組込みプログラム開発　Android アプリ　レポート

202220597 佐藤昂也

1. アプリ開発の背景

今回の Android アプリ開発において、私は、デバイスのセンサーとカメラを用いて物の長さを測定するアプリケーションを作成した。

私は絵を描くことを一つの趣味としており、物を描くときにそのものの大きさなどの特徴を知りたいと思うことが多い。このような需要は、グラフィックスのデザインや設計などに携わる人にも通じると考えられ、また、日常的な要求としても、物の長さを測定することはよくあるだろう。そこで、巻き尺が近くにない場合でも、デバイスだけで簡単に物の長さを測定できるようなアプリケーションを作れないかと考えたのが今回のアプリ開発のきっかけである。

2. アプリの設計

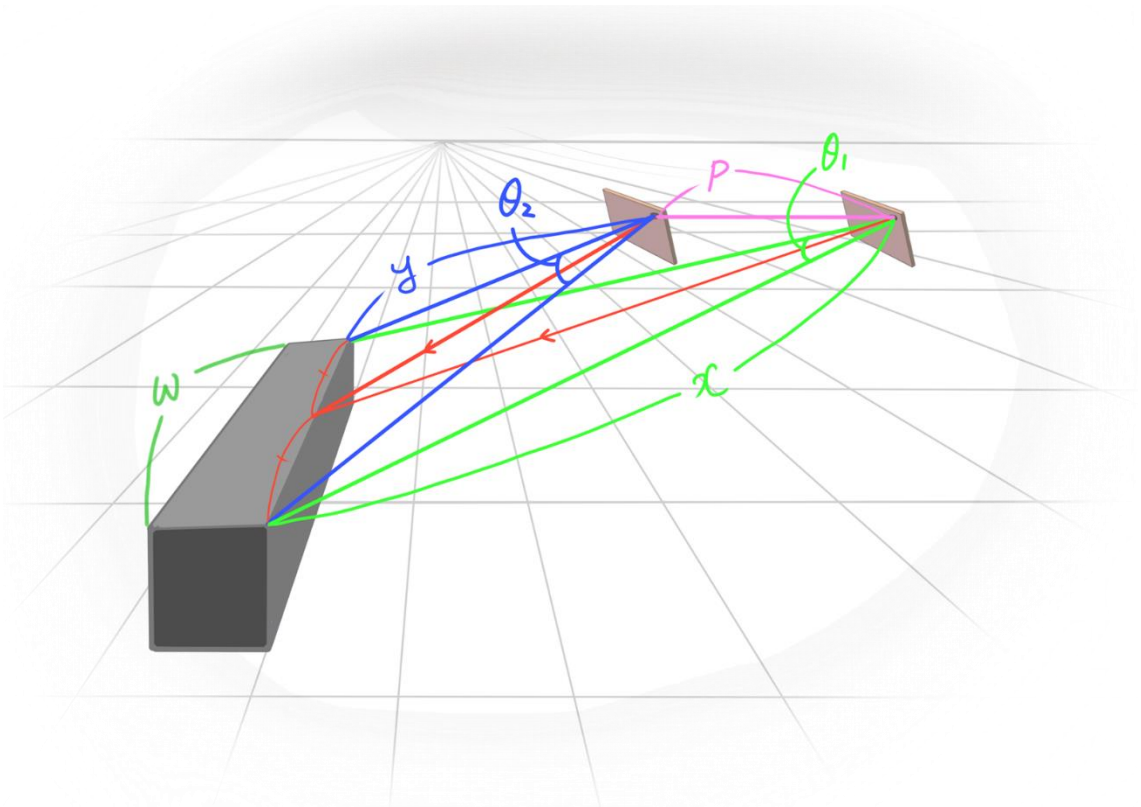

図 1　各角度・長さの設定

図 1、2 のような角度・長さの情報を用いて、幾何的に対象物の長さを求めることを考えた。図 1 のように対象物の求めたい幅（長さ）を w と置き、ユーザーがデバイスを持って対象物の正面に立つとする。初期位置（図の右側）において、タブレットの視点を、求めたい長さの端点から端点へ動かしたときの角度を θ_1 とし、そこから一歩分の長さ p だけ進んだ位置から再度測った角度を θ_2 とする。ここで、p は既知量とする。そのため、ユーザーは自身の歩幅をあらかじめ把握して入力しておくことを前提とする。因みに一般的に歩幅は身長のおよそ 0.45 倍程度になるといわれている。また、初期位置（のカメラの位置）から一つの端点までの距離を x とし、一歩進んだ位置から一つの端点までの距離は y とする。この時、幾何的に次のような関係式が成り立つ。

$$w = 2\,x\,sin\frac{\theta_1}{2} \;= 2\,y\,sin\frac{\theta_2}{2}$$

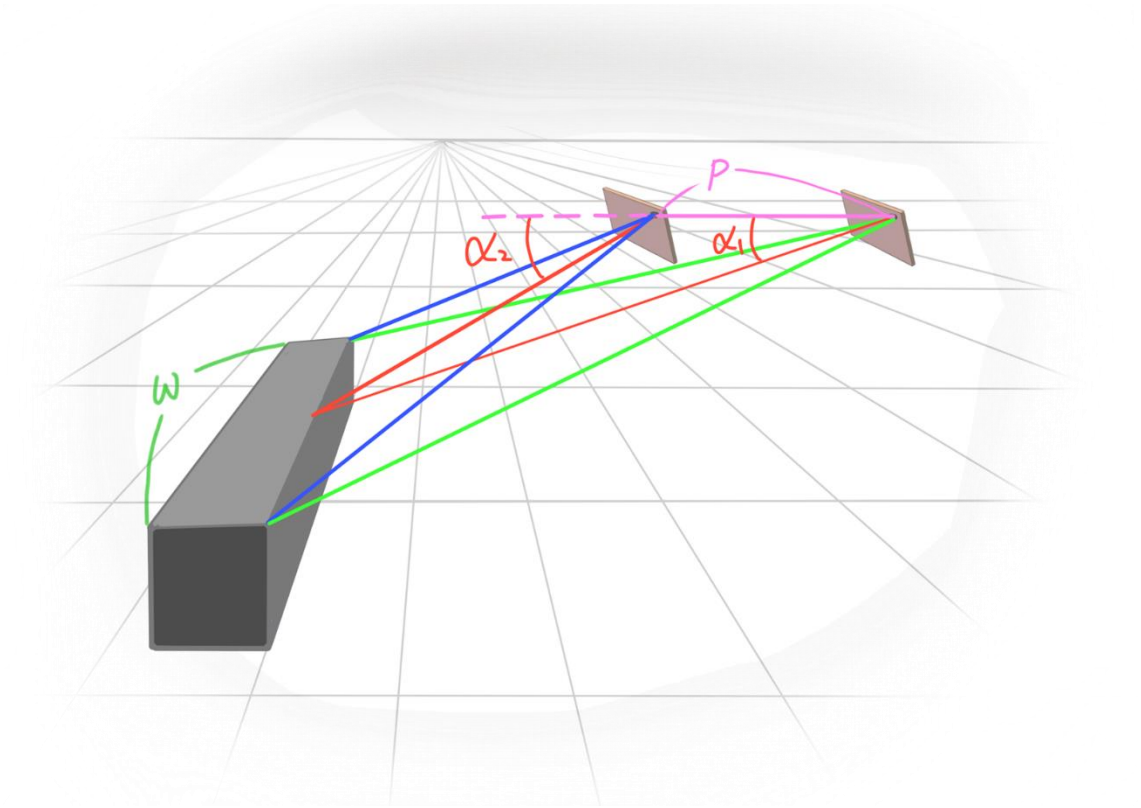

図2　デバイスの傾き角 α_1、α_2 の設定

さらに、デバイスの傾きについても考慮するため、鉛直方向からみたデバイスの傾きの角度を初期位置では α_1、一歩進んだ位置では α_2 とする。この時、図 2 のように、カメラ向く方向と水平方向のなす角もそれぞれ、α_1 と α_2 に対応する。ここでは、次のような式が成り立つことになる。

$$p = x\,cos\frac{\theta_1}{2}cos\,\alpha_1 - ycos\frac{\theta_2}{2}cos\alpha_2$$

以上の数式において、w, x, y という 3 つの未知数に対して等式が 3 つ成り立っているので、未知数を求めることができ、このアプリケーションにおいて目標としている w について解くと次式のようになる。

$$w = \frac{2\,p\sin\frac{\theta_1}{2}\sin\frac{\theta_2}{2}}{\sin\frac{\theta_2}{2}cos\frac{\theta_1}{2}\cos\alpha_1 - sin\frac{\theta_1}{2}cos\frac{\theta_2}{2}cos\alpha_2}$$

この式を用いて、アプリケーション内で、θ_1、θ_2、α_1、α_2という 4 つの変数を測定し、それらをもとに、対象の求めたい長さwを測定し表示することとする。

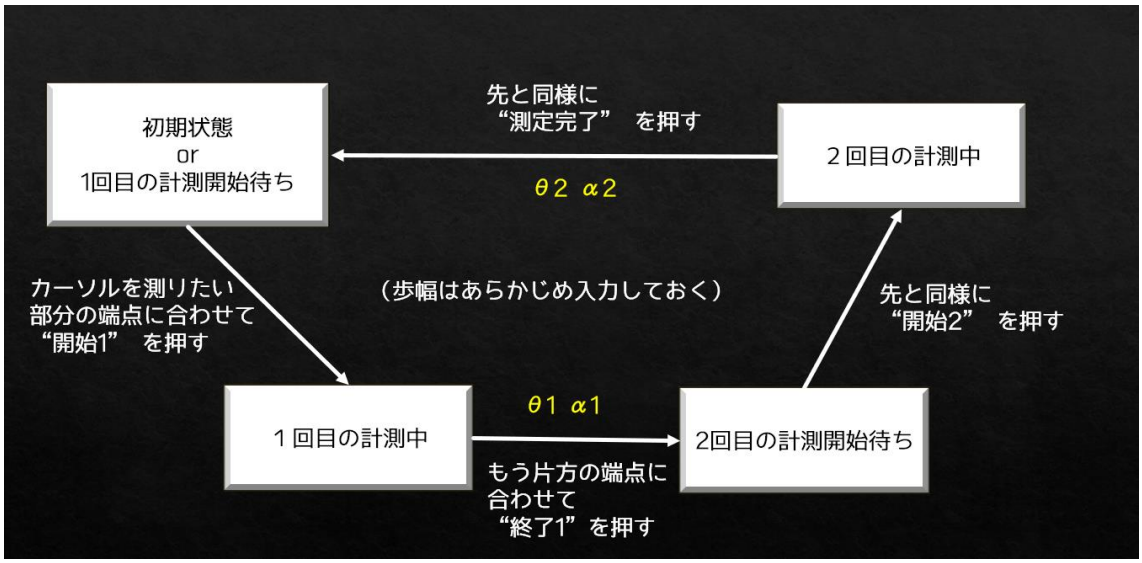

図 3　アプリケーションの状態遷移

以上を踏まえて、測定における状態遷移を図 3 に示す。次の 4 状態を順に遷移していくとして、実装を考える。

① 1 回目の計測開始待ち
② 1 回目の計測中
③ 2 回目の計測待ち
④ 2 回目の計測中

3. 実装

クラス構成は、MainActivity.java と CameraPreview.java の2つであり、CameraPreview.java は授業で扱った CameraSample で用いたものと同じであるので、MainActivity.java のみを図 4 に示す。

MainActivity.java のクラスの中は、onCreate(){}、onSensorChanged(){}、onAccuracyChanged(){}にわかれており、さらに、onCreate(){}の中では、レイアウトの記述、センサーの登録、入力ボタンの設定、計測ボタンを押したときの動作の記述、に分かれる。詳細は図 4 にコードを示す。

レイアウトは、xml ファイルを用いず、コード中で記述する形式を用いた。CameraPreveiw の上にボタンなどのUIを配置するために、CameraPreview とボタンなどをまとめたリニアレイアウトをフレームレイアウトに追加し、そのフレームレイアウトを setContentView()で表示させるという方法をとった。

センサーの登録は OrientationSample と同様である。

入力ボタンの設定においては、ClickListener を設定し、いつでも歩幅を入力することができるようにした。

計測ボタンを押したときの動作の記述部分では、上述した 4 状態を管理するための btnState という変数を用意し、各状態を 0～3 の値で表した。ボタンが押されるごとに、btnState の値が(0→1→2→3→0)と変化し、各状態に合わせた測定値の取得や計算などを行うこととした。

測定値の取得の詳細としては、対象物の視点における角度θは、測定開始時と測定終了時の角度の差により与えられ、デバイスの傾きαは、測定開始時と測定終了時の角度平均として与えられることとした。

```
public class MainActivity extends AppCompatActivity implements SensorEventListener {
    private TextView mAzimuthText, mPitchText, mRollText, tz, tx, ty, stride, strideText, mvText, mv;
    …(その他必要な変数の宣言)

    @Override
    protected void onCreate(Bundle savedInstanceState) {
    //メインの処理
      //レイアウトの記述
        fl.addView(cameraPreview); //fl はあらかじめ宣言しておいた FrameLayout
        fl.addView(linearLayout2);
        fl.addView(linearLayout);
        setContentView(fl);
      //SensorManager 等センサーの登録(OrientationSample 同様なので省略)
      //入力ボタンの設定
        getstridebtn.setOnClickListener(v->{
            String text = strideinput.getText().toString();
```

```
                strideText.setText(text);
                strideValue = Double.parseDouble(text);
            });
        //計測ボタンを押したときの動作の記述
btn.setOnClickListener(new View.OnClickListener(){
                public void onClick(View arg0){
                    SensorManager.getRotationMatrix(mInRotaionMatrix, mInclinationMatrix,
mAccelerationValue, mGeoMagneticValue);
                    SensorManager.remapCoordinateSystem(mInRotaionMatrix, SensorManager.AXIS_X,
SensorManager.AXIS_Z, mOutRotaionMatrix);
                    SensorManager.getOrientation(mOutRotaionMatrix, mOrientationValue);
                    if(btnState == 0){
                        btnState = 1;
                        btn.setText("終了 1");
                        theta1prev = (double)mOrientationValue[0];
                        alpha1prev = (double)mOrientationValue[1];
                    }else if(btnState == 1){
                        btnState = 2;
                        btn.setText("開始 2");
                        theta1next = (double)mOrientationValue[0];
                        alpha1next = (double)mOrientationValue[1];
                        theta1 = Math.abs(theta1next - theta1prev);
                        alpha1 = Math.abs((alpha1next + alpha1prev)/2);
                        String t1Value = String.valueOf(Math.toDegrees(theta1));
                        String a1Value = String.valueOf(Math.toDegrees(alpha1));
                        t1.setText(t1Value);
                        a1.setText(a1Value);
                    }else if(btnState == 2){
                        btnState = 3;
                        btn.setText("測定完了");
                        theta2prev = (double)mOrientationValue[0];
                        alpha2prev = (double)mOrientationValue[1];
                    }else if(btnState == 3){
                        btnState = 0;
                        btn.setText("開始 1");
                        theta2prev = (double)mOrientationValue[0];
```

```
                        alpha2prev = (double)mOrientationValue[1];
                        theta2 = Math.abs(theta2next - theta2prev);
                        alpha2 = Math.abs((alpha2next + alpha2prev)/2);
                        String t2Value = String.valueOf(Math.toDegrees(theta2));
                        String a2Value = String.valueOf(Math.toDegrees(alpha2)
                        );
                        t2.setText(t2Value);
                        a2.setText(a2Value);
                        object_width = 2 * strideValue * Math.sin(theta1/2) * Math.sin(theta2/2) /
(Math.sin(theta2/2) * Math.cos(theta1/2) * Math.cos(alpha1) - Math.sin(theta1/2) * Math.cos(theta2/2) *
Math.cos(alpha2));
                        String measuredValue = String.valueOf(object_width);
                        mvText.setText(measuredValue);
                    }
                }
            });
}
    @Override
    public void onSensorChanged(SensorEvent sensorEvent) {
        //常にデバイスの方位を表示するようにする。
        switch (sensorEvent.sensor.getType()){
            case Sensor.TYPE_MAGNETIC_FIELD:mGeoMagneticValue = sensorEvent.values.clone(); break;
            case Sensor.TYPE_ACCELEROMETER:mAccelerationValue = sensorEvent.values.clone(); break;
        }
        SensorManager.getRotationMatrix(mInRotaionMatrix, mInclinationMatrix, mAccelerationValue,
mGeoMagneticValue);
        SensorManager.remapCoordinateSystem(mInRotaionMatrix, SensorManager.AXIS_X,
SensorManager.AXIS_Z, mOutRotaionMatrix);
        SensorManager.getOrientation(mOutRotaionMatrix, mOrientationValue);
        String azimuthText = String.valueOf(Math.floor(Math.toDegrees((double)mOrientationValue[0])));
        String pitchText = String.valueOf(Math.floor(Math.toDegrees((double)mOrientationValue[1])));
        String rollText = String.valueOf(Math.floor(Math.toDegrees((double)mOrientationValue[2])));
        mAzimuthText.setText(azimuthText);
        mPitchText.setText(pitchText);
        mRollText.setText(rollText);
    }
```

```
    @Override
    public void onAccuracyChanged(Sensor sensor, int i) {
      //処理の記述は特になし
    }
}
```

図４　MainActivity.java の概要

4. 動作例

図5　アプリケーションのレイアウトのレイアウト

アプリケーションのレイアウトを図５に示す。以下に動作手順を示す。

1. 入力欄に歩幅をメートル単位で入力し、歩幅決定を押す。（入力欄の上の歩幅欄に入力した長さが表示されるようになる。）
2. 図６左上のように、対象物の端にカーソルを合わせ、測定開始ボタンを押す
3. 図６左下のように、対象物のもう一方の端にカーソルを合わせ、測定終了ボタンを押す。
4. 一歩前に進む。
5. 図6右上のように、対象物の端にカーソルを合わせ、測定開始ボタンを押す
6. 図６右下のように、対象物のもう一方の端にカーソルを合わせ、測定終了ボタンを押す。
7. 図７のように、測定終了するとボタンの上の欄に測定値が表示される。

図6　測定の流れ(左上→左下→右上→右下)

図７　測定終了するとボタンの上の欄に測定値が表示される。

5. まとめ

実際にアプリケーションを作った結果、動作としては、実測値よりも下振れすることが多く、誤差も問題が非常に大きい。これは、求める際に使用した式の問題も考えられる。掛け算の形をしている項が多く、誤差を増幅しやすい、もしくは、分母が小さくなるケースでは誤差の影響が大きくなるといったことが考えられるだろう。また、測定の仕方として、カメラ部分を空間の一点に固定しながらデバイスの角度を傾けるように行わなければ正確な値を得られないという特徴も存在する。

また、プログラム中にレイアウトを記述したことでコードがかなり長くなってしまった。

もとは屋内で使用することを想定していたが、先生のご指摘の通り、屋外の大きな対象物を測定するというように用途を広げて考えると、アプリケーションの可能性も広がると思われる。