```java
//ABSOLUTE CHAOS CONTROL

package org.firstinspires.ftc.teamcode;

//TeleOp and Hardware
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import com.qualcomm.robotcore.util.ElapsedTime;
import com.qualcomm.robotcore.eventloop.opmode.Disabled;

//Sensors
//   Gyro
import com.qualcomm.hardware.modernrobotics.ModernRoboticsI2cGyro;
import com.qualcomm.robotcore.hardware.IntegratingGyroscope;
//  ODS
import com.qualcomm.robotcore.hardware.OpticalDistanceSensor;
//  Color Sensor
import com.qualcomm.robotcore.hardware.NormalizedColorSensor;
import com.qualcomm.robotcore.hardware.NormalizedRGBA;
import com.qualcomm.robotcore.hardware.SwitchableLight;

//Gyro References
import org.firstinspires.ftc.robotcore.external.navigation.AngleUnit;
import org.firstinspires.ftc.robotcore.external.navigation.AxesOrder;
import org.firstinspires.ftc.robotcore.external.navigation.AxesReference;

//Android App Control
import android.app.Activity;
import android.graphics.Color;
import android.view.View;



@TeleOp(name="AbsoluteChaosControl", group="Basic OP Mode")

public class AbsoluteChaosControl extends LinearOpMode{
   //Initializes hardware
   private DcMotor motor1;
   private DcMotor motor2;
   private DcMotor motor3;
   private DcMotor motor4;
   //Initializes Sensors
   //Gyro
   private IntegratingGyroscope gyro;
   private ModernRoboticsI2cGyro modernRoboticsI2cGyro;
   //ODS
   private OpticalDistanceSensor ods;
   //Color Sensor
   NormalizedColorSensor colorSensor;
```

```
    View relativeLayout;

    @Override public void runOpMode() throws InterruptedException {

        // Get a reference to the RelativeLayout so we can later change the background
        // color of the Robot Controller app to match the hue detected by the RGB
sensor.
        int relativeLayoutId =
hardwareMap.appContext.getResources().getIdentifier("RelativeLayout", "id",
hardwareMap.appContext.getPackageName());
        relativeLayout = ((Activity)
hardwareMap.appContext).findViewById(relativeLayoutId);

        try {
            runSample(); // actually execute the sample
        } finally {
            // On the way out, *guarantee* that the background is reasonable. It
doesn't actually start off
            // as pure white, but it's too much work to dig out what actually was used,
and this is good
            // enough to at least make the screen reasonable again.
            // Set the panel back to the default color
            relativeLayout.post(new Runnable() {
                public void run() {
                    relativeLayout.setBackgroundColor(Color.WHITE);
                }
            });
        }
    }


    public ElapsedTime timer = new ElapsedTime();

    public void runSample() throws InterruptedException{
        double power = 0.2;

        float[] hsvValues = new float[3];
        final float values[] = hsvValues;

        //Telemetry initialized message
        telemetry.addData(  "Status",    "Initialized");
        telemetry.update();

        //Hardware definitions
        motor1 = hardwareMap.get(DcMotor.class,"motor1");
        motor2 = hardwareMap.get(DcMotor.class,"motor2");
        motor3 = hardwareMap.get(DcMotor.class,"motor3");
        motor4 = hardwareMap.get(DcMotor.class,"motor4");
        //Sensors
        //Gyro
```

```
        modernRoboticsI2cGyro = hardwareMap.get(ModernRoboticsI2cGyro.class, "gyro");
        gyro = (IntegratingGyroscope)modernRoboticsI2cGyro;
        //ODS
        ods = hardwareMap.opticalDistanceSensor.get("ods");
        ods.enableLed(true);
        //Color Sensor
        colorSensor = hardwareMap.get(NormalizedColorSensor.class, "color_sensor");

        if (colorSensor instanceof SwitchableLight) {
            ((SwitchableLight)colorSensor).enableLight(true);
        }

        telemetry.log().add("Gyro Calibrating. Do Not Move!");
        modernRoboticsI2cGyro.calibrate();

        while (!isStopRequested() && modernRoboticsI2cGyro.isCalibrating()) {
            telemetry.addData("calibrating", "%s", Math.round(timer.seconds()) % 2 == 0
? "|.." : "..|");
            telemetry.update();
            sleep(50);
        }

        telemetry.log().clear(); telemetry.log().add("Gyro Calibrated. Press Start.");
        telemetry.clear(); telemetry.update();

        //Variable instantiation
        double left_y, left_x;
        double left_t, right_t;
        double g_angle;
        double abs_x, abs_y;
        int iteration = 0;
        boolean bPrevState = false;
        boolean bCurrState;
        //Wait until phone interrupt
        waitForStart();
        timer.reset();
        //While loop for robot operation
        while (opModeIsActive()){
            //long delta_t = (time_base - timer.nanoseconds())/timer.SECOND_IN_NANO;
            //sigmoid(delta_t, false, false,false, 0.5, 0.1, 2);
            iteration ++;
            bCurrState = gamepad1.x;
            //Toggle for light, this is the general toggle setup.
            if (bCurrState != bPrevState) {
                if (bCurrState) {
                    if (colorSensor instanceof SwitchableLight) {
                        SwitchableLight light = (SwitchableLight)colorSensor;
                        light.enableLight(!light.isLightOn());
                    }
                }
```

```
            }
            bPrevState = bCurrState;

            //Gamepad's left stick x and y values
            left_y = -gamepad1.left_stick_y;
            left_x = gamepad1.left_stick_x;

            //Gamepad's left and right trigger values
            left_t = gamepad1.left_trigger;
            right_t = gamepad1.right_trigger;

            //Robot Heading Unit Vector

            //Boolean for distance reset
            g_angle = gyro.getAngularOrientation(AxesReference.INTRINSIC,
AxesOrder.ZYX, AngleUnit.DEGREES).firstAngle;
            g_angle *= Math.PI/180;
            abs_x = (left_x*Math.cos(-g_angle)-left_y*Math.sin(-g_angle));
            abs_y = (left_x*Math.sin(-g_angle)+left_y*Math.cos(-g_angle));

            //Power variable (0,1), average drive train motor speed

            //x component vector
            //motor 2
            motor2.setPower(power*(-abs_x+left_t-right_t));
            //motor4
            motor4.setPower(power*(abs_x+left_t-right_t));

            //y vector
            //motor1
            motor1.setPower(power*(abs_y+left_t-right_t));
            //motor3
            motor3.setPower(power*(-abs_y+left_t-right_t));

            //More telemetry. Adds left stick values and trigger values
            /*
            telemetry.addLine()
                    .addData("right_y", left_y)
                    .addData("left_x", left_x );
            telemetry.addLine()
                    .addData("Motor 1+3", abs_y);
            telemetry.addLine()
                    .addData("Motor 2+4", abs_x);
            telemetry.addLine()
                    .addData("angle", g_angle);
            */
            NormalizedRGBA colors = colorSensor.getNormalizedColors();
            Color.colorToHSV(colors.toColor(), hsvValues);
            telemetry.addLine()
                    .addData("H", "%.3f", hsvValues[0])
```

```
                .addData("S", "%.3f", hsvValues[1])
                .addData("V", "%.3f", hsvValues[2]);
        telemetry.addLine()
                .addData("a", "%.3f", colors.alpha)
                .addData("r", "%.3f", colors.red)
                .addData("g", "%.3f", colors.green)
                .addData("b", "%.3f", colors.blue);
        int color = colors.toColor();
        telemetry.addLine("raw Android color: ")
                .addData("a", "%02x", Color.alpha(color))
                .addData("r", "%02x", Color.red(color))
                .addData("g", "%02x", Color.green(color))
                .addData("b", "%02x", Color.blue(color));
        float max = Math.max(Math.max(Math.max(colors.red, colors.green),
colors.blue), colors.alpha);
        colors.red   /= max;
        colors.green /= max;
        colors.blue  /= max;
        color = colors.toColor();
        telemetry.addLine("normalized color:  ")
                .addData("a", "%02x", Color.alpha(color))
                .addData("r", "%02x", Color.red(color))
                .addData("g", "%02x", Color.green(color))
                .addData("b", "%02x", Color.blue(color));
        telemetry.update();
        // convert the RGB values to HSV values.
        Color.RGBToHSV(Color.red(color), Color.green(color), Color.blue(color),
hsvValues);
        telemetry.addLine()
                .addData("distance", ods.getRawLightDetected());
        telemetry.addLine().addData("distance normal", ods.getLightDetected());
        //telemetry.addLine().addData("Delta_t", delta_t);
        relativeLayout.post(new Runnable() {
            public void run() {
                relativeLayout.setBackgroundColor(Color.HSVToColor(0xff, values));
            }
        });
    }
  }
}
```