**Final Project - NXT**

===================

**1. Introduction**

**Name of Assignment:** Final Project Milestone 4 - Telerobotics

**Due Date:** 11/15/13

**Lab team number**: 5

**Team Members:**

Phuoc Nguyen, Khoa Tran (NXT Part)

Corey Short, Trevor Davenport (PC Part)

**Approximate number of person hours spent on the project**: 50

**Project Code:** Steam/TEAM/TEAM 5/Final Project (All Milestone)/Milestone4

**2. Performance Specification.**

Our robot meets all the specification for this milestone. It can be controlled by using a GUI from windows via Bluetooth. The robot will run to the desired location and update it location by using fixPosition(). Moreover, by integrating the new variance classes the robot is able to automatically update the location, including X, Y and the heading.

**3. Experimental Work**

**3.1. Description and Purpose:** In this whole telerobotic milestones, we need to perform a lot of task in order to find the accurate location of the robot. This milestone is a combination result/work of all the previous milestones. We need to use the LightSensor to read the accurate bearings of those 2 beacons, UltraSonicSensor to read the distance to the 2 walls. Also, the measurement of the wheel diameter and track width play an important role in the accuracy of the rotate/turn of the robot. Our job is to make sure everything is as precise as possible to prevent error from building up.

**3.2. List of Data:** We used the data from Milestone 3 to calculate the variance.

**4. Problem Analysis:**

In order to make the robot more accurate in finding it own location, we use the 2 new classes to calculate the Variance. Our job was to used the same data from the milestone 3 to calculate the variance. Unfortunately, even though we had a small amount of standard deviation, we were off a lot from the original value, especially X. That problem makes our calculation for the variance more difficult as it is not what we wanted. We need to redo the calculation a lot of time in order to come up with the new/accurate value for the Variance class.

To do that, we try to adjust the location of the robot at many different locations and record the new data at 4 headings. Moreover, we slow down the rotation speed for the lightReading and Ultrasonic reading. More likely, when the speed is slower then we can achieve more accurate result. However, that's the problem for the Final Demonstration because we just have only 45 mins in order to finish the mission.

We also found out that whenever the robot reads the values in front of it (0 degree to the heading of the NXT brick), it is always more accurate then it reads the value from its left or right. The value is even getting worse when it reads from it back. By slowing down the rotation speed and adding more delay, we could somehow eliminate this problem a little bit.

**5. Software Design:**

***Milestone 4A:***

**\*\*Mission Control\*\*:** Set up the commands setPose, goTo, and stop. Also, set up a MouseListener where the mouse coordinates are relayed to the class based on position. Another task of this Milestone was to set up the GUI that will be used throughout this entire project.

**\*\*Classes\*\*:**

PC SIDE: CommListener, GNC, GridControlCommunicator, MessageType, OffScreenDrawing.

NXT SIDE: Locator, Scanner, Message, MessageType, Controller, Communicator, CommListener

**\*\*Data Flow for goTo()/STOP Call\*\***

1. GUI is built. (CommListener class)

2. GUI adds GoToButtonActionListener().

3. GoToButtonActionListener() calls sendMove().

4. sendMove() parses the input text fields to get X and Y coordinates according to MouseLocation.

5. sendMove() uses a GridControllerNavigator object to call controller.sendDestination(x,y).

6. GridControllerNav uses two DataStream Objects (One Input and One Output).

7. In sendDestination: dataOut.writeInt(MessageType.MOVE.ordinal());

dataOut.flush();

dataOut.writeFloat(x);

dataOut.flush();

dataOut.writeFloat(y);

dataOut.flush();

- PC Relays type of message above (enum GOTO or STOP) via Bluetooth Connection object btc.

- MessageType arrives at NXT Side via Communicator Class.

- Communicator class uses the Multi Threading Method run() to decode MessageType().

- run() uses a switch statement to determine MessageType() and calls updateMessage() accordingly.

- Control is returned the the Controller class and updateMessage runs to determine what kind of Message is passed.

- If Message is STOP, empty entire ArrayList<Message>. navigator.stop(). And navigator.clearPath().

- If Messsage != STOP, append the CurrentMessage to ArrayList<Message> inbox.

- After performing action, NXT uses Locator class to determine current Location.

- Current Location is relayed back to PC.

***Milestone 4B:***

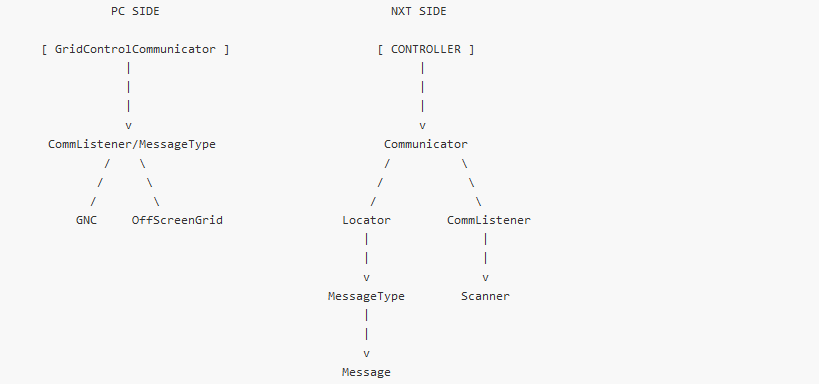
**\*\*Mission Control/Task Analysis\*\*:** Develop our Milestone 4A code to incorporate the three following instructions: Rotate, travel, and Fix Position.

**\*\*Classes\*\*:**

PC SIDE: CommListener, GNC, GridControlCommunicator, MessageType, OffScreenDrawing.

NXT SIDE: Locator, Scanner, Message, MessageType, Controller, Communicator, CommListener

**\*\*Data Flow/Relationships between classes\*\*:**



**\*\*Data Flow Between PC and NXT\*\*:**

PC SIDE handles setting up the gui with the GridControlCommunicator class as well as the OffScreenDrawing.

Once the GUI is setup, NXT SIDE Communicator class is able to establish a connection by using DataInput and DataOutputStreams. Once the connection is established, CommListener is implemented to determine any type of Mouse or ActionListener that the user has the ability to change based on our MessageType ENUM class. The MessageType ENUM class contains 9 enumators that allow the robot to manuever and perform tasks accordingly. Based on which ENUM is being implemented, the PC SIDE sendsMessage() to the NXT side to perform said action. This is all possible due to the dataOutputStreams we referred to previously. As the robot is running, the call to updateMessage() occurs whenever a new action or new event is produced. After performing the actions, NXT relays its position to the Locator class to determine currentLocation which is in turn relayed back to the PC.

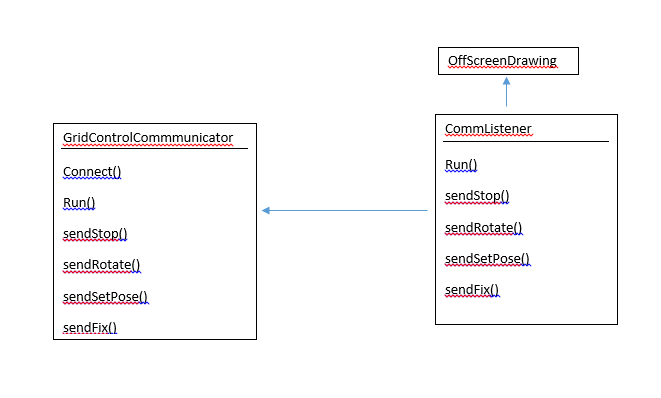
**\*\*Table of Fix Pose vs Reported Position\*\*:**

**\*\*Screenshot of Route Traveled\*\*:**

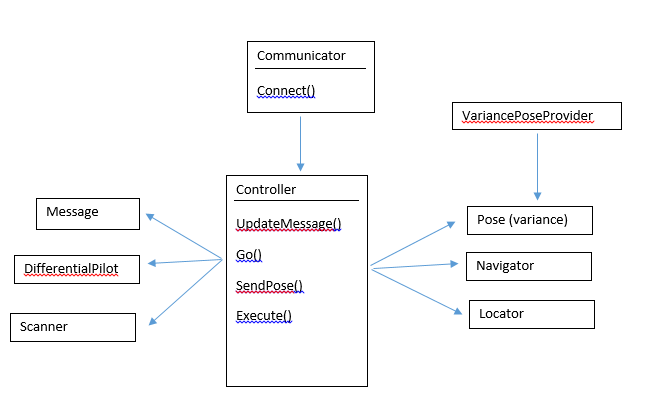
[insert image from coreys comp]

**Class Dependency:**

**PC Diagram**

****

**NXT Diagram**

****

**6. Interesting/Challenging/Difficult Part.**

The most interesting part of this project is that we can program a robot to run by using Bluetooth and java GUI. It is more interesting to see the robot to update the location, and later automatically update the location by using the variance classes.

The challenging part of this project is to design a communication part between the robot and the PC. There has to be a way to manage the incoming/outgoing message, otherwise we will miss all the messages and may mess up the command to the robot. By using queue, we are able to control the robot and save/perform every task as wanted.

The difficult part of this project is adding the variance to the code, for somehow the data that we had and calculated the variance is not exactly as what we want. We need to recalculate everything and test everything out before putting the correct number to the code. Unfortunately, this process took a lot of time because our robot is not as accurate as before, the distance to the wall and the light values are not accurate. We somehow figured out the problem is that when the battery is nearly drained out, it will make every information less accurate.