ECE 558: Final Project Report
“Accelerometer-Based Color via The Cloud”

December 7th, 2018

Aakanksha Mathuria
Alec Wiese

Introduction:

The goal of this project is to convert the mobile device orientation into a color on the 360-degree color wheel. The color is being displayed on the app’s screen and also on the LED connected to an Android Things device. To change the LED color, we upload the RGB values to the google firebase. We are also controlling the motor from the mobile device. If the user shakes the device, it will toggle the motor ON and OFF.

We use the accelerometer data sensor readings to determine the orientation of the device. To compute the readings, we use the android sensor manager. https://developer.android.com/guide/topics/sensors/sensors_overview

To determine the color, we calculate the HSV (Hue, Saturation, and Value) color space using the trigonometry calculation from the device orientation and then map it to RGB (Red, Green, and Blue) color space.

After computing the RGB values, we change the app’s screen color and send these values to the firebase to change the LED color.

Functionalities:

- Rotating the device clockwise or counterclockwise (the “roll”) changes the Hue (color).
- The saturation slider on the mobile app manually changes the Saturation (fullness of color).
- Tilting the device forward or backward determines the Value (brightness).
- If the mobile device is shaken, it will toggle the motor ON and OFF.

Interaction of Mobile App and Android Things App:

We send the RGB and PWM motor values to the android things via google firebase. RGB values are used for changing the LED color and PWM motor value is for changing the motor speed value.

Raspberry Pi 3

Image Sources:
https://www.target.com https://openclipart.org/detail/213897/black-android-mobile-phone
https://www.google.com

Contributions:

- Mobile App - Aakanksha (set up the layout, accelerometer readings), Alec (implemented algorithms to get the HSV and RGB values)
- Android Things App - Alec

Mobile application:

Find the app here: .\MobileApp\SensorManager
Find the Main activity here: .\MobileApp\SensorManager\app\src\main\java \com\example\aakan\sensormanager

The goal of this application is to determine the orientation of the user's Android device, respond to device shake, and update Google Firebase according to those inputs. The app includes algorithms for mapping the device's orientation to a color in the HSV color space and converting that to the RGB color space to update the RGB LED. The user can change the color of the Mobile app background and the RGB LED connected to the Android Things device by setting the hue, saturation, and value. The hue is set by rotating the device on the X-Y plane (the "roll" axis) and getting the angle of rotation of the device which is the hue:

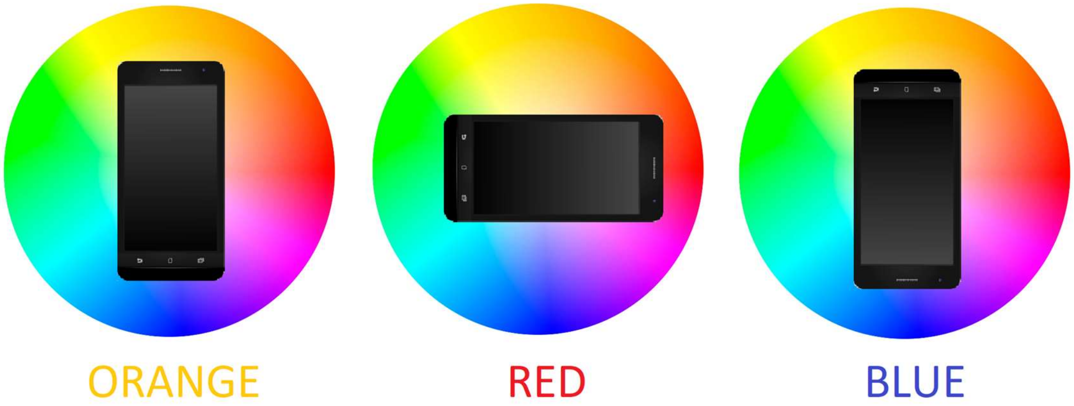
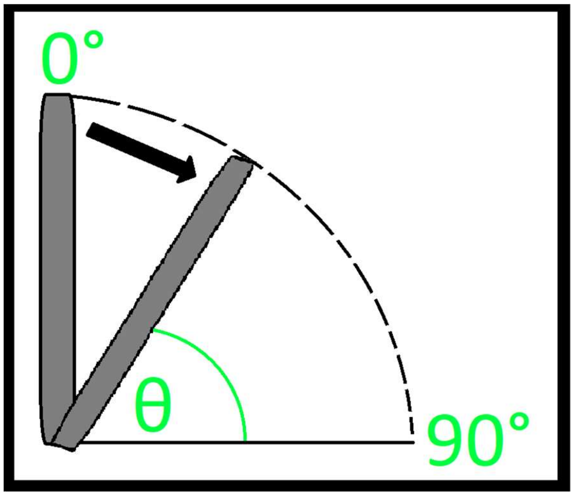


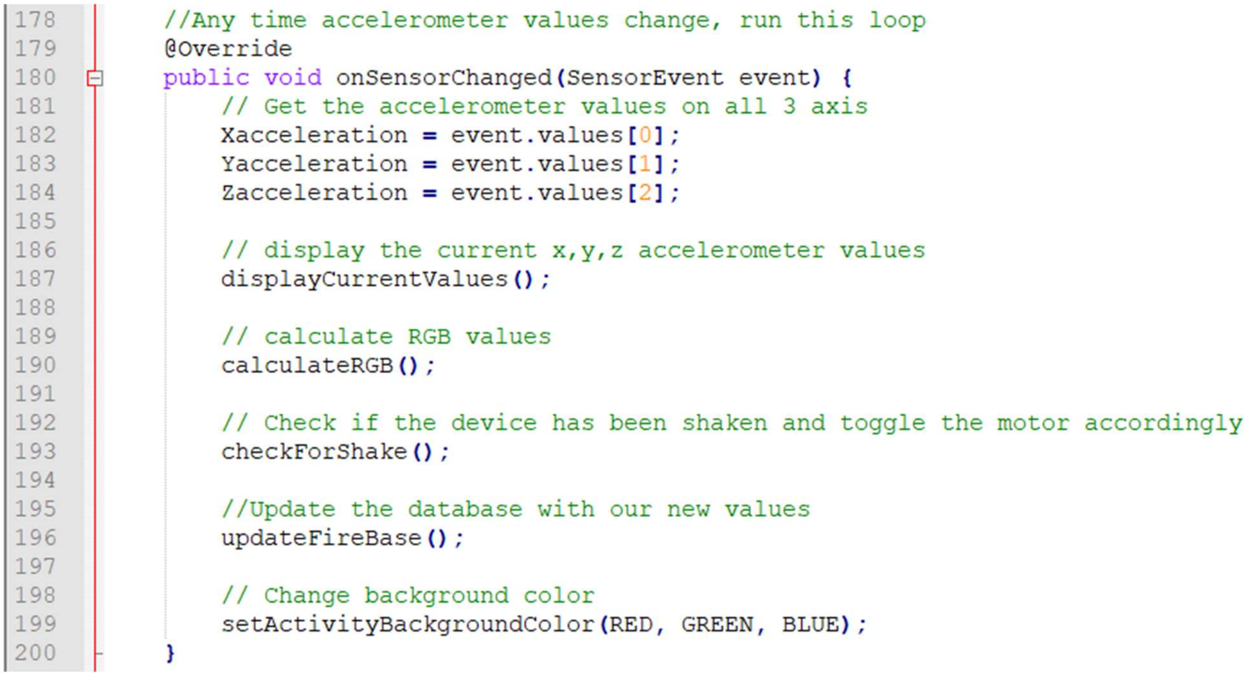
Image Sources:
https://color.adobe.com/create/color-wheel/
https://openclipart.org/detail/213897/black-android-mobile-phone

The saturation is set by a seekbar slider on the activity. The value is determined by the backwards or forwards tilt of the device on the Z-axis by determining the angle between 0 and 90 degrees:



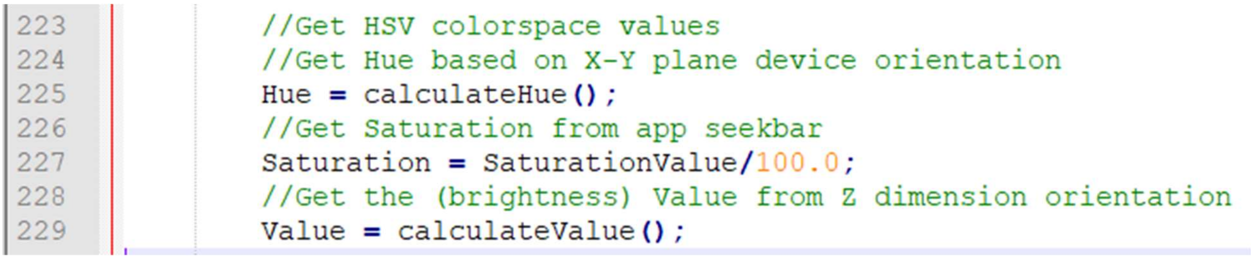
The Motor PWM speed can be set using a seekbar slider, and the motor can be toggled on and off by shaking the device.

We accomplished these functionalities by overriding the Android “onSensorChanged” listener to run through these calculations each time the orientation of the device is changed, this acted as the main loop for our activity. This method updates the app view, calculates the RGB values based on device orientation, checks if the device is being shaken, updates the Google Firebase database, and changes the background of the activity:



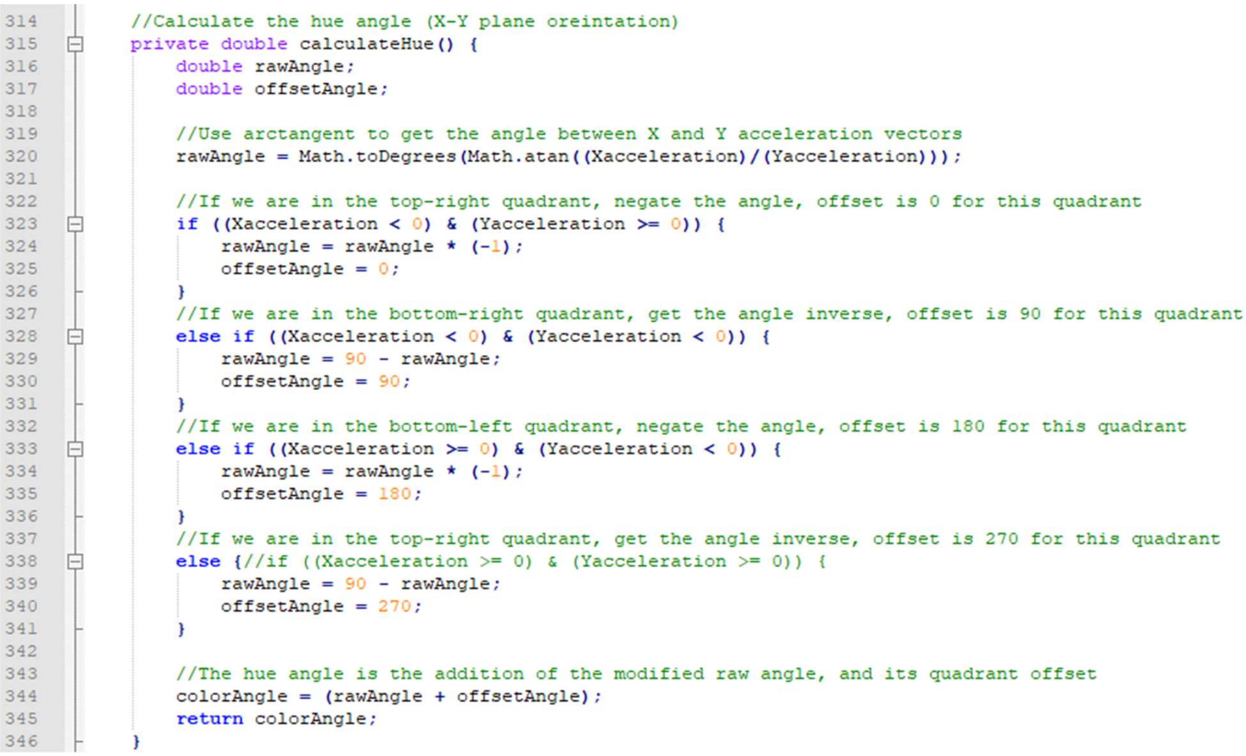
```
//Any time accelerometer values change, run this loop
@Override
public void onSensorChanged(SensorEvent event) {
    // Get the accelerometer values on all 3 axis
    Xacceleration = event.values[0];
    Yacceleration = event.values[1];
    Zacceleration = event.values[2];

    // display the current x,y,z accelerometer values
    displayCurrentValues();

    // calculate RGB values
    calculateRGB();

    // Check if the device has been shaken and toggle the motor accordingly
    checkForShake();

    //Update the database with our new values
    updateFireBase();

    // Change background color
    setActivityBackgroundColor(RED, GREEN, BLUE);
}
```

The calculateRGB method calls functions to calculate the hue and value but gets acceleration from the seekbar slider.



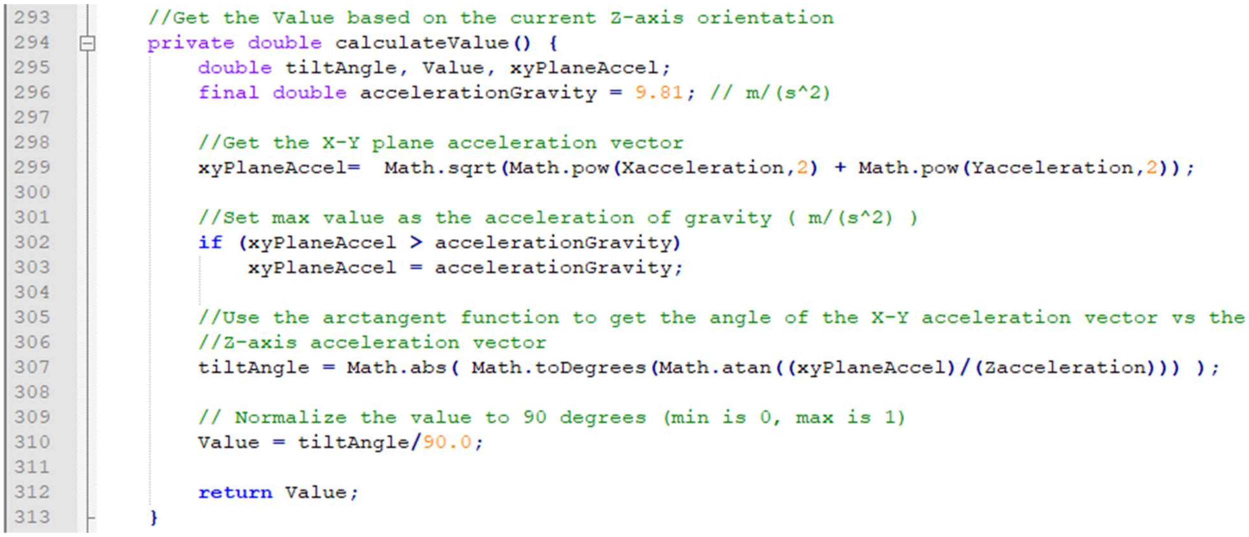
```
        //Get HSV colorspace values
        //Get Hue based on X-Y plane device orientation
        Hue = calculateHue();
        //Get Saturation from app seekbar
        Saturation = SaturationValue/100.0;
        //Get the (brightness) Value from Z dimension orientation
        Value = calculateValue();
```

In the calculateHue method the class variables for acceleration are used to determine the angle between the X and Y accelerometer values. This angle, and knowing which quadrant of the X-Y plane that the device is oriented in, yields the hue angle which is passed back to the calculateRGB function:



```
//Calculate the hue angle (X-Y plane oreintation)
private double calculateHue() {
    double rawAngle;
    double offsetAngle;

    //Use arctangent to get the angle between X and Y acceleration vectors
    rawAngle = Math.toDegrees(Math.atan((Xacceleration)/(Yacceleration)));

    //If we are in the top-right quadrant, negate the angle, offset is 0 for this quadrant
    if ((Xacceleration < 0) & (Yacceleration >= 0)) {
        rawAngle = rawAngle * (-1);
        offsetAngle = 0;
    }
    //If we are in the bottom-right quadrant, get the angle inverse, offset is 90 for this quadrant
    else if ((Xacceleration < 0) & (Yacceleration < 0)) {
        rawAngle = 90 - rawAngle;
        offsetAngle = 90;
    }
    //If we are in the bottom-left quadrant, negate the angle, offset is 180 for this quadrant
    else if ((Xacceleration >= 0) & (Yacceleration < 0)) {
        rawAngle = rawAngle * (-1);
        offsetAngle = 180;
    }
    //If we are in the top-right quadrant, get the angle inverse, offset is 270 for this quadrant
    else {//if ((Xacceleration >= 0) & (Yacceleration >= 0)) {
        rawAngle = 90 - rawAngle;
        offsetAngle = 270;
    }

    //The hue angle is the addition of the modified raw angle, and its quadrant offset
    colorAngle = (rawAngle + offsetAngle);
    return colorAngle;
}
```

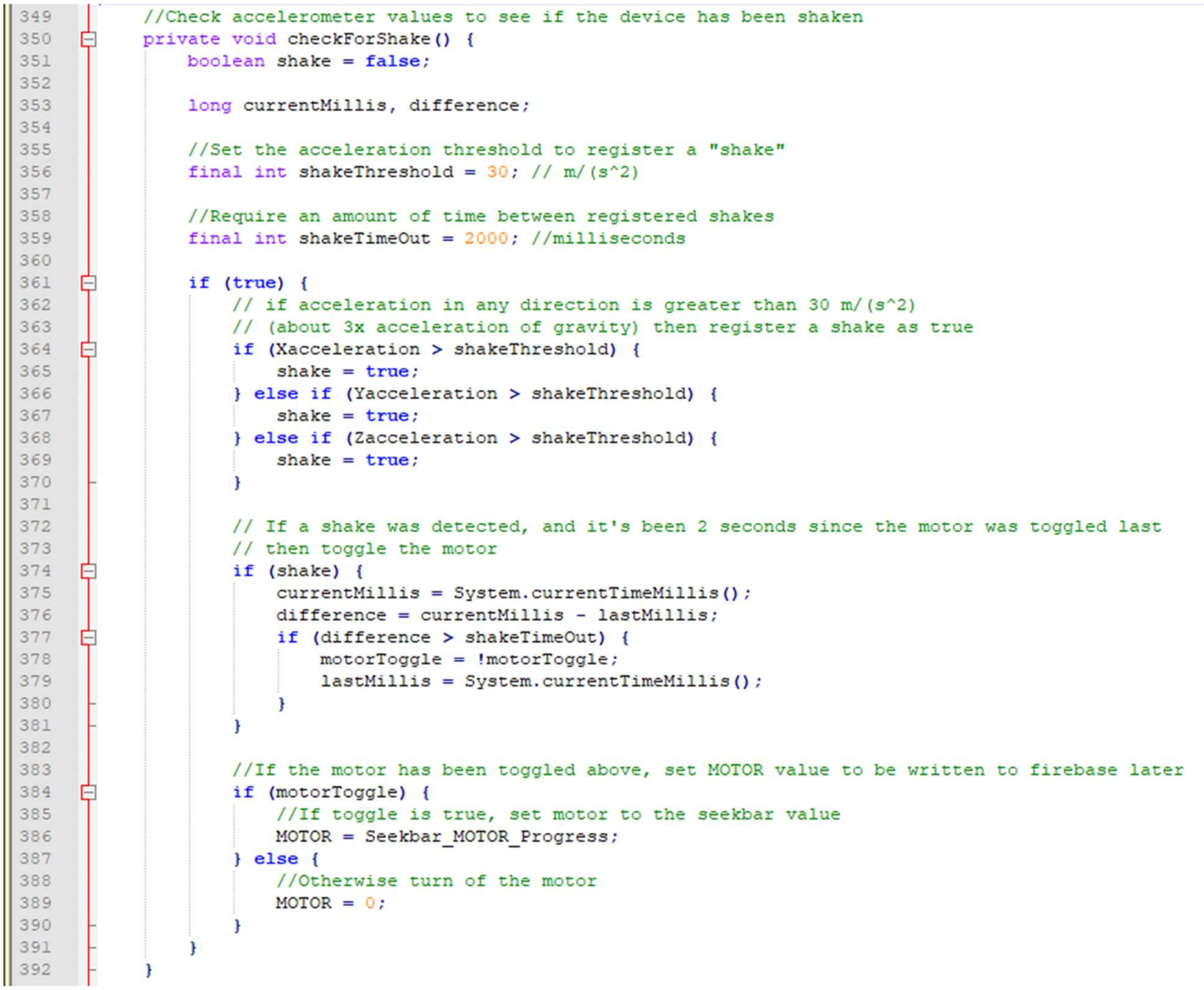
The calculateValue function similarly calculates the angle between the Z-axis acceleration, and the X-Y plane acceleration of the device to determine the angle of the “pitch”. The angle ranges from 0 to 90 degrees and is returned as a value between 0 and 1:



```
//Get the Value based on the current Z-axis orientation
private double calculateValue() {
    double tiltAngle, Value, xyPlaneAccel;
    final double accelerationGravity = 9.81; // m/(s^2)

    //Get the X-Y plane acceleration vector
    xyPlaneAccel=  Math.sqrt(Math.pow(Xacceleration,2) + Math.pow(Yacceleration,2));

    //Set max value as the acceleration of gravity ( m/(s^2) )
    if (xyPlaneAccel > accelerationGravity)
        xyPlaneAccel = accelerationGravity;

    //Use the arctangent function to get the angle of the X-Y acceleration vector vs the
    //Z-axis acceleration vector
    tiltAngle = Math.abs( Math.toDegrees(Math.atan((xyPlaneAccel)/(Zacceleration))) );

    // Normalize the value to 90 degrees (min is 0, max is 1)
    Value = tiltAngle/90.0;

    return Value;
}
```

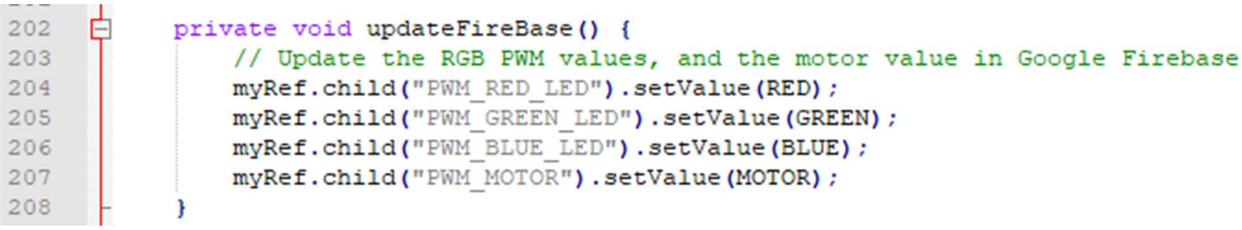
Once the hue, saturation, and value have been returned to the calculateRGB method, it’s time to convert from the HSV color space to the RGB color space. The method uses the algorithm defined from the Wikipedia page https://en.wikipedia.org/wiki/HSL_and_HSV to set the RGB values as class variables.

The checkForShake function determines if any of the accelerometer values have exceeded 30 m/(s^2), which means the device is being shaken. This will toggle the motor ON and OFF, where ON is the user-defined PWM value from the seekbar slider, and OFF is a PWM of 0. The function only recognizes a shake every 2 seconds in order to allow time for the device to stop moving and distinguish subsequent shakes:



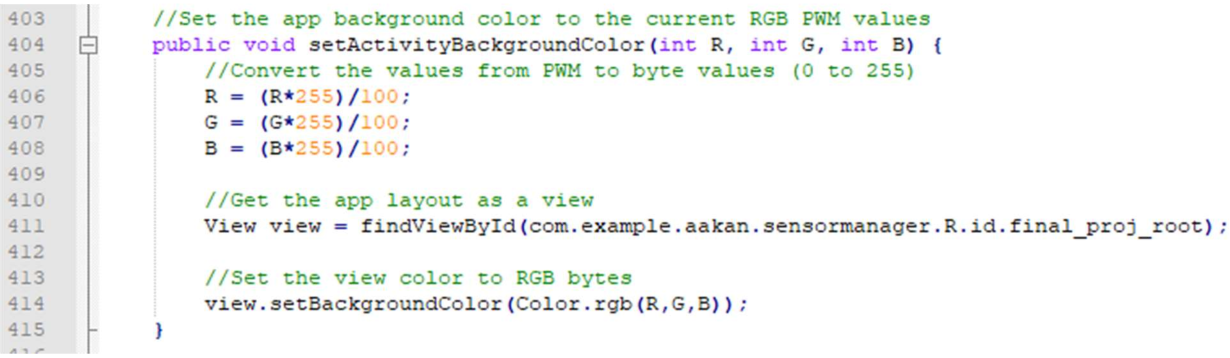
```java
//Check accelerometer values to see if the device has been shaken
private void checkForShake() {
    boolean shake = false;

    long currentMillis, difference;

    //Set the acceleration threshold to register a "shake"
    final int shakeThreshold = 30; // m/(s^2)

    //Require an amount of time between registered shakes
    final int shakeTimeOut = 2000; //milliseconds

    if (true) {
        // if acceleration in any direction is greater than 30 m/(s^2)
        // (about 3x acceleration of gravity) then register a shake as true
        if (Xacceleration > shakeThreshold) {
            shake = true;
        } else if (Yacceleration > shakeThreshold) {
            shake = true;
        } else if (Zacceleration > shakeThreshold) {
            shake = true;
        }

        // If a shake was detected, and it's been 2 seconds since the motor was toggled last
        // then toggle the motor
        if (shake) {
            currentMillis = System.currentTimeMillis();
            difference = currentMillis - lastMillis;
            if (difference > shakeTimeOut) {
                motorToggle = !motorToggle;
                lastMillis = System.currentTimeMillis();
            }
        }

        //If the motor has been toggled above, set MOTOR value to be written to firebase later
        if (motorToggle) {
            //If toggle is true, set motor to the seekbar value
            MOTOR = Seekbar_MOTOR_Progress;
        } else {
            //Otherwise turn of the motor
            MOTOR = 0;
        }
    }
}
```

After the RGB PWM values have been calculated and the motor PWM has been determined, the 4 PWM values are sent to the Google Firebase database which the Android Things device should instantly respond to:



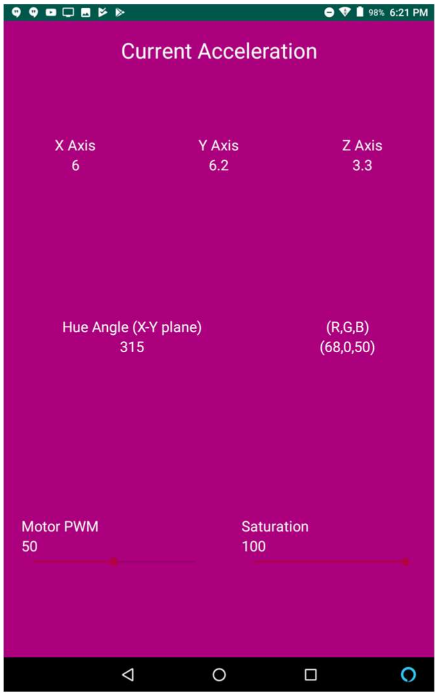
```java
private void updateFireBase() {
    // Update the RGB PWM values, and the motor value in Google Firebase
    myRef.child("PWM_RED_LED").setValue(RED);
    myRef.child("PWM_GREEN_LED").setValue(GREEN);
    myRef.child("PWM_BLUE_LED").setValue(BLUE);
    myRef.child("PWM_MOTOR").setValue(MOTOR);
}
```

Finally, the background of the activity is set to the color of the RGB PWM channels normalized to the 0-255 range (1-byte resolution for each color). This color will match that of the RGB LED connected to the Android Things device:



```
    //Set the app background color to the current RGB PWM values
    public void setActivityBackgroundColor(int R, int G, int B) {
        //Convert the values from PWM to byte values (0 to 255)
        R = (R*255)/100;
        G = (G*255)/100;
        B = (B*255)/100;

        //Get the app layout as a view
        View view = findViewById(com.example.aakan.sensormanager.R.id.final_proj_root);

        //Set the view color to RGB bytes
        view.setBackgroundColor(Color.rgb(R,G,B));
    }
```

The app activity looks like the following in portrait and landscape modes. In these cases, the value is nearly 100%, the saturation is 100%, and the hue is determined by the orientation which result in the different background colors:



Current Acceleration

X Axis
6

Y Axis
6.2

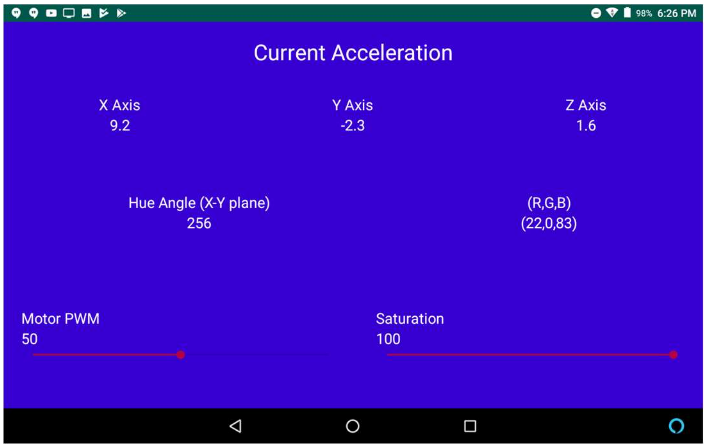
Z Axis
3.3

Hue Angle (X-Y plane)
315

(R,G,B)
(68,0,50)

Motor PWM
50

Saturation
100



Current Acceleration

X Axis
9.2

Y Axis
-2.3

Z Axis
1.6

Hue Angle (X-Y plane)
256

(R,G,B)
(22,0,83)

Motor PWM
50

Saturation
100

Android Things application:

Find the app here: .\AndroidThingsApp

Find the main activity here: .\AndroidThingsApp\HWProject\app\src\main\java \alec\androidthingsece558\HomeActivity.java

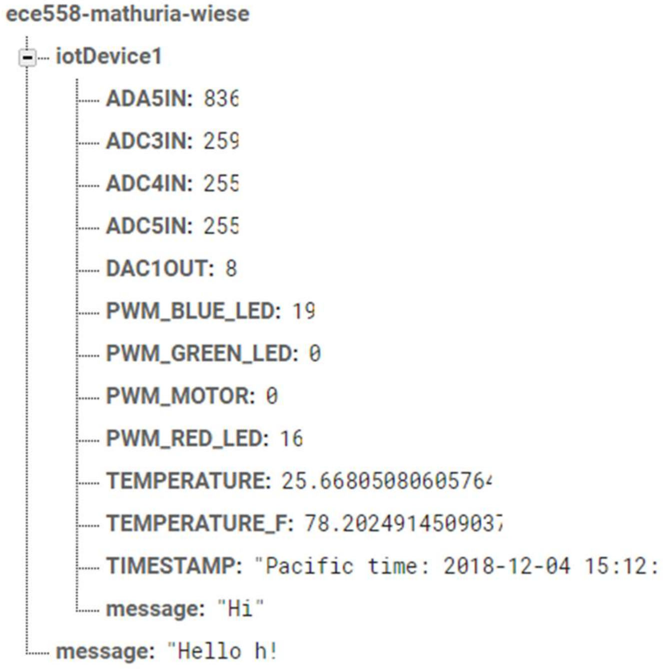
Find JavaDoc comments here: .\AndroidThingsApp\JavaDoc

The goal of this app was to control the color of the RGB LED and to set the motor PWM speed. This Android Things app was very similar to project 3. The app ran on a Raspberry Pi 3, interfaced with the PIC microcontroller, and connected to Google Firebase. The app's only functionality is to respond to changes in the Google Firebase database and update the PWM channels that control the motor speed and the RGB lights in the LED by writing to the PIC microcontroller.

We accomplished this task by creating a ValueEventListener that responds to Google Firebase database changes. Any time a change is detected, the Raspberry Pi 3 writes the PWM values to their respective registers on the PIC microcontroller. We re-used methods created in project 3 to convert the data to bytes and write to registers using I2C. See the Java code or JavaDoc listed above for implementation details.

Firebase database:

In this project, we are using the same firebase real-time database as project 3. To control the LED color, brightness, and saturation we use "PWM_BLUE_LED", "PWM_RED_LED" and "PWM_GREEN_LED" child values. When the motor is ON, we send the PWM motor value to the "PWM_MOTOR" child.



```
ece558-mathuria-wiese
  iotDevice1
      ADA5IN: 836
      ADC3IN: 259
      ADC4IN: 255
      ADC5IN: 255
      DAC1OUT: 8
      PWM_BLUE_LED: 19
      PWM_GREEN_LED: 0
      PWM_MOTOR: 0
      PWM_RED_LED: 16
      TEMPERATURE: 25.6680508060576
      TEMPERATURE_F: 78.202491450903
      TIMESTAMP: "Pacific time: 2018-12-04 15:12:
      message: "Hi"
  message: "Hello h!
```

Tools and Hardware used:

- Raspberry Pi 3
- PIC for PWM control
- RGB LED
- Google firebase
- Device Accelerometer sensor

Goals Achieved:

- Required Features
  - Rotating device on “roll” axis changes Hue
  - Rotating device on “pitch” changes Value/brightness
  - Shaking the device toggles the motor on and off

- Additional Features
  - The color of the mobile app background changes to match the LED
  - Saturation and motor speed seekbars to change values

Design Challenges:

- Creating the algorithms for determining Hue and Value using trigonometry calculations from accelerometer readings.
- The challenge in changing the background color was related to accessing the ID of the view. The view could not be accessed by just using the view ID. Then we tried to access the view ID by taking package name then it worked perfectly.

Video of Project Demonstration:

https://youtu.be/IZfvH51OIkQ