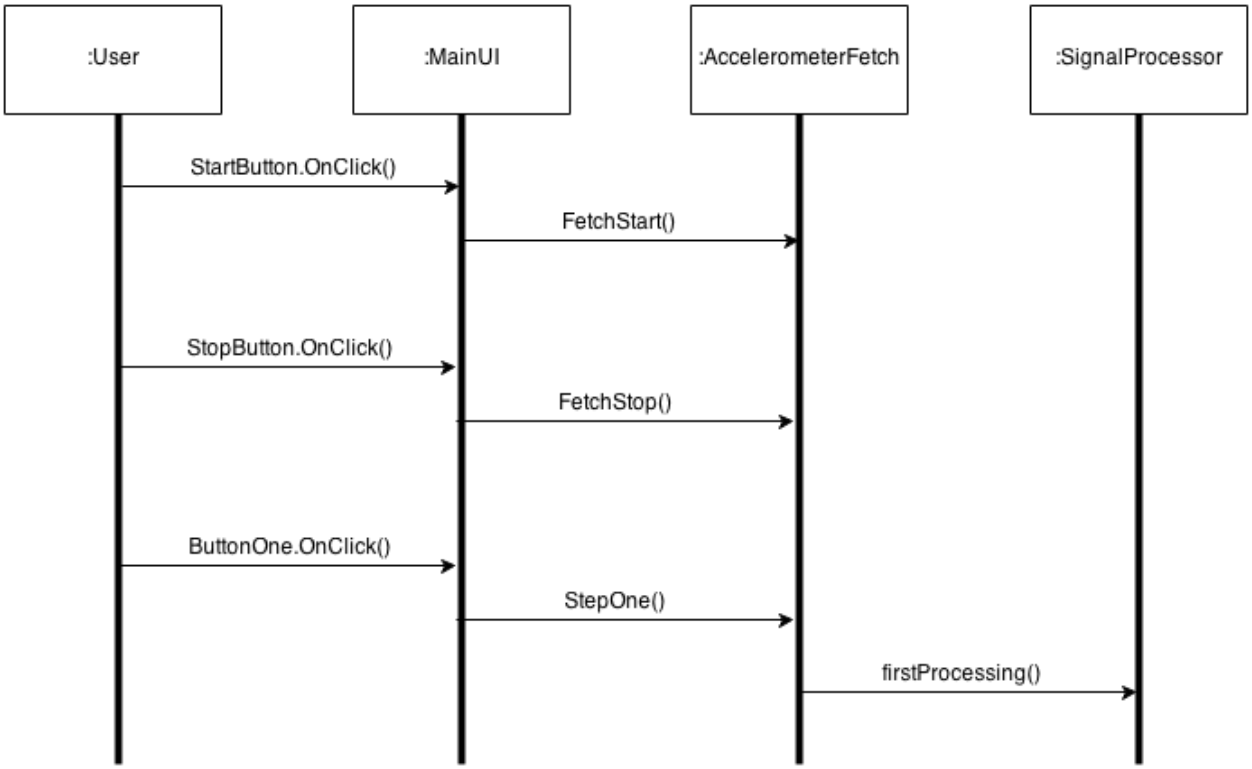

Figure 5: Sequence Model 2.1

Validation 2.2 : Run test 1.2. Then run test 2.2
Test 2.2 :
JUnit Test : test_Checkingalgorithm_two()

1. Implement Algorithm-II in Java.

Algorithm–II: $H_v(Z) = \left(\frac{1+b}{2}\right)\left(\frac{1-z^{-1}}{1-bz^{-1}}\right)$

2. Run test 4.2 (A)
3. Perform second level processing by pressing 'Step-2' button and calling secondProcessing() function (According to the sequence model 2.2).
4. Save the output data.
5. Upon completion, compare output file of Test 5.2 with out.bin, which was obtained from a reference implementation in MATLAB in Test 1.2 (B).

Result 2.2: The output of implementation in Java should match the output obtained from reference implementation in Matlab.

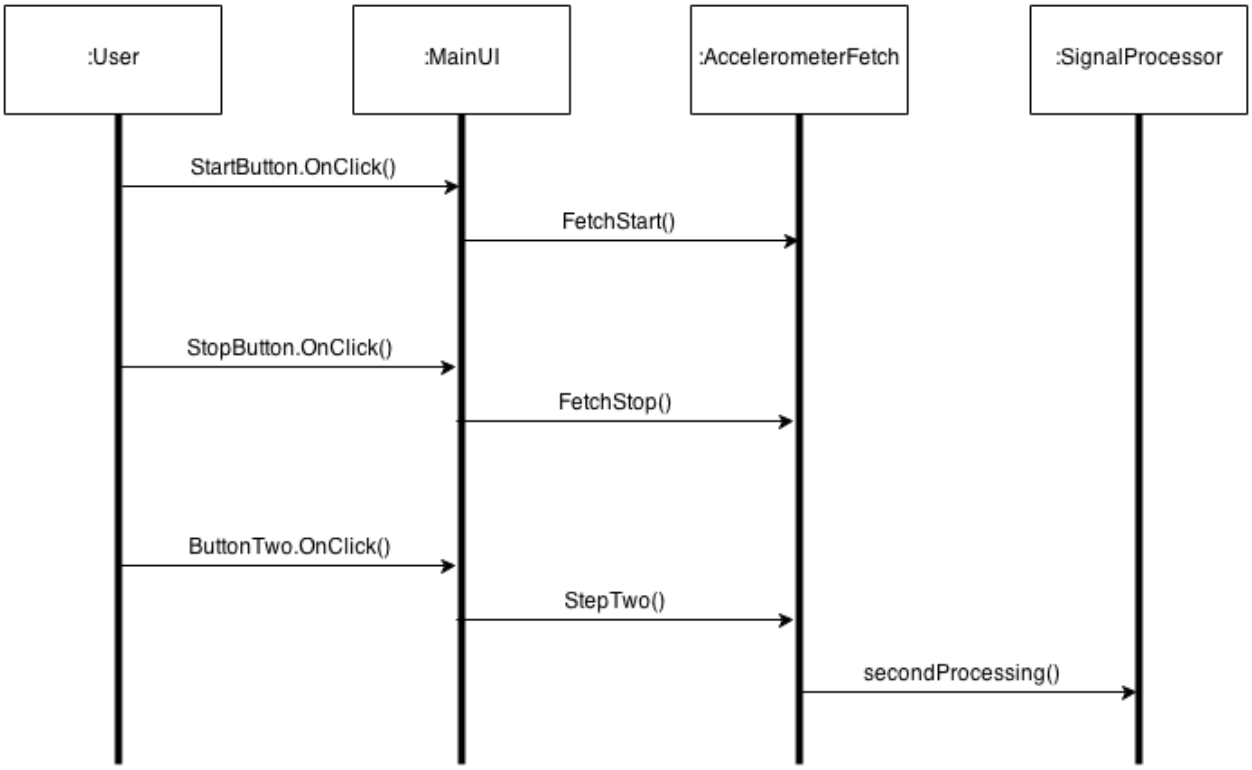

Figure 6: Sequence Model 2.2

Validation 2.3 : Run test 1.3. Then run test 2.3
Test 2.3 :
JUnit Test : test_Checkingalgorithm_three()

1. Implement Algorithm-III in Java.

Algorithm–III: $H_{z-axiz}(Z) = a_0 + a_1Z^{-1} + a_2Z^{-2} + a_3Z^{-3} \dots\dots (-a_{n-3})Z^{-(n-3)} + (-a_{n-2})Z^{-(n-2)} + (-a_{n-1})Z^{-(n-1)}$

2. Run test 1.3 (A)
3. Perform third level processing by pressing 'Step-3' button and calling thirdProcessing() function (According to the sequence model 2.3).
4. Save the output data.
5. Upon completion, compare output file of Test 2.3 with out.bin, which was obtained from a reference implementation in MATLAB in Test 1.3 (B).

Result 2.3: The output of implementation in Java should match the output obtained from reference implementation in Matlab.

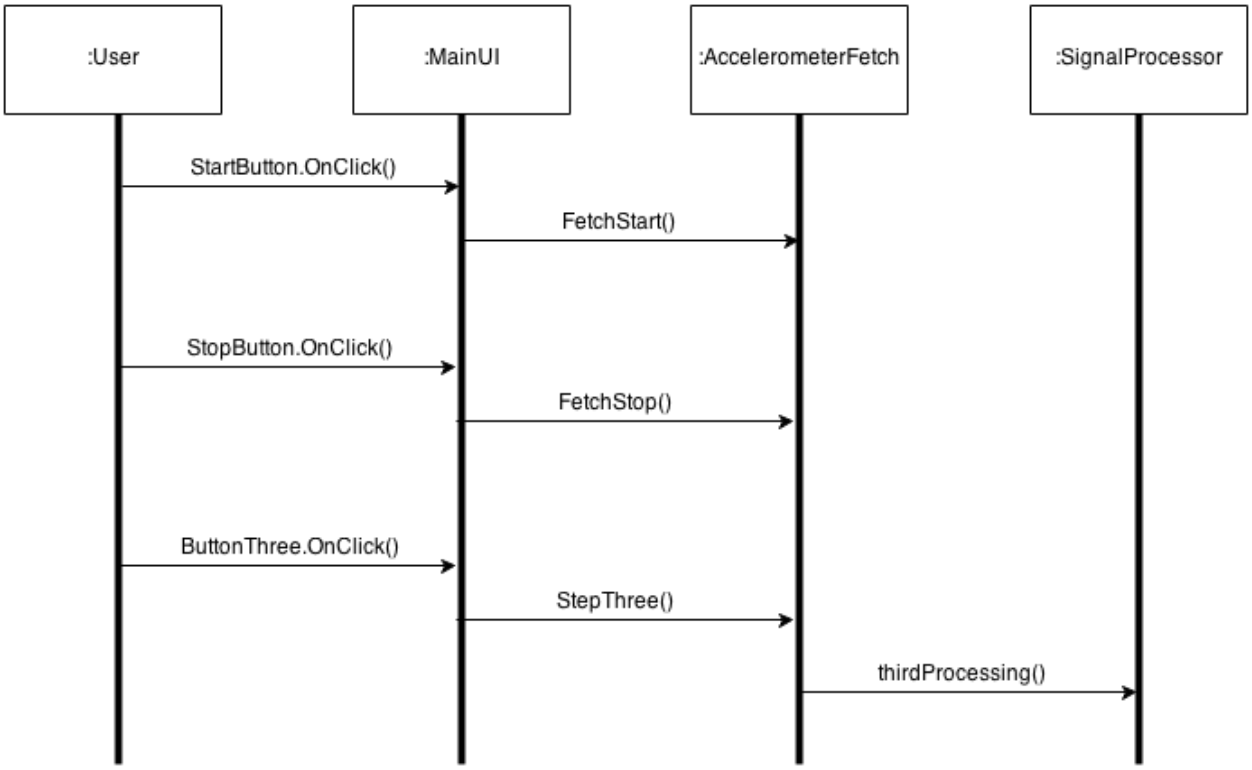

Figure 7: Sequence Model 2.3

Validation 2.4 : Run test 1.4. Then run test 2.4
Test 2.4 :
JUnit Test : test_Checkingalgorithm_four()

1. Implement Algorithm-IV in Java.

Algorithm–IV: $H_{x-axiz}(Z) = b_{p_z0} + b_{p_z1}Z^{-1} + b_{p_z2}Z^{-2} + b_{p_z3}Z^{-3} \dots\dots (-b_{p_zn-3})Z^{-(n-3)} + (-b_{p_zn-2})Z^{-(n-2)} + (-b_{p_zn-1})Z^{-(n-1)}$

2. Run test 1.4 (A)
3. Perform fourth level processing by pressing 'Step-4' button and calling fourthProcessing() function (According to the sequence model 2.4).
4. Save the output data.
5. Upon completion, compare output file of Test 2.4 with out.bin, which was obtained from a reference implementation in MATLAB in Test 1.4 (B).

Result 2.4: The output of implementation in Java should match the output obtained from reference implementation in Matlab.

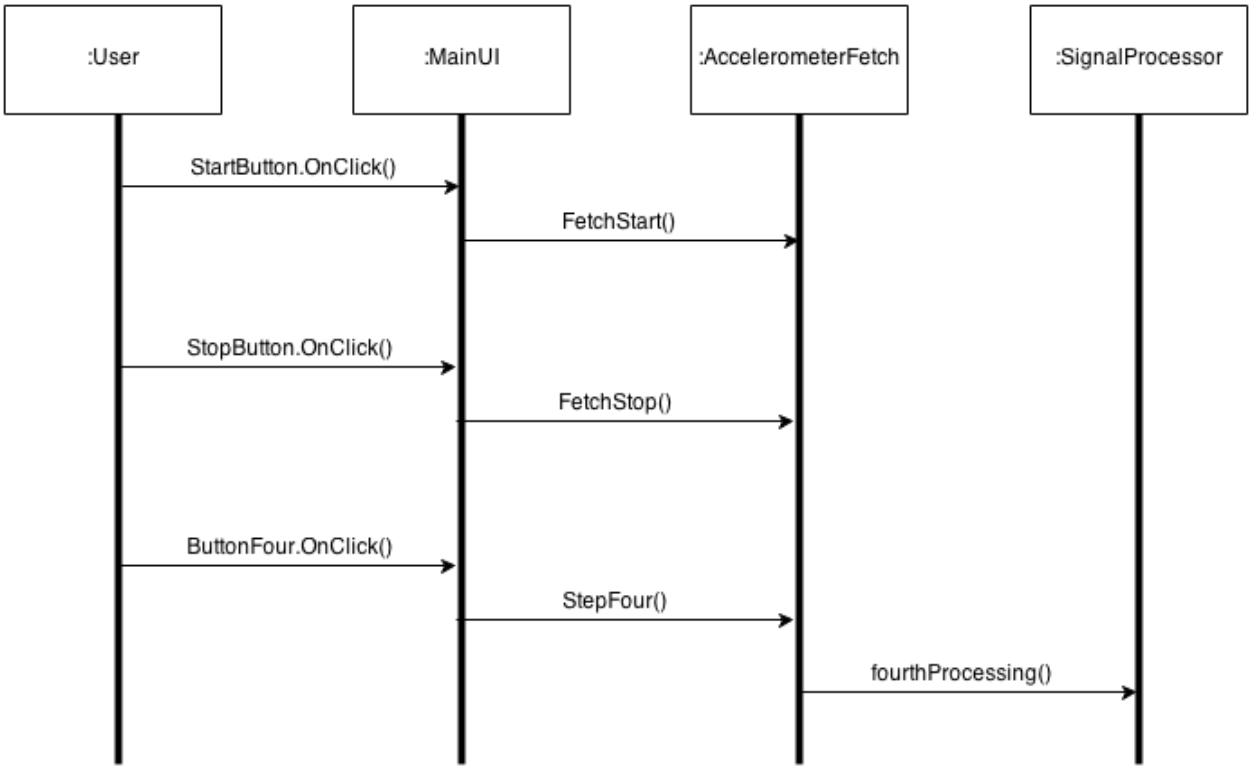

Figure 8: Sequence Model 2.4

Validation 2.5 : Run test 1.5. Then run test 2.5
Test 2.5 :
JUnit Test : test_Checkingalgorithm_five()

1. Implement peak reject filter in Java.
2. Run test 1.5 (A)
3. Perform fifth level processing by pressing 'Step-5' button and calling fifthProcessing() function (According to the sequence model 2.5).
4. Save the output data.
5. Upon completion, compare output file of Test 2.5 with out.bin, which was obtained from a reference implementation in MATLAB in Test 1.5 (B).

Result 2.5: The output of implementation in Java should match the output obtained from reference implementation in Matlab.

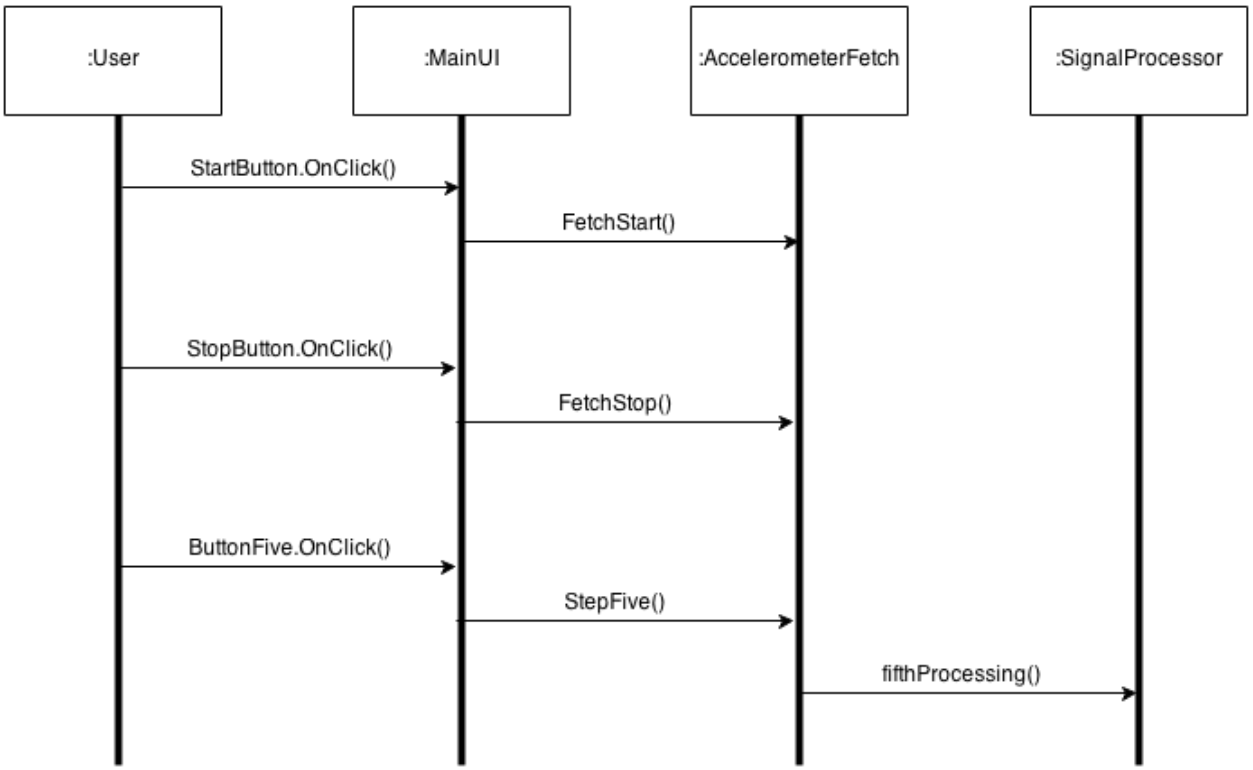

Figure 9: Sequence Model 2.5

Validation 2.6 : Run test 2.6
Test 2.6 :

1. Implement all 5 algorithm in Java.
2. Place the device on the dash board. Adjust the position of the device so direction of motion of vehicle be in line with the X-axis of the device. Start the app.
3. Drive at the speed of 40 Km/hr over a pothole.
4. Save the data.
5. Perform whole processing by pressing 'Process' button and calling all five processing function (According to the sequence model 2.6).
6. Save the output data.

Result 2.6: The user should receive a Toast saying "Pothole Detected".

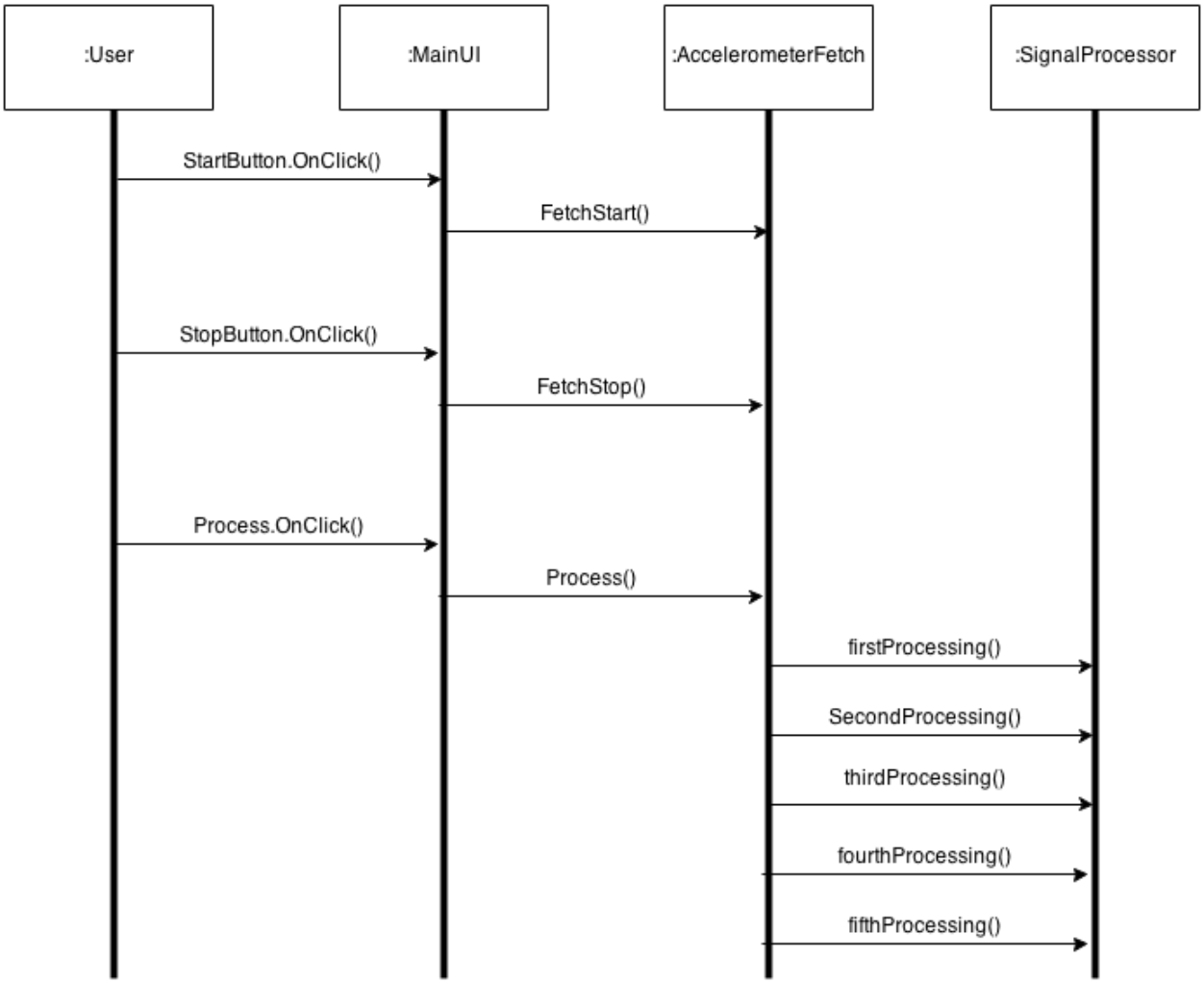

Figure 10: Sequence Model 2.6

Requirement 3: Record GPS co-ordinates of detected events
Validation 3: Run Test 3
Test 6: Sequence Model 3

1. Open app. Press 'ManualMap button.
2. Press buttons 'Smooth Road', 'Phone Drop', 'Pothole', 'Curb Hit' and 'Speed Bump' at interval of 10 meters.
3. Press show map button.

Result 6: Readings on GPS are displayed as toast on the bottom of the screen. All the Recorded GPS are plotted on the screen with their Nametags.

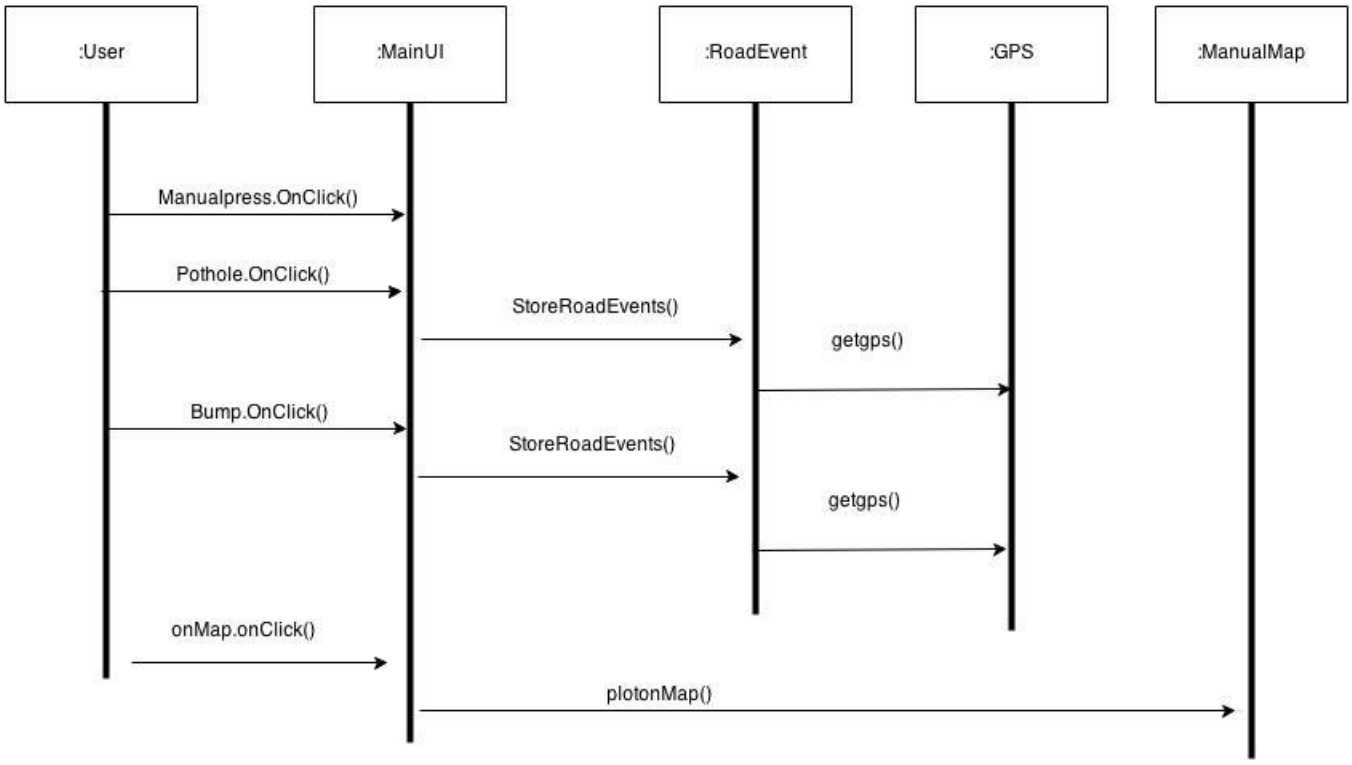

Figure 11: Sequence Model 3