POTHOLE MAPPER

Requirement Verification

Team Name: Droiders

Term:Spring 2014
Course: ECE 573 Software Engineering Concepts

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

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

Table of Contents

Table of Contents 2

Table of Figures 3

Requirements 4

Requirements Verification 7

Table of Figures

Figure 1: Sequence Model 1.1 10
Figure 2: Sequence Model 1.2 11
Figure 3: Sequence Model 1.3 12
Figure 4: Sequence Model 1.4 13
Figure 5: Sequence Model 1.5 14
Figure 6: Sequence Model 1.6 15
Figure 7: Sequence Model 2 16
Figure 8: Sequence Model 3 17
Figure 9: Sequence Model 4 18

Requirements

Critical Requirements (B-Class Requirements)

B.1. Transfer function for all 5 algorithm be implemented on Matlab.

B.1.1. Algorithm-1 transfer function: $H_h(Z) = \left(\frac{1+a}{2}\right)\left(\frac{1-z^{-1}}{1-az^{-1}}\right)$

B.1.1.1. Matlab implements this transfer function successfully based on the parameters calculated. This mean if X-axis acceleration values is filtered through this transfer function, then output should not contain any window which contains either Low-Speed or Curb Hit flag.

B.1.2. Algorithm-2 transfer function: $H_v(Z) = \left(\frac{1+b}{2}\right)\left(\frac{1-z^{-1}}{1-bz^{-1}}\right)$

B.1.2.1. Matlab implements this transfer function successfully based on the parameters calculated. This mean if Z-axis acceleration values is filtered through this transfer function, then output should not contain any window which contains flags like either Smooth Road or Turn.

B.1.3. Algorithm-3 transfer function:

$$H_{z-axiz}(Z) = a_0 + a_1 Z^{-1} + a_2 Z^{-2} + a_3 Z^{-3} \dots\dots (-a_{n-3})Z^{-(n-3)} + (-a_{n-2})Z^{-(n-2)} + (-a_{n-1})Z^{-(n-1)}$$

B.1.3.1. Matlab implements this transfer function successfully based on the parameters calculated. This mean if Z-axis acceleration values is filtered through this transfer function, then output should not contain any window which contains flags like Break and Phone Drop.

B.1.4. Algorithm-4 transfer function:

$$H_{x-axiz}(Z) = b_{p_z 0} + b_{p_z 1} Z^{-1} + b_{p_z 2} Z^{-2} + b_{p_z 3} Z^{-3} \dots\dots (-b_{p_z n-3})Z^{-(n-3)} + (-b_{p_z n-2})Z^{-(n-2)} + (-b_{p_z n-1})Z^{-(n-1)}$$

B.1.4.1. Matlab implements this transfer function successfully based on the parameters calculated. This mean if X-axis acceleration values is filtered through this transfer function, then output should not contain any window which contains flags like Speed bump and Joint Expansion/Railway Crossing.

B.1.5. Algorithm-5 implementation: Discarding peak with less than T_s*Speed. (T_s = 5)

B.1.5.1. Matlab implements this function successfully based on the parameters calculated. This mean if Z-axis acceleration values is processed through this function, then output should not contain any window which contains flags like High speed trivial hit.

B.2. Transfer function for all 5 algorithm be implemented on smart phone. The results obtained from Algorithm-I, Algorithm-II, Algorithm-III, Algorithm-IV, Algorithm-V on smart phone should match respective results obtained through Matlab implementation in B.4.

B.3. Record GPS co-ordinates of detected events.

B.4. On detection of a road hazard, App gives user options to choose the type of "Anomaly" encountered.

B.5. When vehicle is stationary and experience a jerk, App does not record this event as pothole encounter.

Bells & Whistles (A-Class Requirements)

A.1 App automatically differentiates pothole encounters from other false targets with in the error bounds that will be established.

A.1.1. When vehicle stops short due to sudden breaks, App does not detect this scenario as pothole.

A.1.2. If user drops the phone while driving, the app does not consider this activity as a pothole encounter.

A.2. When vehicle is driven on long stretches of low quality road, the app should consider the whole stretch as a road hazard.

A.3. App differentiates pothole encountered based on their size as Small Medium Large.

A.4 During a call, the App goes on pause state and stops recording data.

A.5. GPS data recorded are plotted on the map.

A.6 Potholes detected are categorized based on current vehicle speed.

A.7. App supports a demo mode for user interaction

A.7.1. In demo mode, events are injected using pre-recorded data and application detects the event accurately as either speed bump, pothole or driving over a curb.

Requirements Verification

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: $H_{z-axiz}(Z) = a_0 + a_1 Z^{-1} + a_2 Z^{-2} + a_3 Z^{-3} \dots\dots (-a_{n-3})Z^{-(n-3)} + (-a_{n-2})Z^{-(n-2)} + (-a_{n-1})Z^{-(n-1)}$

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: $H_{x-axiz}(Z) = b_{p_z 0} + b_{p_z 1} Z^{-1} + b_{p_z 2} Z^{-2} + b_{p_z 3} Z^{-3} \dots\dots (-b_{p_z n-3})Z^{-(n-3)} + (-b_{p_z n-2})Z^{-(n-2)} + (-b_{p_z n-1})Z^{-(n-1)}$

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.

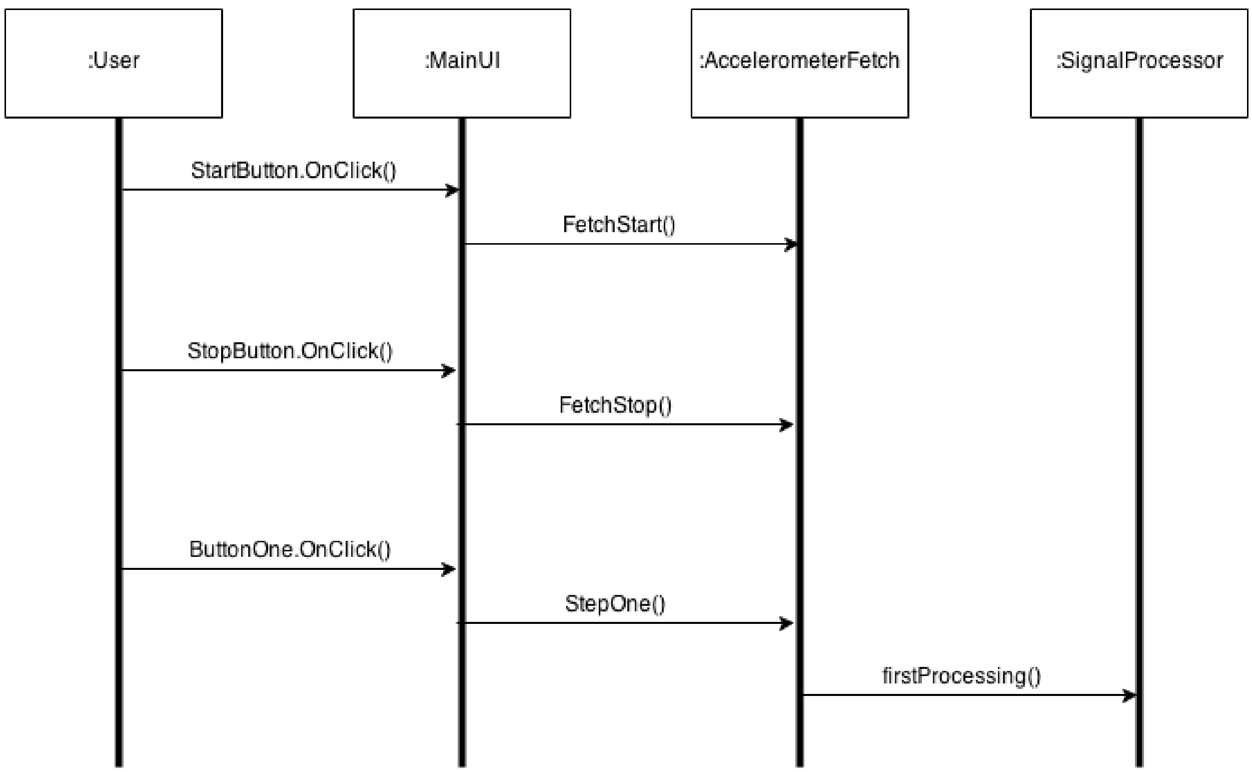

Figure 1: Sequence Model 2.1

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.

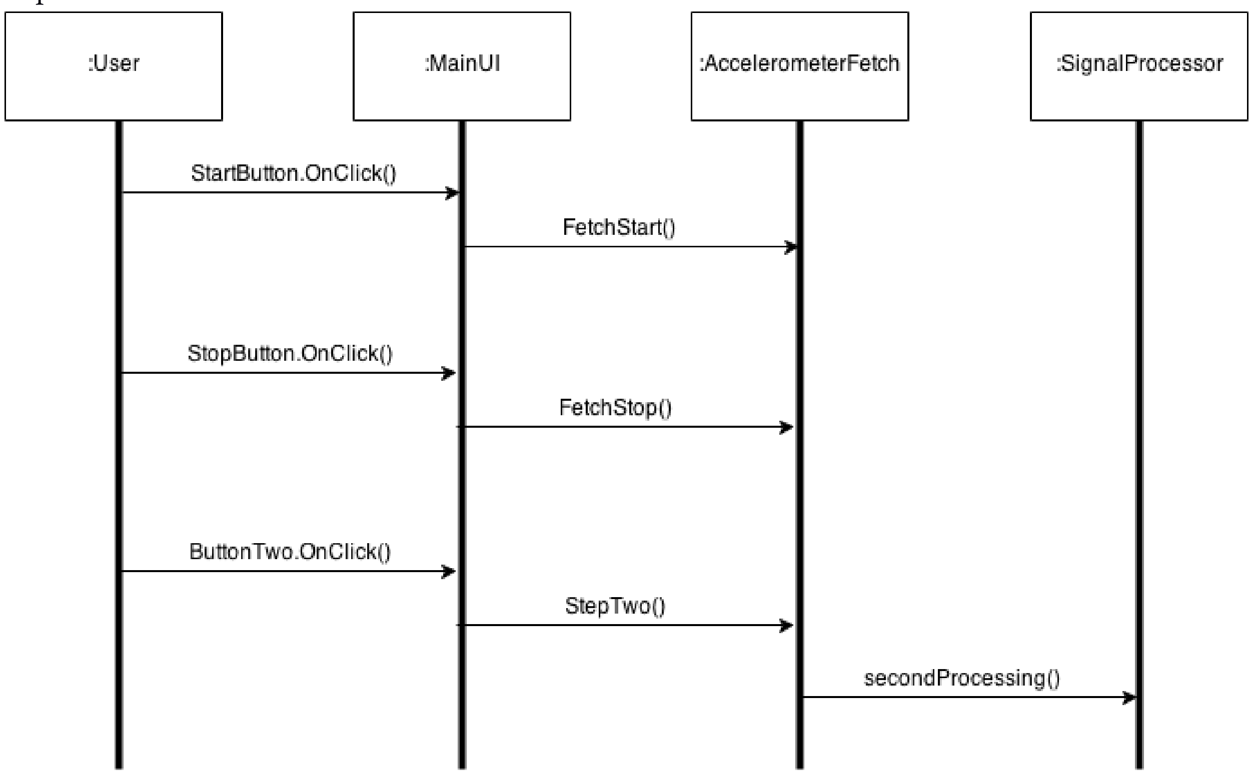

Figure 2: Sequence Model 2.2

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: $H_{z-axiz}(Z) = a_0 + a_1 Z^{-1} + a_2 Z^{-2} + a_3 Z^{-3} \ldots \ldots (-a_{n-3}) Z^{-(n-3)} + (-a_{n-2}) Z^{-(n-2)} + (-a_{n-1}) Z^{-(n-1)}$

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.

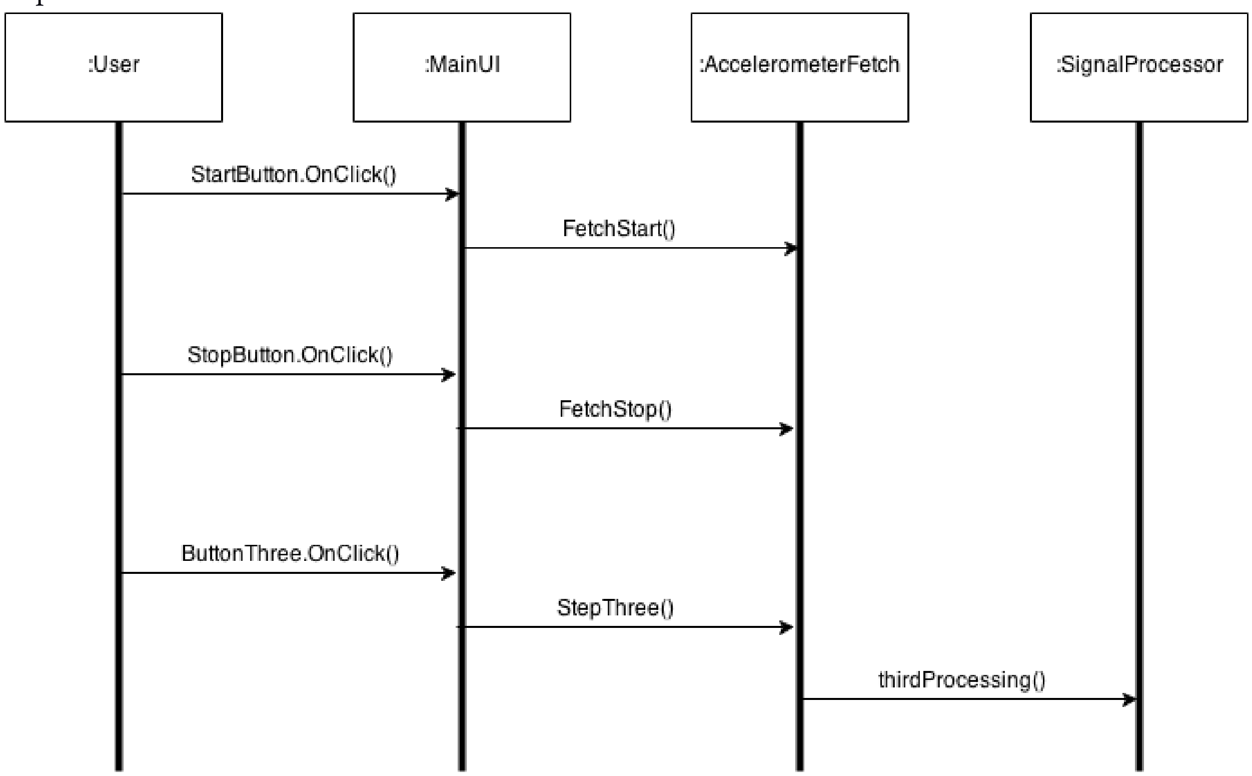

Figure 3: Sequence Model 2.3

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: $H_{x-axiz}(Z) = b_{p_z 0} + b_{p_z 1}Z^{-1} + \; b_{p_z 2}Z^{-2} + \; b_{p_z 3}Z^{-3} \ldots\ldots \; (-b_{p_z n-3})Z^{-(n-3)} + (-b_{p_z n-2})Z^{-(n-2)} + \; (-b_{p_z n-1})Z^{-(n-1)}$

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.

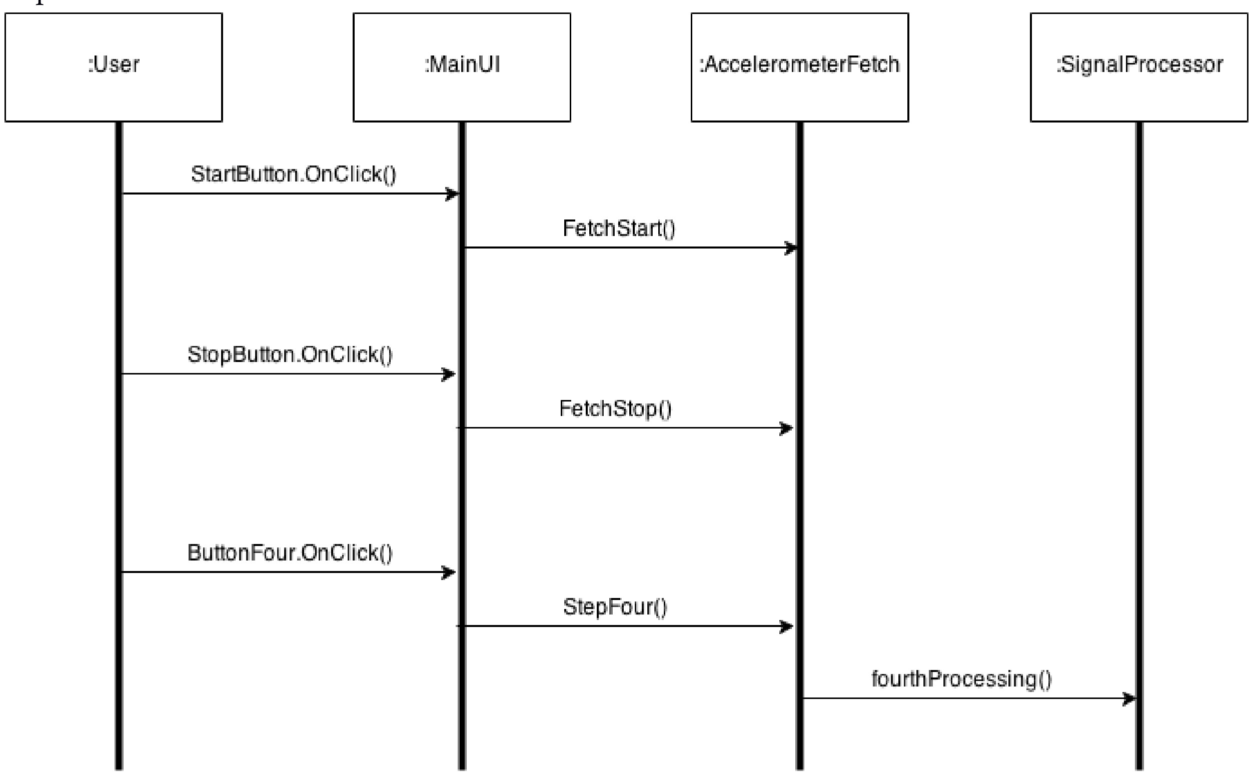

Figure 4: Sequence Model 2.4

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.

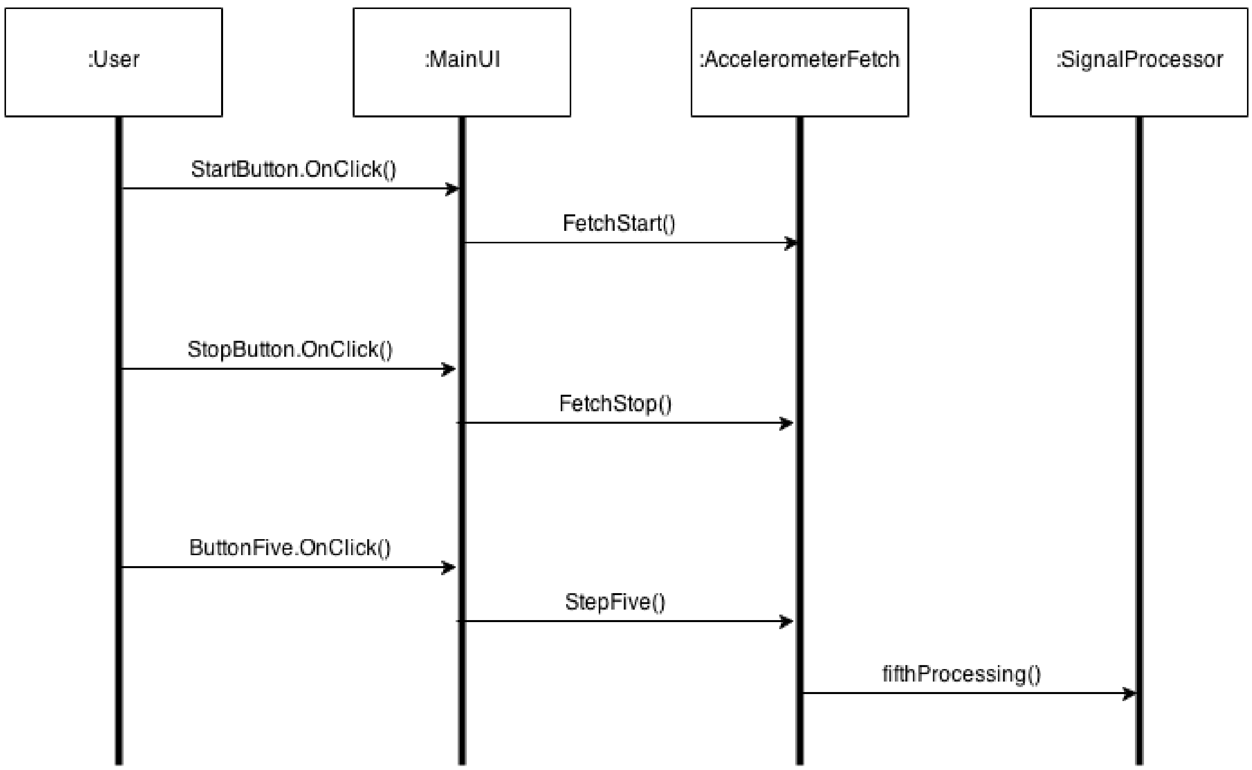

Figure 5: Sequence Model 2.5

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”.

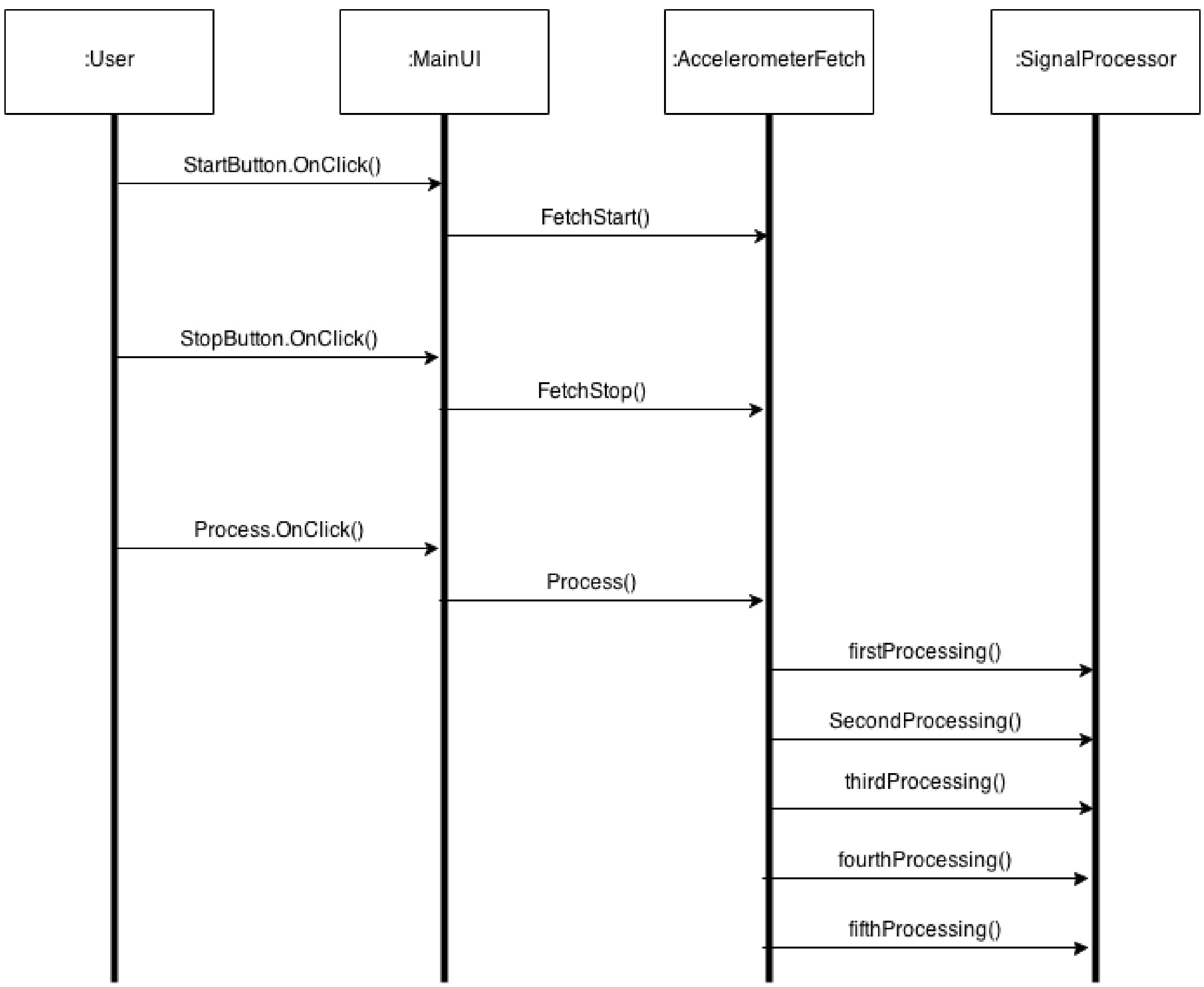

Figure 6: Sequence Model 2.6

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.

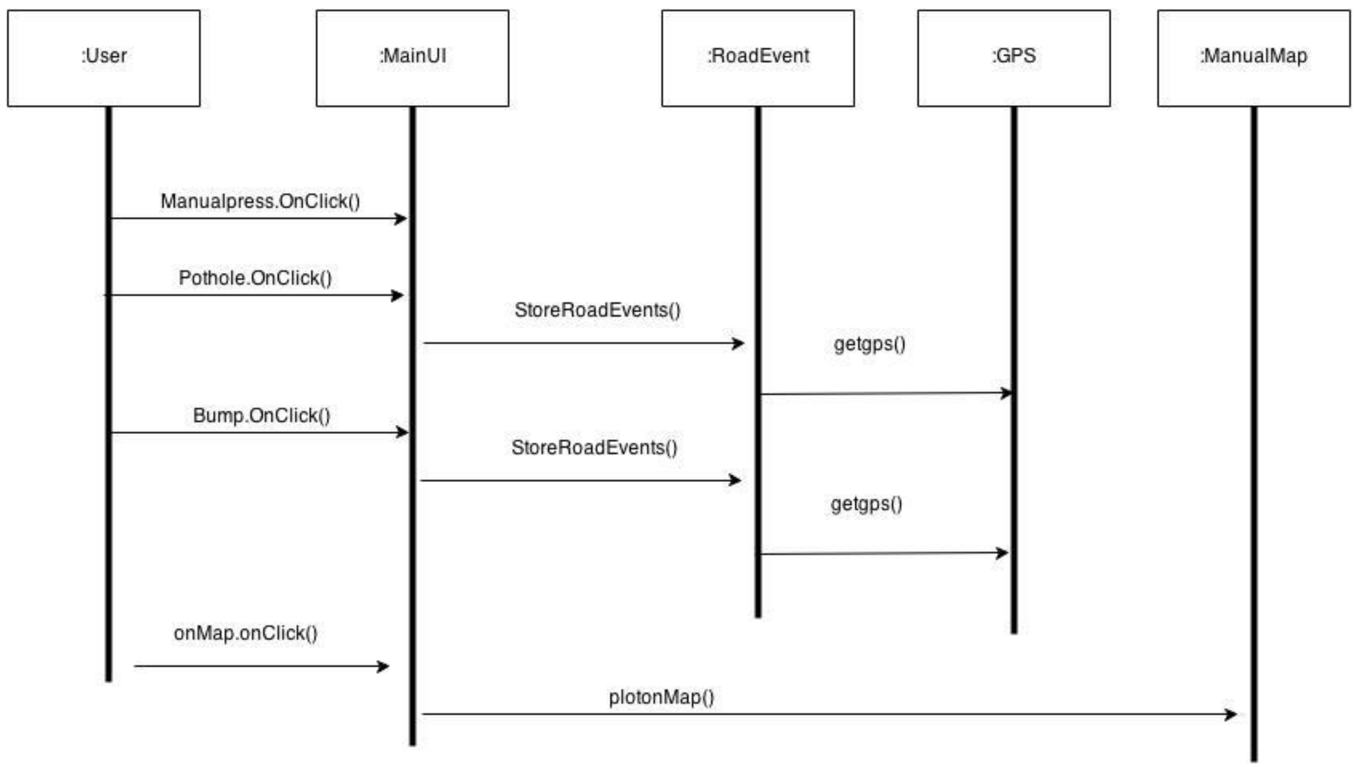

Figure 7: Sequence Model 3

Requirement 4 : User options to choose the type of "Anomaly"
Validation 4: Run Test 4
Test 4 : Sequence Model 4

1. Open app.
2. Press “Manual” Radio button.
3. Press ‘Start’ button.
4. Press ‘Stop’ button.
5. Press ‘Process’ button.
6. Press buttons ‘Smooth Road’, ‘Phone Drop’, ‘Pothole’, ‘Curb Hit’ or ‘Speed Bump’ as per the appropriate road hazard.

Result 4: All the data is recorded according to the event types.
Refer ‘Figure 6: Sequence Model 2.6’ for steps till stop and processing function

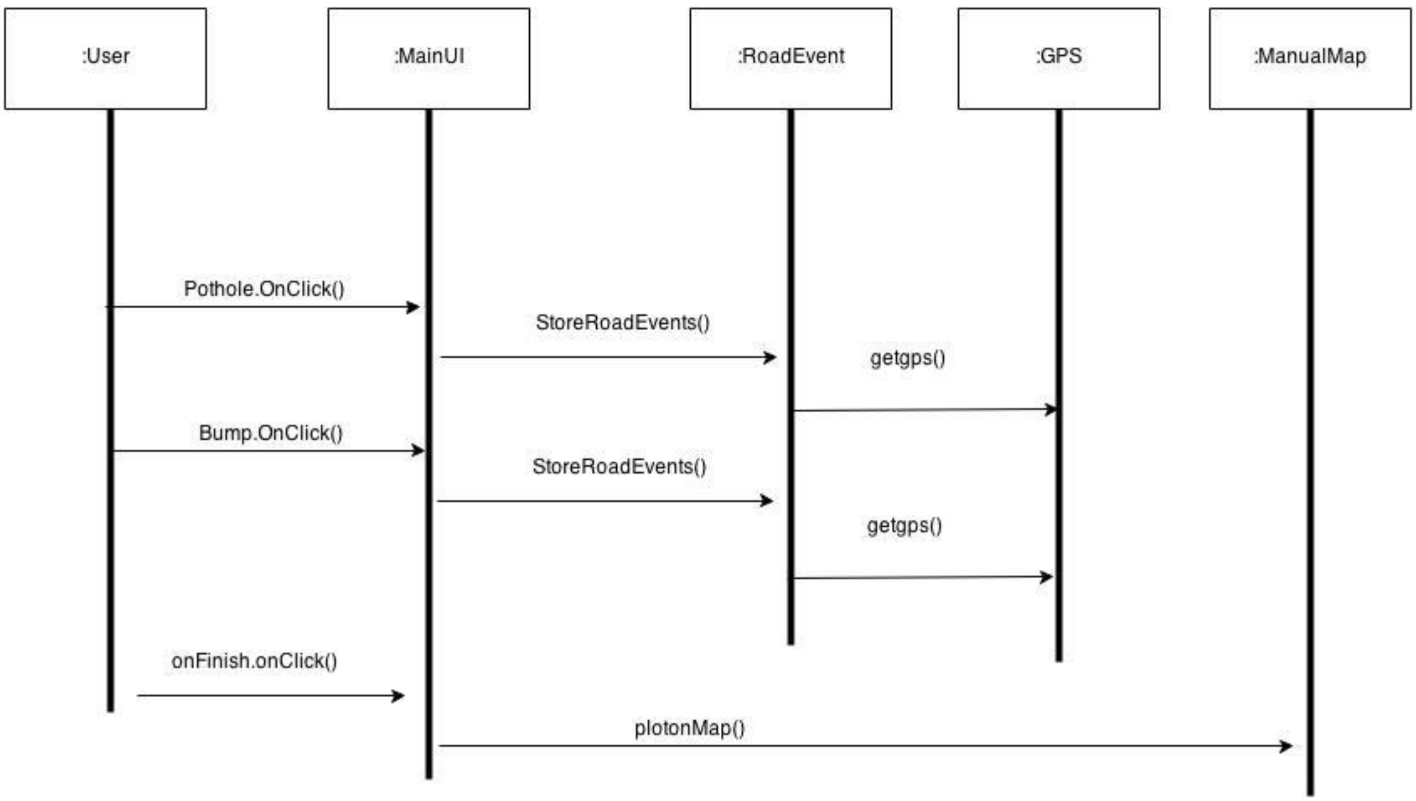

Figure 8: Sequence Model 4

Requirement 5: Vehicle is stationary and experience a jerk
Validation 5: Run Test 5
Test 5: Sequence Model 5

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. Press ‘Start’ button.
3. Make sure the car is stationary.
4. Perform scenarios which can make the accelerometer sense vibrations for e.g. Person Entering a car, Closing of Doors.
5. Press ‘Stop’ button.
6. Press ‘Process’ button.

Result 5: “No event detected” is displayed by the app.

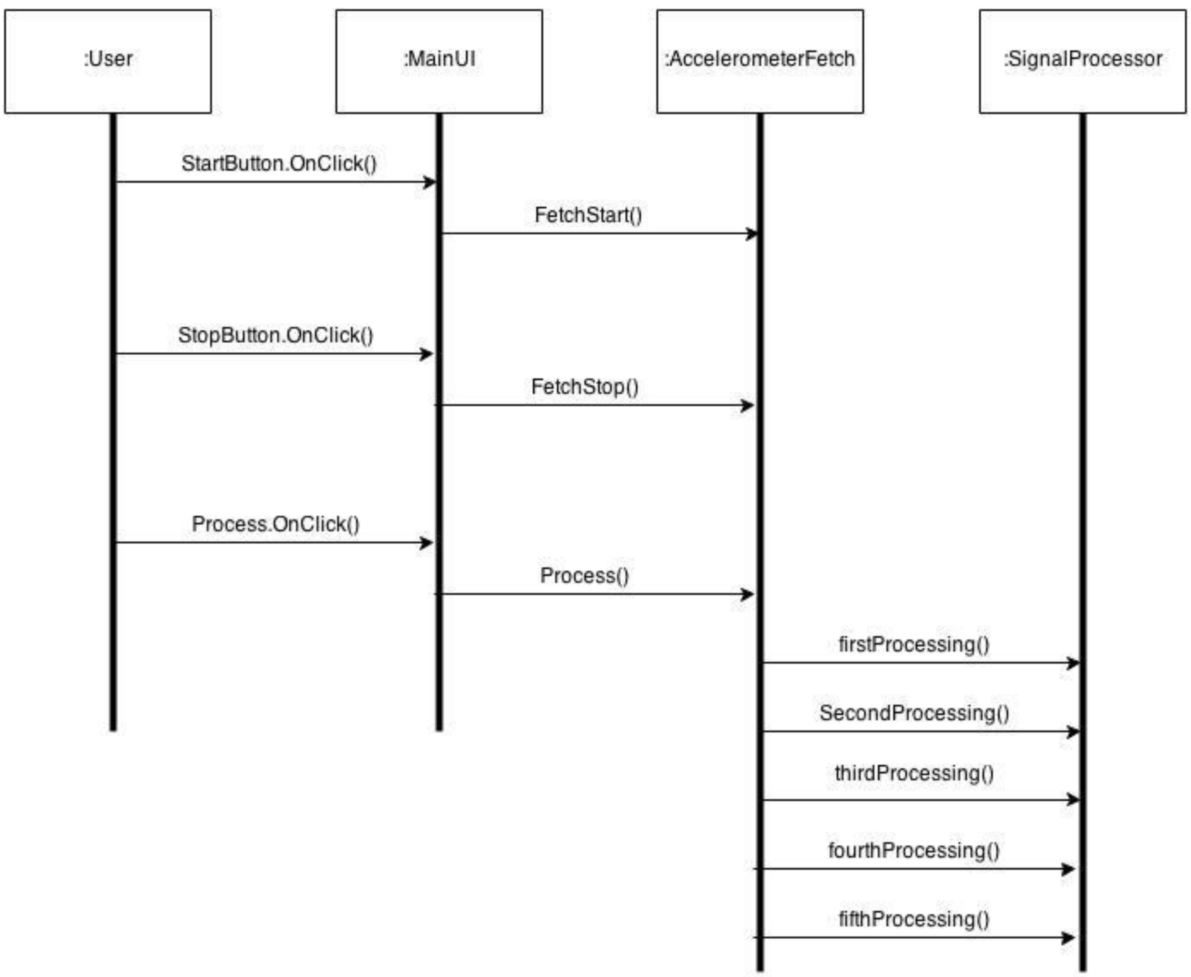

Figure 9: Sequence Model 5

Requirement A.1.1: When vehicle stops short due to sudden breaks
Validation A.1.1: Run Test A.1.1
Test A.1.1: Sequence Model 5.6

1. Place the device on the dashboard. Park the vehicle. Switch on the app.
2. Press ‘Start’ button.
3. Drive at the speed of 40 m/hr on smooth road
4. Apply sudden break.
5. Press ‘Stop’ button.
6. Press ‘Process’ button.

Result A.1.1: “No event detected” is displayed by the app.

Requirement A.1.2: user drops the phone while driving
Validation A.1.2 : Run Test A.1.2
Test A.1.2 : Sequence Model 5.6

1. Place the device on the dashboard. Park the vehicle. Switch on the app.
2. Press ‘Start’ button.
3. Drive at the speed of 40 m/hr. on smooth road
4. Drop the phone.
5. Press ‘Stop’ button.
6. Press ‘Process’ button.

Result A.1.2: “No event detected” is displayed by the app.

Requirement A.2: Is driven on long stretches of low quality road
Validation A.2: Run Test A.2
Test A.2: Sequence Model 5.6

1. Place the device on the dashboard. Park the vehicle. Switch on the app.
2. Press ‘Start’ button.
3. Drive at the speed of 40 m/hr. on low quality road
4. Press ‘Stop’ button.
5. Press ‘Process’ button.

Result A.2: App should mark the first pothole coordinate and to the end pothole coordinate.

Requirement A.3: differentiates pothole encountered based on their size as Small Medium Large
Validation A.3: Run Test A.3
Test A.3: Sequence Model 5.6

1. Place the device on the dashboard. Switch on the app.
2. Press ‘Start’ button.
3. Drive at the speed of 40 m/hr. through different size potholes
4. Press ‘Stop’ button.
5. Press ‘Process’ button.

Result A.3: App should mark the pothole according to the intensity High or Low.

Requirement A.4: During a call, the App goes on pause state and stops recording data.
Validation A.4: Run Test A.4
Test A.4:

1. Place the device on the dashboard. Switch on the app.
2. Press ‘Start’ button.
3. Call the phone on which the app is being run
4. Press ‘Stop’ button.
5. Press ‘Process’ button.

Result A.4: App should go on the pause state and stop recording data.

Requirement A.5: GPS data recorded are plotted on the map.
Validation A.5: Run Test 5
Test A.5: Sequence Model 5.8

1. Place the device on the dashboard. Switch on the app.
2. Press ‘ManualMap button.
3. Press buttons ‘Smooth Road’, ‘Phone Drop’, ‘Pothole’, ‘Curb Hit’ and ‘Speed Bump’ at interval of 10 meters.
4. Press show map button.

Result A.5: App should Plot all the road events recorded on the google maps.

Requirement A.6: Potholes detected are categorized based on current vehicle speed.
Validation A.6: Run Test A.6
Test A.6: Sequence Model 5.6

1. Place the device on the dashboard. Switch on the app.
2. Press ‘Start’ button.
3. Drive at the speed of 40 m/hr. through different size potholes
4. Select the low speed radio button.
5. Press ‘Stop’ button.
6. Press ‘Process’ button.

Result A.6: Data file generated by the app should contain the speed notation L representing low.

Requirement A.7.1: demo mode, events are injected using pre-recorded.
Validation A.7.1: Run Test A.7.1
Test A.7.1:

1. A data file consisting of the accelerometer value, speed and Gps data will be read by the demo app.

Result 7.1: The app should respond to the values from the data file and should plot potholes on the GPS coordinates mentioned by the data file.