Programmer's Guide to Directional Step Counter

Application Interface and functionality

The directional step counter (DSC) application is built in Android Studio using Java to integrate different application interfaces (API) for Android Operating System (OS). The main functionality will be creating a compass, a directional step counter, and a magnetometer quality detector. These APIs ensure that the application can run properly as intended. The primary APIs that are worked on are:

1. Android Sensors API
2. Android Permissions API
3. Android Animations API
4. Android User-Interface (UI) API
5. Android Context API

The Android sensors API includes five sensors: an accelerometer, gyroscope, magnetic field, step counter, and step detector. They are designated as *TYPE_ACCELEROMETER*, *TYPE_GYROSCOPE*, *TYPE_MAGNETIC_FIELD*, *TYPE_STEP_COUNTER*, and *TYPE_STEP_DETECTOR*, respectively, according to Android documentation [1]. The Android permissions API requests permission to access the sensors and high sampling rates, with user prompts handled in the AndroidManifest.xml file. The Android animations and UI APIs are implemented in both Java classes and layout XML files to create user-friendly UIs, including rotating animations, TextView, ImageView, Button, ImageButton, ListView, and various layouts.

Classes, Methods, Sensors, and Listeners

The **MainActivity** class serves as the primary component of the application, where sensors are managed, and relevant calculations are performed to output data in the user interface. To leverage the device's sensors, the class implements the *SensorEventListeners* interface. The sensors used in the application are created as objects from the Sensor class, including *aSensor*, *mSensor*, *gyroSensor*, *stepCSensor*, and *stepDSensor*. These sensors are then given the designated sensor from the *SensorManager* class, declared as *sm*.

The **onCreate()** method is crucial in setting up the layout and starting the application's lifecycle. Permission checks for body sensors and physical activity are executed using the *checkSelfPermission*() method from the *ContextCompat* class. These checks are done inside the *stepSensorChecks*() function to ensure modularity, called at the start of the *onCreate*() method after the setContentView() method. If permissions are granted, the availability of sensors is confirmed using the *getDefaultSensor*() method. The sensors are then connected to the *SensorManager* within the *initialization*() function using the *registerListener*() method.

Layout identifications (IDs) are assigned in the *initialization*() function using the *findViewById*() function, which is necessary for displaying the user interface. Overall, the *MainActivity* class is structured to provide high modularity by separating tasks into functions, minimising redundancies, and optimising performance. As the sensors' value changes, the **onSensorChange()** function will be run in the application. This function contains the following:

1. switch-case statement of using the different sensors
2. *directionalSteps() function*
3. *magnetometerQuality()* function
4. *calculateOrientation()* function

The switch-case statements are used to detect the type of sensors using *getType()* method, which then runs the intended codes when the sensors are detected. The other functions will be explained more in the data processing section. The **onAccuracyChanged()** method will also be needed to receive updates from the device's sensors. The **onResume()** method is also used to register the sensors using *registerListener()* method whenever the application is resumed, while the **onPause()** and **onDestroy()** methods will do the opposite using *unregisterListener()*.

Data processing

The data processing mainly occurs inside the *onSensorChanged()* method, which is driven by the *calculateOrientation()*, *directionalSteps(),* and *magnetometerQuality()* functions. The **calculateOrientation()** function enables the data from the accelerometer and magnetometer to be used as the inputs in getting the compass orientation angle and direction. This is done by utilising the *getRotationMatrix()* and *getOrientation()* methods which contain the sensor values that have been filtered using a low-pass filter. The low-pass filter is created in the *lowPF()* function, in which the coefficient, *alpha*, has been calibrated after multiple testing to have an accurate compass rotation [2]. This resulted in using quite a low alpha value of 0.08 for achieving high accuracy. This function is called at the end of the *onSensorChanged()* to process the compass's angle continuously.

The **directionalSteps()** function is created to input the change in steps from the step counter sensor and the direction the compass is facing. The step detector sensor was not used since, after testing, it was not as accurate as the step counter and is more error-prone. This function is called whenever the step counter is detected inside the switch-case statement. The changes are then incremented to an int array, *stepCounterCount,* which is used to save the number of steps in different directions. Lastly, the **magnetometerQuality()** function will assess the magnetometer quality by comparing the bearings calculated using the filtered data from the magnetometer and gyroscope. The bearings are calculated using the Pythagoras theorem formula on the specified axis of the sensors. The average bearing over 3 seconds is used to calculate the percentage accuracy of the magnetometer then and displayed in "GOOD", "OKAY", or "BAD", which have black, yellow, and red colours, respectively.

The data are then inputted into the respective *View* objects, which will be outputted on the activity_main.xml file. For compass rotating animation, an *ImageView* object is used with *RotateAnimation* class with a *setDuration()* of 1000, which is one second to have a smooth animation. This is done inside the *rotatingCompass()* function at the bottom of the *calculateOrientation()* function.

Extra features

Two extra features were done for the application: the reset button and the total step button. The reset button uses the *resetButtonOnClick()* function that resets the array elements to zero, so the directional steps will then be seen as zero. The total step button uses the *Intent* class to transfer the *stepCounterCount* array from MainActivity class to the PopUp class, which is then outputted as a pop-up window using a ListView on another layout.

References

[1] "Sensor: android developers," *Android Developers*. [Online]. Available: https://developer.android.com/reference/android/hardware/Sensor[Accessed: 11-Feb-2023].

[2] "Motion sensors; android developers," Android Developers. [Online]. Available: https://developer.android.com/guide/topics/sensors/sensors_motion. [Accessed: 16-Feb-2023].