1. **Introduction**

1.1 Assignment Name and Due Date

Project 2, Milestone 1, Grid Navigation, Line Following

The assignment was about programming the robot so that using the information gathered by the light sensors, circular and square paths could be followed using proportional control.

9/6/2012

* 1. Team Information

Team # 2

Team Members: Albert An (Mechanical Designer), Mario Farren (Report Writer), Victor Sandberg (Program Coder)

Hours Spent: 10

1. **Experimental Work**

2.1 Experiment Description

An experiment was carried out in order to test the readings shown by the light sensors as they move into different colored terrain. The experiment was done at only 2 LU of height and the method lightsensor.readNormalizedvalue() is used instead of lightSensor.readValue() since Tracker uses the first.

The purpose of the experiment was to demonstrate the necessity of calibrating the robot and how precise readings from this experiment could contribute to coding a smarter calibration.

* 1. Listing of Data

**Graph 1: Raw sensor readings**

**Graph 2: Calibrated Sensor Readings**

* 1. Calculations and Analysis

From the first graph it can be concluded that the sensors show different readings for the same points on the scale. The right sensor’s readings are shifted by around 40 upwards in comparison to the left sensor’s reading.

The second graph takes this shift in the sensor’s reading into account and the curves become much more similar. The midpoint where the readings are the same and the difference becomes 0, is found at 6.1 cm which is equivalent to the middle of the ruler, and is where the tape is found. The graph shows that very near the equilibrium (approx. 1 cm to the right and the left) the differences are very high, but 3 cm to left or the right the differences become almost 0 once again.

* 1. Implementing Findings in the code

A first attempt could have been made calibrating the robot using the fact that when the robot was deviated either to the right or the left by at most 3 cm it should return to the path depending on the size of the difference.

Following advice from the professor we tried the simplest solution: using the calibrate() method every time. To our benefit, it worked very well in both the square and elliptical paths.

1. **Task analysis and Class responsibilities**

1) Tracker: makes the robot track a line and stay on the line through calibrating the light sensors.  
  
2) LineDataCollect: Help understand how the sensors are working and calibrate the sensors

3) Datalogger: stores value from the line data collect and transmits it to data viewer to enable graphing.

**4 Interesting/Challenging/Difficult parts of the project**

Interesting parts: The way slight errors make the robot spin in random directions or different ways we calibrated the robot affected the robot’s movement was fascinating.  
  
Challenging/Difficult: It was hard to make the robot continue after spotting the black marker and then turn after the marker was in the center of the robot to make it accurately turn to make a figure eight shape. We have solved this problem by putting in a time delay so it kept going on after seeing the black marker.

**5 Source Code Listing**

import java.util.Random;  
  
import lejos.robotics.navigation.DifferentialPilot;  
import lejos.util.Delay;  
import lejos.nxt.\*;  
  
/\*\*  
This class needs a higher level controller to implement the navigtion logic<br>  
Responsibility: keep robot on the line till it senses a marker, then stop <br>  
also controls turning to a new line at +- 90 or 180 deg<br>  
Hardware: Two light sensors , shielded, 2 LU above floor.  
Classes used: Pilot, LightSensors<br>  
Control Algorithm: proportional control. estimate distance from centerline<br>  
Calibrates both sensors to line, background  
Updated 9/10/2007 NXT hardware  
@author Roger Glassey  
\*/  
public class Tracker  
{  
  
/\*\*  
\* controls the motors  
\*/  
public DifferentialPilot pilot;  
/\*\*  
\*set by constructor , used by trackline()  
\*/  
private LightSensor leftEye;  
/\*\*  
\*set by constructor , used by trackline()  
\*/  
private LightSensor rightEye;  
/\*\*  
\* controls the direction of turns  
\*/  
private int turnDirection = 1;  
/\*\*  
\* count numbers of turns that been made  
\*/  
private int count = 0;  
  
  
/\*\*  
\*constructor - specifies which sensor ports are left and right  
\*/  
// public Tracker( Pilot thePilot,SensorPort leftI,SensorPort rightI)  
public Tracker(DifferentialPilot thePilot, LightSensor leftEye , LightSensor rightEye)  
{  
pilot = thePilot;  
pilot.setTravelSpeed(15);  
pilot.setRotateSpeed(180);  
pilot.setAcceleration(400);  
this.leftEye = leftEye;  
this.leftEye.setFloodlight(true);  
this.rightEye = rightEye;  
this.rightEye.setFloodlight(true);  
}  
  
/\*\*  
follow line till intersection (a black marker) is detected  
uses proportional control <br>  
Error signal is supplied by CLdistance()<br>  
uses CLdistance(), pilot.steer()  
loop execution about 65 times per second in 1 sec.<br>  
\*/  
public void trackLine() {  
float gain = (float) 0.7f; // you may need to change this for smooth tracking  
// This method needs to detect a black maker.   
while(true) {  
int lval = leftEye.getLightValue();  
int rval = rightEye.getLightValue();   
System.out.println(count);  
int error = CLDistance(lval, rval);   
double control = (error\*gain); // do better  
pilot.steer(control);  
if (lval < -25 || rval < -25) {  
count++;  
Sound.playTone(1000,100);  
pilot.travel(7.8);  
stop();  
break;  
}  
}  
}  
  
public void gridNavigation() {  
Button.waitForAnyPress();  
int numberOfTurns = 0;  
gridNavigation(numberOfTurns);  
}  
  
public boolean gridNavigation(int numberOfTurns) {  
if (numberOfTurns > https://lh4.googleusercontent.com/Cpdn0gYmGhuehbLcO1749Z7BY2-N8kruR7TzZBZoUiWqO4Y1KqIvSAIb6vOzLqugwHmd_ioku0JnF5IIk8ZvWgdj8UdHY-NxgIVy1vJCXCKt-0aKumI{  
return true;  
}  
trackLine();  
Random rng = new Random();  
int turnDir = rng.nextInt(3)-1;  
System.out.println(turnDir);  
int lval = leftEye.getLightValue();  
int rval = rightEye.getLightValue();  
System.out.println(lval);  
System.out.println(rval);  
if (lval > 90 || rval > 90) {  
turnDir = 2;  
}  
pilot.rotate(90\*turnDir);  
numberOfTurns++;  
gridNavigation(numberOfTurns);  
return false;  
}  
  
  
  
/\*\*  
follow the track for 4 complete circuits, turn around, and complete 4 circuits in opposite direction  
\*/  
public void trackAndTurn() {  
Button.waitForAnyPress();  
while(count < https://lh6.googleusercontent.com/NNuEymjJoAD6XLDqakDrHarZGuUgdyDm-9lEVAAXVOT_2Y6l7BEE9NT7Tm45MUxODR8UhaSwO2XtFkC6dyuPZuuRgmWqdW95RBetaHim97bdnm8yJ4c{  
trackLine();  
}  
stop();  
pilot.rotate(180);  
while(count < 16) {  
trackLine();  
}  
stop();  
count = 0;  
}  
  
/\*\*  
\* makes the robot do 4 figure 8 circuits  
uses trackAnEight(int param)   
\*/  
public void trackAnEight() {  
Button.waitForAnyPress();  
int numberOfTurns = 0;  
trackAnEight(numberOfTurns);  
}  
  
/\*\*  
\* Help method of trackAnEight(),  
\* @param numberOfTurns  
\*/  
public boolean trackAnEight(int numberOfTurns) {  
if (numberOfTurns >= https://lh4.googleusercontent.com/9b49WNfnvM7-u-a6mYhR7FcFX4PKQaG28kGy7LFcHqJ9MAzBlIDcQqmTFPcY4hwM9RgU0SBDjFNj0_QwP9PRi9DKnL-tp3HVD9L9-FWxtP2dGySubns{  
stop();  
return true;  
}  
while(count < 1) {   
trackLine();  
}  
stop();  
pilot.rotate(90);  
while(count < 2) {  
trackLine();  
}  
stop();  
pilot.rotate(-90);  
while(count < 3) {   
trackLine();  
}  
stop();  
pilot.rotate(-90);  
while(count < 4) {  
trackLine();  
}  
stop();  
pilot.rotate(90);  
count=0;  
numberOfTurns = numberOfTurns + 4;  
System.out.println(numberOfTurns);  
trackAnEight(numberOfTurns);  
return false;  
}  
  
/\*\*  
\* helper method for Tracker; calculates distance from centerline, used as error by trackLine()  
\* @param left light reading  
\* @param right light reading  
\* @return distance  
\*/  
int CLDistance(int left, int right) {  
int error = left - right; // if positive to much to the left, if negative to much to the right  
return error;  
}  
  
public void stop()  
{  
pilot.stop();  
}  
  
/\*\*  
calibrates for line first, then background, then marker with left sensor. displays light sensor readings on LCD (percent)<br>  
Then displays left sensor (scaled value). Move left sensor over marker, press Enter to set marker value to sensorRead()/2  
\*/  
public void calibrate()  
{  
System.out.println("Calibrate Tracker");  
  
for (byte i = 0; i < 3; i++)  
{  
while (0 == Button.readButtons())//wait for press  
{  
LCD.drawInt(leftEye.getLightValue(), 4, 6, 1 + i);  
LCD.drawInt(rightEye.getLightValue(), 4, 12, 1 + i);  
if (i == 0)  
{  
LCD.drawString("LOW", 0, 1 + i);  
} else if (i == 1)  
{  
LCD.drawString("HIGH", 0, 1 + i);  
}   
}  
Sound.playTone(1000 + 200 \* i, 100);  
if (i == 0)  
{  
leftEye.calibrateLow();  
rightEye.calibrateLow();  
} else if (i == 1)  
{  
rightEye.calibrateHigh();  
leftEye.calibrateHigh();  
}   
while (0 < Button.readButtons())  
{  
Thread.yield();//button released  
}  
  
}  
while (0 == Button.readButtons())// while no press  
{  
int lval = leftEye.getLightValue();  
int rval = rightEye.getLightValue();  
LCD.drawInt(lval, 4, 0, 5);  
LCD.drawInt(rval, 4, 4, 5);  
LCD.drawInt(CLDistance(lval, rval), 4, 12, 5);  
LCD.refresh();  
}  
LCD.clear();  
}  
  
}