1. **Introduction**

* 1. Assignment and Due Date

Project 2, Milestone 3, Go to Destinations

* 1. Team Information

Team # 2

Team Members: Joong Hyuk An (Mechanical Designer and Code Designer), Mario Farren (Report Writer and Code Contributer), Victor Sandberg (Principal Program Coder)

Hours Spent: 1

* 1. Location of JavaDocs

\\steam.ieor.berkeley.edu\profiles\UGrads\mafarren\My Documents\workspace

1. **Performance Specifications**

The robot starts on (0,0) in the grid, facing the positive X direction. After using the buttons to introduce X and Y coordinates, the robot moves to its final destination where it completes its travel facing the positive X direction. From here a new set of coordinates can be programmed, and the robot will begin a new trip to this point and reach its destination successfully.

At each grid intersection, the robot makes a sound.

1. **Hardware Design**

The robot design utilized is a very standard one, since the instructions posted on the course page were used to build it. The light sensors are located in the front part of the robot 2 LU over the track. The two rubber wheels are in the middle and the third small support wheel is in the back of the robot. Most of the robot’s weight is located behind the middle wheels and is well supported by the layout of the wheels. This design allows the robot for fast starting and braking which benefits a quick completion of tasks.

Gear ratios and ultrasonic sensor are not currently being used.

1. **Experimental Work**

Milestone 3 did not require any experimental work of the type that was done in Milestone 1. Nonetheless the robot was tested continually during the code writing process to check that it went to the right coordinates and behaved as it should when it passed each marker. For example, in the beginning the trackline() method made the robot stop at each marker. After seeing this defect a quick fix to the code was done by delaying the stop depending on the number of markers the robot had passed.

1. **Problem Analysis**

Class: Milestone 3 (Grid Navigator)

Task 1- Go to Coordinates

Class: Tracker

Sub-Task 1.1- Track the line

Class: Differential Pilot

Sub-Task1.2- Calibrate sensors

Task 2- Get coordinates from user

Class: ButtonCounter

1. **Software design**

Public class “Milestone3 ” has private variables “tracker” that stores the tracking data and “currentX”, “ currentY” that stored the current x and y coordinates.

In the beginning we set the current x and y coordinates to (0,0).

Then with the method “toCoordinate()” it uses “ButtonCounter” to get the number of times the button was pressed and sets the coordinates. if x < 0 or y < 0 or x > 6 or y > 8 the screen of the robot prints out "Enter new coordinates! " to warn that the user pressed in an invalid coordinate.

if the coordinate pressed in is the exact same coordination as the current one it is on, then the robot points out the error by printing out “Your are already in this position. Enter new coordinates! ” when “ (x == currentX && y == currentY) “.

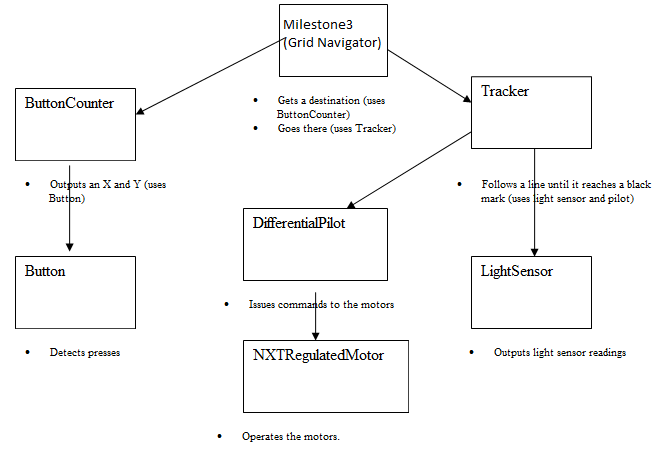
Through the method “toCoordinate(int x, int y)” it gets the difference of the current coordinates and pressed in destination to steer and move until the destination and current coordinates are the same.

Then tracker will stop with the “tracker.stop();”

6.1 Sources of programming ideas and also classes that are used but not coded or modified for this lab (except NXJ classes).

6.2 Class responsibilities and how they relate to tasks and subtasks of 5.1

6.3 For each class mention other classes used. Which supply data? How? Which are delegated to perform subtasks? Use diagrams and tables where appropriate



Each square represents the classes used to separate tasks into subtasks. The tester class is called Lejos 2, where an instance from the Milestone 3 is created and operated. The three top level classes were created by us: ButtonCounter, Tracker, and Milestone3.

6.4 Documentation generated from JavaDoc comments within the code:

The documentation is here: Path: milestone3/doc/

6.5 Use NetBeans to generate JavaDoc (html) documentation for your code. Do not use a period within your comments (everything after it will be suppressed from the summary pages). Include only the summary pages. You do not to include the JavaDocs, but your report must contain the path to the file in you project folder.

Path: milestone3/doc/

1. **Interesting/Challenging/Difficult**

Most interesting: It was really interesting how precisely and accurately the robot could maneuver through the grid with pretty good speed. Our team imagined many random and crazy things we could do and program into the robot for our amusement and joy of programing .

Challenging: It was a bit challenging and slightly confusing when trying to make the robot move according to the specification. We were a bit too busy coding to read the specifications thoroughly so after a few tips from our instructor we have change the code slightly to specifically follow the specifications.

Difficult: A difficult part of the project was trying to find the right speed and calibration in the beginning. Another slight difficult part was finding the format for reports and documentation on “UC Wise” website in the.

1. **Appendix**

public class Milestone3 {

private Tracker tracker;

private int currentX;

private int currentY;

public Milestone3(Tracker tracker) {

this.tracker = tracker;

currentX = 0;

currentY = 0;

}

public void toCoordinate() {

ButtonCounter buttonCounter = new ButtonCounter();

buttonCounter.count("Set Coordinates. Current position: ", currentX, currentY);

int x = buttonCounter.getLeftCount();

int y = buttonCounter.getRightCount();

if (x < 0 || y < 0 || x > 6 || y > 8 ) {

System.out.println("Enter new coordinates! ");

toCoordinate();

}

if (x == currentX && y == currentY) {

System.out.println("Your are already in this position. Enter new coordinates! ");

toCoordinate();

}

toCoordinate(x, y);

}

private void toCoordinate(int x, int y){

int diffX = x - currentX;

int diffY = y - currentY;

int turnDirectionY = 1;

int turnValue = 0;

if (diffX < 0) {

tracker.turn(2);

turnDirectionY = -1;

turnValue = turnValue + 2;

}

tracker.trackLine(Math.abs(diffX));

if (diffY < 0) {

tracker.turn(turnDirectionY);

turnValue = turnValue - turnDirectionY;

} else {

tracker.turn(turnDirectionY\*-1);

turnValue = turnValue - turnDirectionY\*-1;

}

tracker.trackLine(Math.abs(diffY));

tracker.turn(turnValue);

currentX = x;

currentY = y;

tracker.stop();

toCoordinate();

}

}