

SMART PARKING SYSTEM

ECE59500 SMART DEVICES DESIGN & MODELING

TEAM MEMBERS

Raghavan Naresh Sarangapani

Ashley Dale

Susan Bose

Akhila Surneni

IUPUI

Date: 12/15/2016

CONTENTS

1	Abstract:	3
2	Introduction:	3
2.1	Assumptions:	4
3	Analysis:	5
3.1	SensorManager:	5
3.2	Sensor:	5
3.3	SensorEvent:	5
3.4	SensorEventListener:	5
3.5	Identifying sensors and sensor capabilities	5
3.6	Monitor sensor events	5
3.7	Kalman Filter	6
4	Sensors:	6
4.1	Accelerometer:	6
4.2	Gyroscope:	6
4.3	GPS:	6
5	Design:	7
6	Project Flow:	8
7	Results:	10
8	Challenges:	13
10	Future Work:	13
11	References:	14

1 Abstract:

Often a lot of people waste their time driving in circles in search for a parking space. Drivers around the world are turning to mobile apps to find parking spots. This idea of finding the parking spots in and around the user is the base of this project.

The idea of the project is to create an android app that could tell the user regarding parking spots in the school area and give the distance from the spot and the directions to the location. This is done by using GPS coordinates and time stamp to tell us when the user uses an application to get the appropriate parking spot. Since students and faculty are usually connected to the school Wi-Fi, using this information, we could track the location of the users if they are in the parking spot or not.

We store these GPS coordinates of different parking spots and the capacity in the application. If the GPS sample of the user says they have stayed at a spot for more than a certain amount of time (or average amount of time to park and get out of the car), the application logs a spot as taken. The application could also have the functionality for the user to confirm if the spot is taken already.

If the app sees the person who has parked has returned to logged GPS location and stays for a certain period, we assume that the person is leaving the parking spot and sends notification to the most nearby user that the parking spot is free. This application is smart and context aware since it has the functionality of understanding if a certain parking spot is free or not.

We plan to initially limit the process of searching the parking spots to only a handful of parking locations and see if the application works without any glitch and then expand it to an entire geographic location by improving the application accuracy. If the number of users increases, the app updates itself searching for more parking spots.

2 Introduction:

Twenty first century has everything one wants to make life easy and comfortable. All of this is possible due to drastic improvement in technology and resources. Rapid increase in the usage of phone and customizations of applications according to the user's needs and desires has led many companies to come with various ideas to develop applications that could help users save time and money.

One such successful applications includes "finding parking spots" for the users. These applications are often called smart due to their functionality of not only being able to find the nearest parking spots using the user's current location through the GPS receivers (Global Positioning System) but also provide other details such as the number of parking spots left, cost per hour etc.

- **Smart Parking System** is an integrated system to organize cars in public spaces. The motivation of this project is to help users (drivers) save their time. **Goal: Implement outdoor parking space tracking system**
- Current parking applications use two methods:
 - Indoor Systems utilize Wi-Fi /cellular network to track available parking spaces
 - Outdoor Systems utilize cameras, RFID tags, & other sensor devices
 - Both systems require complex hardware and are difficult to implement
- **Solution:** Build an android application that utilizes sensors already included in the phone to track parking positions

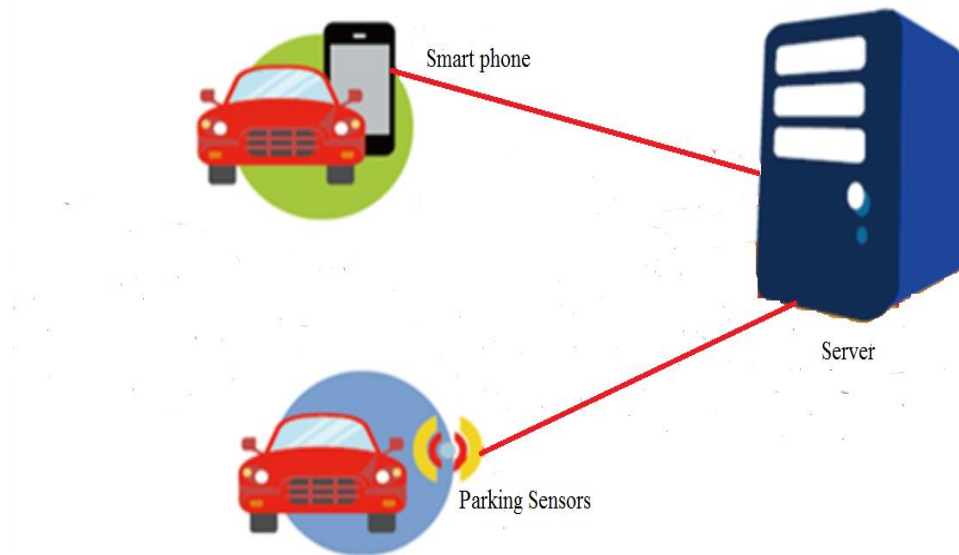


Figure 1. Exchange of Input and Output with the server

2.1 Assumptions:

- Phone is inside car, calibrated to a fixed gyroscope orientation that matches the orientation of car.
- The user presses start button to start monitoring movement (HCI)
- Test parking environment consists of 4 parking spots distant from each other
- Parking areas are pre-determined, and stored in a database in the server. When a user enters an area, the database uses context awareness to determine which parking spots are available to that user.

3 Analysis:

Most Android-powered devices have built-in sensors that measure motion, orientation, and various environmental conditions. The Android platform supports three broad categories of sensors: motion sensors, environment sensors, position sensors. The Android platform provides several sensors that let you monitor the motion of a device. Two of these sensors are always hardware-based (the accelerometer and gyroscope), and three of these sensors can be either hardware-based or software-based (the gravity, linear acceleration, and rotation vector sensors).

The sensor framework is part of the android.hardware package and includes the following classes and interfaces:

3.1 SensorManager:

This class is used to create an instance of the sensor service. This class provides various methods for accessing and listing sensors, registering and unregistering sensor event listeners, and acquiring orientation information. This class also provides several sensor constants that are used to report sensor accuracy, set data acquisition rates, and calibrate sensors.

3.2 Sensor:

This class can be used to create an instance of a specific sensor. It also provides various methods to determine a sensor's capabilities.

3.3 SensorEvent:

The system uses this class to create a sensor event object, which provides information about a sensor event. A sensor event object includes the following information: the raw sensor data, the type of sensor that generated the event, the accuracy of the data, and the timestamp for the event.

3.4 SensorEventListener:

This interface is used to create two callback methods that receive notifications (sensor events) when sensor values change or when sensor accuracy changes.

In a typical application, we use these sensor-related APIs to perform two basic tasks:

3.5 Identifying sensors and sensor capabilities

Identifying sensors and sensor capabilities at runtime is useful if the application features that rely on specific sensor types or capabilities.

3.6 Monitor sensor events

Monitoring sensor events is how you acquire raw sensor data. A sensor event occurs every time a sensor detects a change in the parameters it is measuring. A sensor event provides you with four pieces of information: the name of the sensor that triggered the event, the timestamp for the event, the accuracy of the event, and the raw sensor data that triggered the event.

Android provides a flexible framework for UI design that allows your app to display different layouts for different devices, create custom UI widgets, and even control aspects of the system UI outside your app's window.

3.7 Kalman Filter

- The Kalman filter is used to remove the noise in the accelerometer values and make them uniform.
- Simple form of Kalman filter is used
 - $\text{New State} = A * \text{Alpha Values} * X + C$
 - New State is the values to be predicted.
 - Alpha value is multiplication factor and varies from each level..
 - X is the predicted values.
 - C is co variance matrix.

4 Sensors:

4.1 Accelerometer:

Accelerometer is a sensor that is present in all the phones these days and it measures the acceleration of the device. A mobile generally contains a 3 axis accelerometer and they measure the motion by sensing how much a mass presses in something when a force acts in it.

At rest, the value of the accelerometer is approximately 1 G upwards because any point on the earth's surface is accelerating upwards relative to the local inertial frame.

4.2 Gyroscope:

Gyroscope is a sensor that measures the orientation based on the principles of conservation of angular momentum. A conventional gyroscope measure orientation where as other sensors can measure angular rate and are called as rate-gyros.

Accelerometer and magnetometer measure acceleration and angle relative to the earth, gyroscope measures angular velocity to the body.

4.3 GPS:

GPS is a sensor that measures the exact location of a device on the surface of the earth. This is done by beaming signals from multiple satellites, positioned such a way that at least 3 are focusing on a single device for triangulation. The android sensor gives out the latitude, longitude and altitude values from the sensor. The difficult part is getting the

signal inside closed buildings as it works only in line of sight. Hence there is the need for indoor localization using other sensors.

5 Design:

- Gyroscope and accelerometer sensors present in our phone are used to calculate the velocity and the orientation of the car.
- The values from these sensors are taken to calculate the motion detection, linear acceleration, the amount of angle moved, distance increment
- These values are evaluated to find out the final distance and the parking spot occupied (P1/P2/P3/P4).

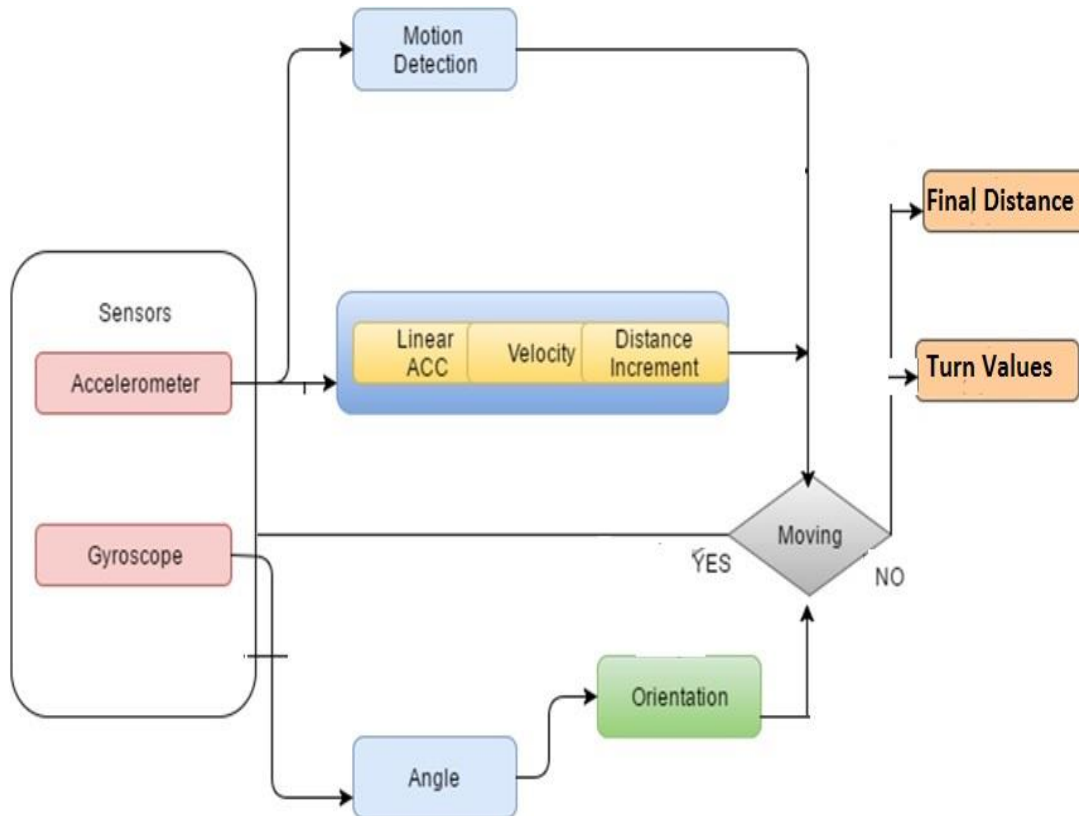


Figure 2. Sensor usage flow

6 Project Flow:

- The android app is used to get values from the Accelerometer, Gyroscope and GPS, store the values in a file in the phone, and finally upload the file to the FTP server.
- This file is then downloaded by the PC based JAVA application and processed to identify the parking spot in which the car was parked.
- The mobile setup in the car is fixed such that the Z axis is facing the front side of the car.
- In the mobile app, when the user presses the start button, we start sampling the accelerometer and gyroscope data at the fastest rate using `SENSOR_DELAY_GAME` delay.
- This data is added with a timestamp of the exact time and the time in milli seconds, and the letters A for accelerometer, G for GPS and Y for gyroscope are added as a field and then written to a CSV file in the phone.
- This is continued until the user presses the stop button.
- When the user presses STOP button, it is assumed that this is the exact parking spot. So the application tries to get the GPS signal for about 10 seconds. This is also stored in the file to mark the GPS of the parking spot.
- Next the file is uploaded using the FTP protocol.
- Now the system Java app can be run, this will download the file from the FTP location and process the data.
- Once the file is downloaded the values are read as Accelerometer and Gyroscope pairs from the downloaded CSV file.
- These values are then added to a Kalman filter, implemented using the `simplematrix` and `equations` functions of the Efficient Java Matrix Library.
- The Kalman filter takes in the values of 3 axes of the accelerometer and 3 axes of the gyroscope, processes them by adding the previous state to a continuously generated value and predicts the future state.
- This is done to smooth out the erroneous output of the accelerometer and gyroscope.
- After this the axes of the accelerometer are subtracted from the previous values and this is stored in another simple matrix, which gets updated with each new set of values.
- The concept behind this is to divide the entire trip into discrete events in which in a specific axis if the Car travelled at $X \text{ m/s}^2$ of acceleration then the velocity can be obtained by the first integration and the distance can be obtained by the second integration.
- Integration in this context is just multiplying the difference in acceleration across an axes with the square of the time in milli seconds, to get the distance in that interval.
- This is added up continuously until the end of the file is reached, giving the total distance across all the axes.

- As we have fixed the direction of movement as Z axis, we try to analyze the gyroscope along that axes and it was seen that when making a right turn the gyroscope value went below -2.5 and for left turn it was greater than 2.5 for at least 20 values.
- So, we fixed to detect these extremes beyond the 2.5 and -2.5 limit to detect the type of the turn, also during this process the straight distance up to the turn is stored separately from the Z axis distance.
- Now the type of turn has been predicted so if the turn is right then parking spot should be R1 or R2, vice versa for left turn L1 or L2.
- To predict the exact parking spot, we map the environment as in this example keep 7 meters as a decision point before which the parking spot is 1 and beyond that parking spot is 2.
- Then, the final GPS value is also stored.
- Now using all these inputs the exact parking spot is decided and the corresponding GPS co-ordinates for the spot is shown.
- The current implementation of the PC app uses a console window to show the output, this can later be programmed to implement graphically the parking spots and to suggest open parking spots.
- Also in the future, instead of user pressing start and stop buttons, the app can detect the GPS co-ordinates when reaching the parking garage and start automatically and stop when there is less movement.
- The distance calculation can also be fine-tuned for further error reduction and the criteria can also be increased to accommodate more parking spots.
- We have tested the system in a closed room with phone connected to wifi and by walking in four types.
 - Going straight for long distance and turning right.
 - Going straight for short distance and turning right.
 - Going straight for long distance and turning left.
 - Going straight for short distance and turning left.
- The program was able to predict the exact parking spots out of the four stops.
- To test this, we have saved the files separately and to check this in the java application please comment the lines 113 and 115 which will skip downloading the file from ftp, and set the corresponding file name to *FileToDownload* parameter in line 75. This will open the saved file and show the output.
- To change back to implementation of file download , uncomment the lines 113 and 115, and change the *FileToDownload* parameter in line 75 to "Saved_Sensor_Values.csv".

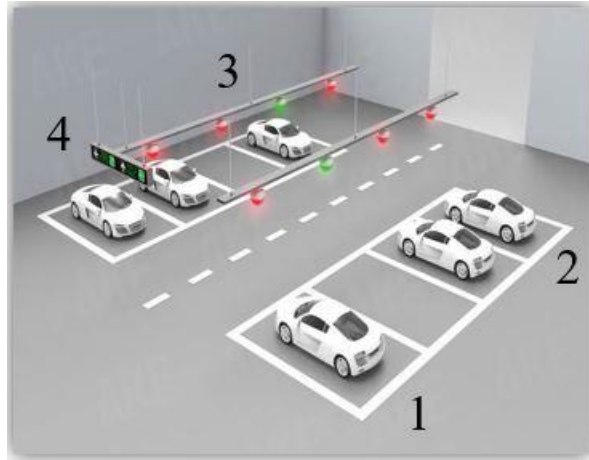


Figure 3. Parking lot with 4 parking spots P1,P2,P3,P4

7 Results:

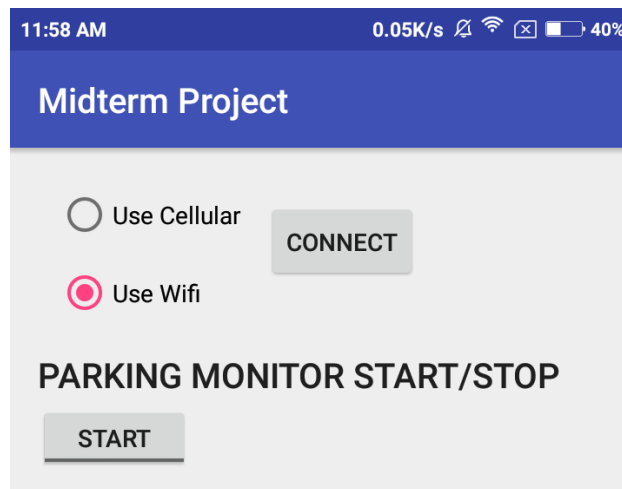
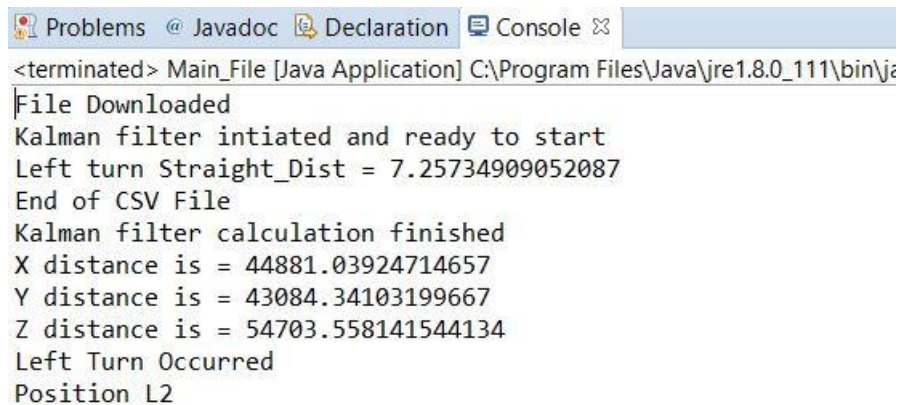


Figure4. User Interface of the Smart Parking System



```
<terminated> Main_File [Java Application] C:\Program Files\Java\jre1.8.0_111\bin\java.exe
File Downloaded
Kalman filter initiated and ready to start
Left turn Straight_Dist = 3.104053001612709
Kalman filter calculation finished
X distance is = 7.222243631039055
Y distance is = 4.746323986622124
Z distance is = 10.384619345611238
Left Turn Occurred
Position L1
```

Figure5. Result after the vehicle starts and turns left to occupy position L1



```
<terminated> Main_File [Java Application] C:\Program Files\Java\jre1.8.0_111\bin\java.exe
File Downloaded
Kalman filter initiated and ready to start
Left turn Straight_Dist = 7.25734909052087
End of CSV File
Kalman filter calculation finished
X distance is = 44881.03924714657
Y distance is = 43084.34103199667
Z distance is = 54703.558141544134
Left Turn Occurred
Position L2
```

Figure6. Result after the vehicle starts and turns left to occupy position L2



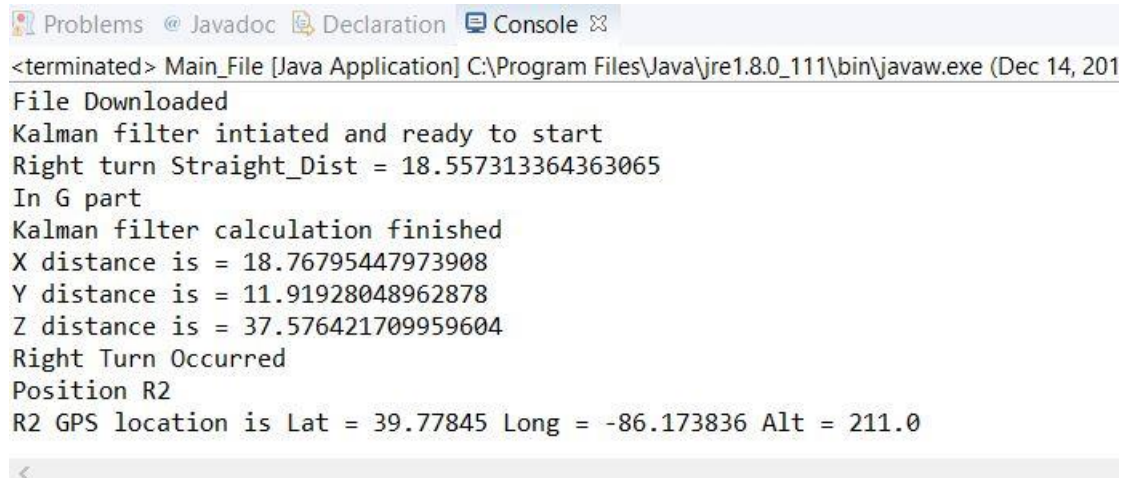
```
<terminated> Main_File [Java Application] C:\Program Files\Java\jre1.8.0_111\bin\ja
File Downloaded
Kalman filter initiated and ready to start
Right turn Straight_Dist = 1.6851350291680272
End of CSV File
Kalman filter calculation finished
X distance is = 3.8991362496544095
Y distance is = 2.9994941845334933
Z distance is = 7.449769095880736
Right Turn Occurred
Position R1
```

Figure7. Result after the vehicle starts and turns Right to occupy position R1



```
<terminated> Main_File [Java Application] C:\Program Files\Java\jre1.8.0_111\bin\ja
File Downloaded
Kalman filter initiated and ready to start
Right turn Straight_Dist = 5.289521210134728
End of CSV File
Kalman filter calculation finished
X distance is = 7.404727510371984
Y distance is = 6.554395734625203
Z distance is = 16.60326710599388
Right Turn Occurred
Position R2
```

Figure8. Result after the vehicle starts and turns left to occupy position R2



```
Problems @ Javadoc Declaration Console
<terminated> Main_File [Java Application] C:\Program Files\Java\jre1.8.0_111\bin\javaw.exe (Dec 14, 201
File Downloaded
Kalman filter initiated and ready to start
Right turn Straight_Dist = 18.557313364363065
In G part
Kalman filter calculation finished
X distance is = 18.76795447973908
Y distance is = 11.91928048962878
Z distance is = 37.576421709959604
Right Turn Occurred
Position R2
R2 GPS location is Lat = 39.77845 Long = -86.173836 Alt = 211.0
```

Figure9. Coordinates of the GPS location at position R2

8 Challenges:

- Implementing the covariance matrix with the initial values
- Calibration issues
- Data storage
- Algorithmic speed Vs actual parking

10 Future Work:

- Improve accuracy for better detection of the adjacent parking spots.
- Include Indoor parking along with the existing system.
- Automatic start of the application without user having to press “Start” and suggest parking spots nearby by making use of current location of the user.
- Improve the functionality of the application.
- Database of the parking system to know the placement of the parking lots.
- Improve Kalman filter properties to increase accuracy of prediction

11 References:

- Vincent, David. "Accurate Position Tracking Using Inertial Measurement Units", PNI Whitepaper (2013).
- Scarlett, Jim. "Enhancing the Performance of Pedometers Using a Single Accelerometer", Analog Devices, AN900.
- Shala, Ubejd. Rodriguez, Angel. "Indoor Positioning using Sensor-fusion in Android Devices", School of Health and Society (2013)
- Foxlin, Eric. "Pedestrian Tracking with Shoe-Mounted Inertial Sensors", Intersense (2005)
- Paul.S, Anindhya, Wan.A, Eric, "RSSI-Based Indoor Localization and Tracking Using Sigma-Point Kalman Smoothers", IEEE Selected topics in Signal Processing (2009)
- "Example Kalman Filter - Efficient Java Matrix Library", Ejml.org, 2016. [Online]. Available: http://ejml.org/wiki/index.php?title=Example_Kalman_Filter. [Accessed: 08-Dec- 2016].
- "Filtering Sensor Data with a Kalman Filter", Interactive Matter Lab, 2009. [Online]. Available: <http://interactive-matter.eu/blog/2009/12/18/filtering-sensor-data-with-a-kalman-filter/>. [Accessed: 08- Dec- 2016].
- "Java Code Example", Programcreek.com, 2016. [Online]. Available: http://www.programcreek.com/java-api-examples/index.php?source_dir=Rubik-Cube-Wizard-master/Rubik%20Solver/src/org/ar/rubik/CubePose.java. [Accessed: 08- Dec- 2016].
- D. Kohanbash, "Filtering Sensor Data with a Kalman Filter", Interactive Matter Lab, 2014. [Online]. Available: <http://interactive-matter.eu/blog/2009/12/18/filtering-sensor-data-with-a-kalman-filter/>. [Accessed: 08- Dec- 2016].
- "How to calculate distance based on phone acceleration," 2016. [Online]. Available:<http://stackoverflow.com/questions/4329164/how-to-calculate-distance-based-on-phone-acceleration>. Accessed: Dec. 15, 2016
- L. Gavin, "Android - how to track steps and calculate running distance,". [Online]. Available: <http://www.lewisgavin.co.uk/Step-Tracker-Android/>. Accessed: Dec. 15, 2016.
- [Online]. Available: http://4.http://www.programcreek.com/java-api-examples/index.php?source_dir=Rubik-Cube-Wizard-master/Rubik%20Solver/src/org/ar/rubik/CubePoseEstimator.java. Accessed: Dec. 15, 2016.
- "Java object and classes," www.tutorialspoint.com, 2016. [Online]. Available: https://www.tutorialspoint.com/java/java_object_classes.htm. Accessed: Dec. 15, 2016.
- mkyong, "How to read and parse CSV file in java," 2013. [Online]. Available: <https://www.mkyong.com/java/how-to-read-and-parse-csv-file-in-java/>. Accessed: Dec. 15, 2016.
- "How to remove gravity factor from accelerometer readings in Android 3-axis accelerometer," 2016. [Online]. Available:

- <http://stackoverflow.com/questions/3377288/how-to-remove-gravity-factor-from-accelerometer-readings-in-android-3-axis-accel>. Accessed: Dec. 15, 2016.
- "Java abs() method," [www.tutorialspoint.com](http://www.tutorialspoint.com/java/number_abs.htm), 2016. [Online]. Available: https://www.tutorialspoint.com/java/number_abs.htm. Accessed: Dec. 15, 2016.
 - "Example Kalman filter - efficient java matrix library,". [Online]. Available: http://ejml.org/wiki/index.php?title=Example_Kalman_Filter#SimpleMatrix_Example. Accessed: Dec. 15, 2016.
 - [Online]. Available: <http://www.sauronsoftware.it/projects/ftp4j/manual.php#14>. Accessed: Dec. 15, 2016.