Matt Donnelly
Danny Bjorklund

Lab 7 report

Pictures on next pages

Overall this question isn't that complicated once we understood the easy way to get around the object (doing a circle instead of a box like path). We copied the proportional line following from lab 6 Q2 and made it into a task. Then all we had to do was implement a way to turn a certain number of degrees (which was easy except for one mistake we made which threw us off). Then we made a function that made the robot to a circle around the box and stop once it hit the line. Then once it hits the line, it just starts the line following task again and it takes care of itself. If we had not been told to use a circular trajectory to get around the object we may have never have gotten it, so thank you for that help.

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

// To turn a certian degree
float degree_turned_so_far;
int degree_global;
int ratio;
bool turn_is_done;

task monitor_deg_of_turn(){
    float starting_deg_robot_facing=getGyroDegrees(gyroSensor);
    degree_turned_so_far=0;
    while (true){
        degree_turned_so_far= getGyroDegrees(gyroSensor)-starting_deg_robot_facing;
    }
}
void ratio_finder(){
    int error_turn;
    error_turn=degree_turned_so_far-degree_global;
    if(error_turn>=1){
        ratio=-100;
        }else{
        ratio=100;
    }
    return;
}
void turn(int degree){
    turn_is_done=false;
    degree_global=degree;
    ratio_finder();
    startTask(monitor_deg_of_turn);
    while (true){
        setMotorSync(leftMotor, rightMotor, ratio, 20);
        if(degree_turned_so_far>= degree){
            setMotorSync(leftMotor, rightMotor, 0, 0);
            stopTask(monitor_deg_of_turn);
            turn_is_done=true;
            return;
        }
    }
}
```

```
// to follow the line

int dark = 5;
int light = 38;
int current_color;
int threshold=(dark+light)/2;
float error;
float desiredChange;


int motorPower = 20;
float k = 1.99;



task updateColor(){
    while(true){
        current_color = getColorReflected(colorSensor);
    }
}

task motorControl(){
    while(true){
        setMotorSync(leftMotor, rightMotor, desiredChange, motorPower);
    }
}

task updateDesiredChange(){
    while(true){
        error = current_color-threshold;
        desiredChange = k*error;
    }
}


void start_follow_line(){
    startTask(updateColor, 7);

    startTask(updateDesiredChange, 7);
    startTask(motorControl, 7);
}

void stop_follow_line(){
    stopTask(updateColor);
    //stopTask(display);
    stopTask(updateDesiredChange);
    stopTask(motorControl);
}
```

```
// range updater

float global_current_range;

task range_update(){
    while (true){
        global_current_range=getUSDistance(sonarSensor);
    }
}


//// find if it is past

//void move_past_object(){
//  while(true){
//      go(300);
//      turn(-90);
//      if(global_current_range<250){
//          return;
//      }
//  }
//}

// display
task display(){
    while(true){
        //displayBigTextLine(4, "%d", current_color);
        displayBigTextLine(6, "threshold %d", threshold);
        displayBigTextLine(8, "desCh%d", desiredChange);
        displayBigTextLine(10, "global var%d", global_current_range);
    }
}


void circle(){
    displayBigTextLine(2, "in circle");
    startTask(updateColor, 7);
    while(current_color> threshold-7){
        displayBigTextLine(2, "in while");
        setMotorSpeed(leftMotor, 18);
        setMotorSpeed(rightMotor, 30);

    }
    displayBigTextLine(4, "out while");
    stopTask(updateColor);
    return;
}
```

```
task main(){
    startTask(display, 7);
    startTask(range_update,7);
    sleep(100);
    start_follow_line();
    while( global_current_range>13){ }//about 6in
    stop_follow_line();
    turn(90);
    while(!turn_is_done){}
    displayBigTextLine(12, "past turn");
    circle();
    start_follow_line();

    while(true){}



}
```