Matthew Donnelly
Daniel Bjorklund
CSC375
https://youtu.be/V3kr9F2nYJk
https://youtu.be/XBu_V99OMFg

Lab 5 report: team 2

1A. This program is all together very simple. We have a task for display that checks whether it should be displaying (checks based off bump), if not it flashes “wait” on the screen. Otherwise it displays the required values. The values are updated in a separate task that simply calls the methods that return the required information and storing the result in variables. The only oddity is getColorName, which we must translate to a string through use of a switch statement. In the below screenshot the updateVariables task is condensed and shown in another screenshot.

```
#pragma config(StandardModel, "EV3_REMBOT");
bool wait=true;
short ambient;
short reflected;
string color;

long redValue;
long greenValue;
long blueValue;

task updateVariables(){...
}

task display(){
    while (true){
        if (wait){
            displayBigTextLine(2, "Wait");
            eraseDisplay();
            }else{
            displayBigTextLine(2, "No Wait");
            displayBigTextLine(4, "Ambient: %d ", ambient);
            displayBigTextLine(6, "Reflected: %d ", reflected);
            displayBigTextLine(8, "Color: %s", color);
            displayBigTextLine(12,"RGB: %d, %d, %d", redValue, greenValue, blueValue);
        }
    }
}
task checkbump(){
    while (true){
        if (getBumpedValue(touchSensor)==1){
            wait=false;
            }else{
            wait=true;
            resetBumpedValue(touchSensor);
        }
    }
}
task main(){
    startTask(checkbump,7);
    startTask(display,7);
    startTask(updateVariables, 7);
    while(true){}
}
```

```
task updateVariables(){
    while(true){
        sleep(100);
        //ambient
        ambient = getColorAmbient(colorSensor);
        //reflection
        reflected = getColorReflected(colorSensor);
        //Name color
        switch(getColorName(colorSensor)){
        case colorBlack:
            color = "Black";
            break;

        case colorBlue:
            color = "Blue";
            break;

        case colorGreen:
            color = "Green";
            break;

        case colorYellow:
            color = "Yellow";
            break;

        case colorRed:
            color = "Red";
            break;

        case colorWhite:
            color = "White";
            break;

        case colorBrown:
            color = "Brown";
            break;

        default:
            color = "None";

        }
        // RGB values
        getColorRGB(colorSensor, redValue, greenValue, blueValue);
    }
}
```

1B. This one only has a display for debugging purposes, to help us assess what to use for a threshold. Move runs constantly, checking whether the color sensor detects the black tape. If it does it changes the direction of the motors and sleeps for a second (to let the robot get away from the black tape). Really nothing groundbreaking here.

```
#pragma config(StandardModel, "EV3_REMBOT");
int threshold = 29;
int speed=25;

task moves(){
    while(true){
        if(getColorReflected (colorSensor) > threshold){
            setMotorSpeed(leftMotor, speed);
            setMotorSpeed(rightMotor, speed);
            }else{
                speed=speed*(-1);
                setMotorSpeed(leftMotor, speed);
                setMotorSpeed(rightMotor, speed);
                sleep(1000);
        }
    }
}

task display(){
    while (true){
        displayBigTextLine(2, "%d", getColorReflected(colorSensor));
    }
}

task main(){
    startTask(moves, 7);
    startTask(display, 7);

    while(true){}


}
```

3A. This question has no surprises. We have a task the updates the variables by storing the values returned by the various method calls. And then a separate task displays them.

```
#pragma config(StandardModel, "EV3_REMBOT");

long degree;
long heading;
long rate;




task updateVariables(){
    while(true){
        degree= getGyroDegrees(gyroSensor);
        heading= getGyroHeading(gyroSensor);
        rate= getGyroRate(gyroSensor);

    }
}


task display(){
    while (true){
            displayBigTextLine(2, "Degree: %d", degree);
            displayBigTextLine(4, "Heading: %d ", heading);
            displayBigTextLine(6, "Rate: %d ", rate);
    }
}

task main(){
    resetGyro(gyroSensor);
    startTask(display,7);
    startTask(updateVariables, 7);

    while(true){}


}
```

3B. The insertGryoBuffer is straight from the lab handout and is not shown in the screenshot. The average just sums the array and divides it by the number of elements. The display task once again just displays the different required values. But updateVariables adds values to the different datalog series as well as inserts to the moving average and averages the array. We noticed that with our current bufferSize of 6 that the average was not too different from the actual measured value. The peaks/troughs were more compressed than the raw but not by a massive amount. I imagine if we tested with a larger buffer that the compression of the moving average would be larger.

```
#pragma config(StandardModel, "EV3_REMBOT");
#define DATALOG_SERIES_0 0
#define DATALOG_SERIES_1 1
long degree;
long heading;
long rate;
const int bufferSize = 6;
int gyroBuffer[bufferSize];

void insertGyroBuffer(int var){...
}

int average(){
    int x = 0;
    int i;
    for(i=0;i<bufferSize-1;i++){
        x+=gyroBuffer[i];
    }
    return x/bufferSize;
}

    task updateVariables(){
        while(true){
            int avg;
            int rateTMP = getGyroRate(gyroSensor)
            insertGyroBuffer(rateTMP);
            datalogAddValue(DATALOG_SERIES_1, rateTMP);

            degree= getGyroDegrees(gyroSensor);
            heading= getGyroHeading(gyroSensor);
            avg = average();
            datalogAddValue(DATALOG_SERIES_0, avg);
            rate= avg;

        }
    }


    task display(){
        while (true){
            displayBigTextLine(2, "Degree: %d", degree);
            displayBigTextLine(4, "Heading: %d ", heading);
            displayBigTextLine(6, "Rate: %d ", rate);
        }
    }

    task main(){
        resetGyro(gyroSensor);
        startTask(display,7);
        startTask(updateVariables, 7);
        while(true){}
    }
```

3C. This program is mainly taken from another lab that had proportional turning. The only thing that we had to change was how it was checking the error. Now when it calculates error, it uses the gryo sensor instead of the wheel encoders. The only thing we would have to change to make it pivot is the motorMethod. Currently it uses a sync motor method, but we could have it use just one motor.

```
#pragma config(StandardModel, "EV3_REMBOT");
float desiredDeg;
float currentDeg;
float error;
int desiredPower=25;
int currentPower;
int slewRate = 1;
float k =1;
float tolerance = 1;
int dir;

task slew(){
    while (true){
        error = desiredDeg - currentDeg;
        if(abs(error)>tolerance){//this should take care of the
            desiredPower = k*error*-1;//this will be negative if
            }else{
            desiredPower=0;
        }
        if(error<0){//error calculation already tells us which wa
            if((currentPower + slewRate) < desiredPower){
                currentPower += slewRate;
                }else{
                currentPower = desiredPower;
                }if (currentPower> desiredPower){
                currentPower = desiredPower;
            }
        }
    }
}

task motorControl(){
    while(true){
        setMotorSync(leftMotor, rightMotor, -100, desiredPower);

    }
}

task update(){
    while (true){
        currentDeg=getGyroDegrees(gyroSensor);
    }
}
task display(){
    while (true){

        displayBigTextLine(4, "DesDegree: %d", desiredDeg);
        displayBigTextLine(6, "CurrPower: %d", currentPower);
        displayBigTextLine(8, "DesPower: %d", desiredPower);
        displayBigTextLine(10, "Error: %d", error);
    }
}
```

```
void turn(int deg){
    currentDeg= getGyroDegrees(gyroSensor);
    desiredDeg=deg+currentDeg;
    startTask(slew,7);
    startTask(motorControl,7);
    startTask(update,7);
}

task main(){
    startTask(display,7);
    turn(-60);
    while (true){}
}
```