University Of The South Pacific

EE300

Reactive Control and Obstacle Avoidance of a Lego NXT Robot

Supervisor : Dr. Praneel Chand

Authors : Avneet Prasad – s11030205

Liam Kwong-Wah – s11040917

11

Acknowledgement

We would like to thank the following people for their unparalleled assistance and guidance throughout the semester towards the successful completion of this project.

First and foremost, we would like to thank Dr. Praneel Chand for not only being a supervisor for this project but his direction, patience and continuous challenging encouraged and motivated us immensely.

Secondly, we would like to thank Binal, Krishneel and Sirilo, the laboratory supervisors who were always there to make the lab available to us.

Lastly, we would like to show our appreciation to our colleague, Ron Jacquier, for his help and keen insight.

Declaration Of Originality

We, Avneet Prasad and Liam Kwong-Wah, hereby assert that the work presented in this report is our own and is in no way plagiarised. Any published or unpublished works used in the gathering of information have been appropriately identified and sourced.

Avneet Prasad (S11030205)

Liam Kwong-Wah (S11040917)

Abstract

Robotics is an exciting field with advancements being made on a continual basis. The question of how an otherwise inanimate entity can be made to move and behave in a certain way, without physical assistance, is the major hindrance that afflicts those wanting to work with mobile robots. As advancements in robotics have been made, various approaches to mobile robot control have been developed each having their own pros and cons. This project delves into the method of reactive control to manage a system with the sole purpose of being able to detect and avoid obstacles in most unstructured environments. The navigation system design is one originally utilised by the supervisor of this project, Dr. Praneel Chand and his counterpart Dale A. Carnegie (Chand and Carnegie, 2005), in a previous undertaking that saw them employ the use of reactive and deliberative control to influence the behaviour of another mobile robot. Experiments to tune the various parameters of the codes were conducted along with several simulation tests. Physical, real world testing was also carried to gauge the robot's ability to detect and avoid obstacles.

Table of Content

List of Figures

List of Tables

Location of Added Codes

This section describes the location of all the added codes/functions for easier access if the occasion arises to modify these codes.

Function/m-file	Added Function Name	Line #
Gui_main	ConnectBT	64
	MotorFeedback	747
	gui_move_pose	755
	predict_pose	756
Sim_obstacle Detect	SensorRange	50
Sim_robot_motion	MotorSettings	21

Introduction

The development of *mobile robots* is an area of interest that generates much enthusiasm and fascination not only in the world of technology but with almost every person in general no matter what field of expertise. Mobile robots can be found in a variety of settings including industry, military and have made tremendous contributions in space expeditions for the National Aeronautics and Space Administration (NASA). As the name suggests, mobile robots are not fixed to any one point and are capable of moving around in an environment. Achieving this control in a robot, that is, to be able to move from one location to another is an important concern and is in no way a simple feat. There are a number of ways to implement a robot's navigation system and it must be noted that the type of control runs in conjunction with the type of application. A few classes of navigation systems include *deliberative control, hybrid control, behaviour based control* and *reactive control* which is the type of control selected for this project.

Reactive Control

Reactive control has been described as a "Don't think, react!" method and is a system that involves closely combining the robot's sensory inputs with its outputs to achieve a quick and effective response while traversing an amorphous environment. Like each of the other approaches, reactive control has its disadvantages, that is, in order to achieve quick response times, these robot types do not have memory, hence, they do not have representations of their environment and therefore the robot is incapable of learning over time as it encounters the various obstacles. It must be noted that as a result of these shortcomings the more complex the environment is the less proficient the robot's navigational system performance.

This project concentrates primarily on the design and construction of a suitable LEGO NXT robot that utilises reactive control to achieve obstacle detection and avoidance in a variety of worlds. The successful navigation of the robot is also tied to a MATLAB run algorithm provided by the supervisor of this project *Dr. Praneel Chand* which incorporates the various parameters of the robot and in turn dictates its overall motion.

Related Work

As previously mentioned, the algorithm used in this project was provided by the project supervisor, Dr. Praneel Chand and it is therefore the work that he and others (Chand and Carnegie, 2005) have carried out, towards the better understanding of mobile robot navigation systems, that form the basis of our project. To achieve the proper control needed for this project, the design is primarily based upon a *modified dynamic window approach and polar histogram technique*. The *A* algorithm* which was also used by Chand and Carnegie (2005) is mainly for path planning which is not appropriate for reactive control.

Modified Dynamic Window Approach with the Polar Histogram Technique:

The dynamic window approach is an approach established by Dieter Fox, Wolfram Burgard and Sebastian Thrun (1997) and further developed by Brock and Khatib (1999) which implements real time collision avoidance. This strategy is uniquely developed according to the specifications or characteristics of the robot, for example, the size and shape of the robot, the type of drive (differential, tricycle), robot radius and especially the kinematics of the robot that is, current velocity, angular velocity, linear acceleration and angular acceleration. Furthermore, the dynamic window approach (Fox et al., 1997) was initially tested on a synchronous drive where the linear and angular accelerations were independently altered. This project, however, involves a differential drive robot meaning the linear and angular accelerations are dependent on the current velocities and constraints of the robot.

The polar histogram technique is likened to the Vector Field Histogram method (VFH) developed by Johann Borenstein and Iwan Ulrich (1991). Much like the dynamic window approach (Fox et al., 1997), the VFH technique (Borenstein and Ulrich, 1998) also incorporates the robots features and then returns a steering command. One can see that both the dynamic window approach (Fox et al., 1997) and the VFH technique (Borenstein and Ulrich, 1998) utilise the same parameters allowing them to work hand-in-hand but the VFH allows for a “statistical representation” of the environment through a histogram grid in order to take into account sensor uncertainties or errors occurred in the modelling stage.

Using the aforementioned techniques, the general process of attaining robot velocities in order to get to the desired location while being able to maneuver around obstacles is as follows;

A target heading is input to the algorithm in the form of x and y coordinates. This is then changed into a modified target heading in the form of theta (θ). This modified heading is then translated into linear and angular velocities to the robot.

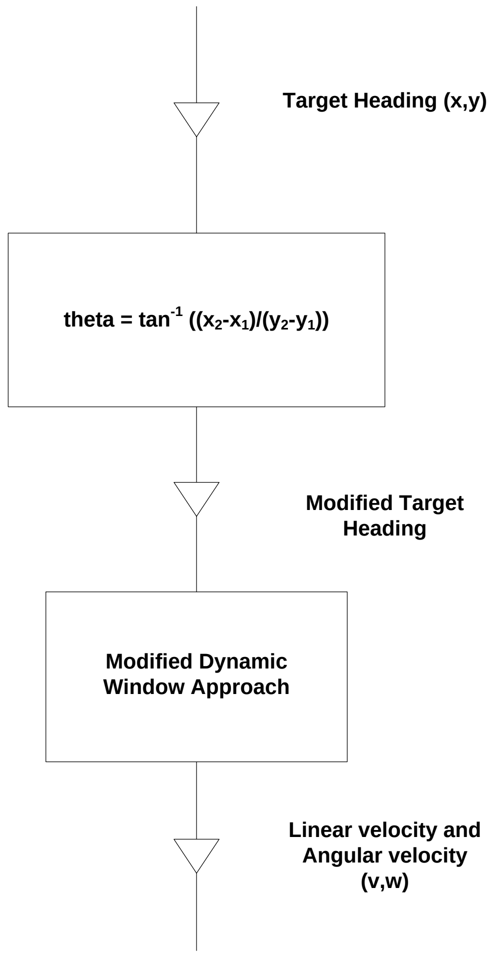

Target Heading (x,y)
theta = $\tan^{-1}((x_2-x_1)/(y_2-y_1))$
Modified Target Heading
Modified Dynamic Window Approach
Linear velocity and Angular velocity (v,w)

Figure 1 - Generalised Flow Chart of Process for Initial Robot Motion

Current System Overview

To begin with, it must be made known that the aims of this project are two-fold. Firstly, an appropriate design for a robot platform was to be made and constructed taking into account the sensor placements and other onboard components. Secondly, reactive control is to be established with a suitable Graphical User Interface (GUI) to be implemented.

This section, therefore, deals with the particulars of the current robot design and especially navigation system controlling the differential drive robot chosen for this project.

Phase 1 - Design and Assembly

The first half of the semester (7 weeks) was allocated towards the construction of a suitable robot platform. During the course of these 7 weeks a number of prototypes were built and tested for *Speed, Stability, Maneuverability and Size.* The various robotic platforms were tested against an NXT program (3-button program) found online that allowed for forward and turning motion (The main algorithm used in this project had not yet been supplied).

<u>Prototype 1:</u>

Drive Type – Tricycle (Tribot)
Speed – Unsatisfactory (Hindered by Caster Wheel)
Stability – Unstable
Maneuverability – Unsatisfactory (Hindered by Caster Wheel)
Size – Too small (Unable to house NXT Intelligent Brick let alone IR sensors)

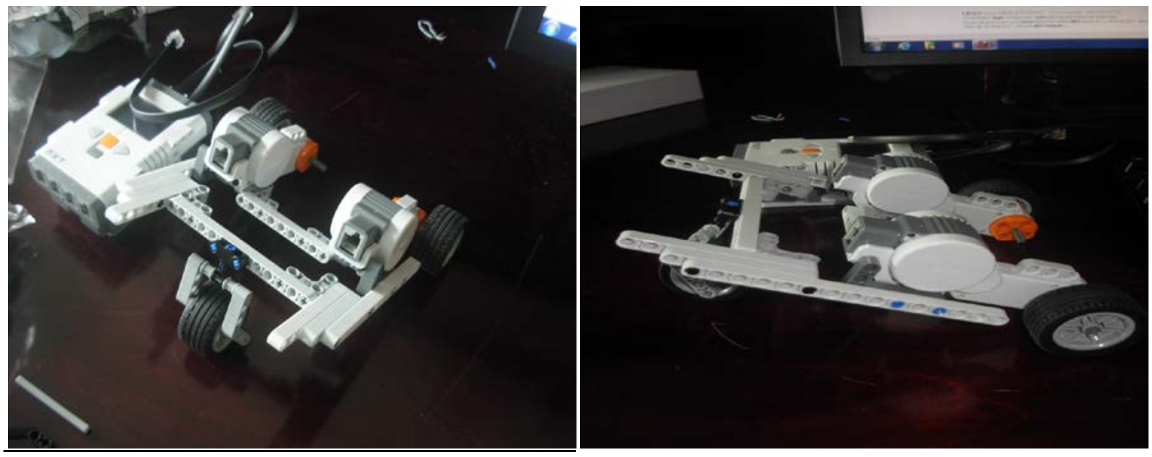


Figure 2 - Tribot Front and Side Views

Prototype 2:

Drive Type – Differential Drive (Gear System)
Speed – Satisfactory
Stability – Very Stable
Maneuverability – Good (Able to turn in place)
Size – Too small (Unable to house necessary onboard components)

This system utilises a gear system and this caused two problems concerning construction. First, there were not enough gears to equalise each side of the robot (gear ratio was still 1:1). Second, using the gears disallowed the expansion of the platform in order to house extra components.

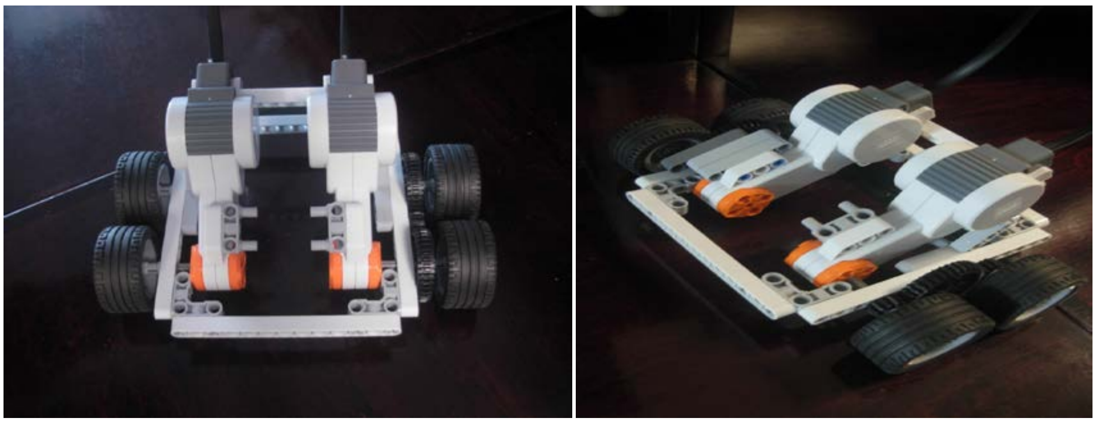


Figure 3 - Differential Drive (Gear System) Front and Side Views

Prototype 3:

Drive Type – Differential Drive (Continuous Tracks/Caterpiller Tracks)
Speed – Good
Stability – Very Stable
Maneuverability – Excellent
Size – Small (Better though due to expandable base)

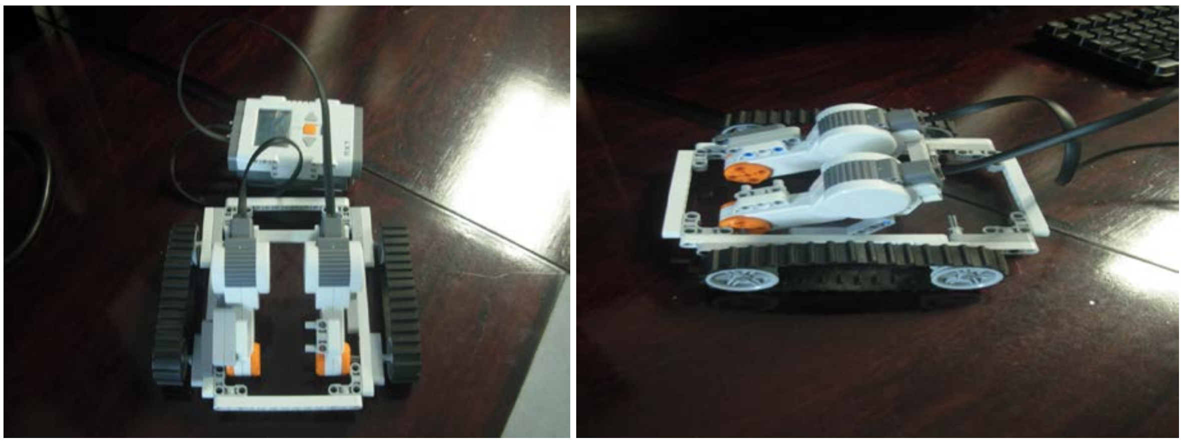


Figure 4 - Differential Drive (Continuous Tracks) Top-Front and Side View

Prototype 4 – Final Design:

Drive Type – Differential Drive (Continuous Tracks/Caterpillar Tracks)
Speed – Good
Stability – Very Stable
Maneuverability – Excellent
Size – Satisfactory (Best possible size due to LEGO piece constraints and able to house all onboard components)

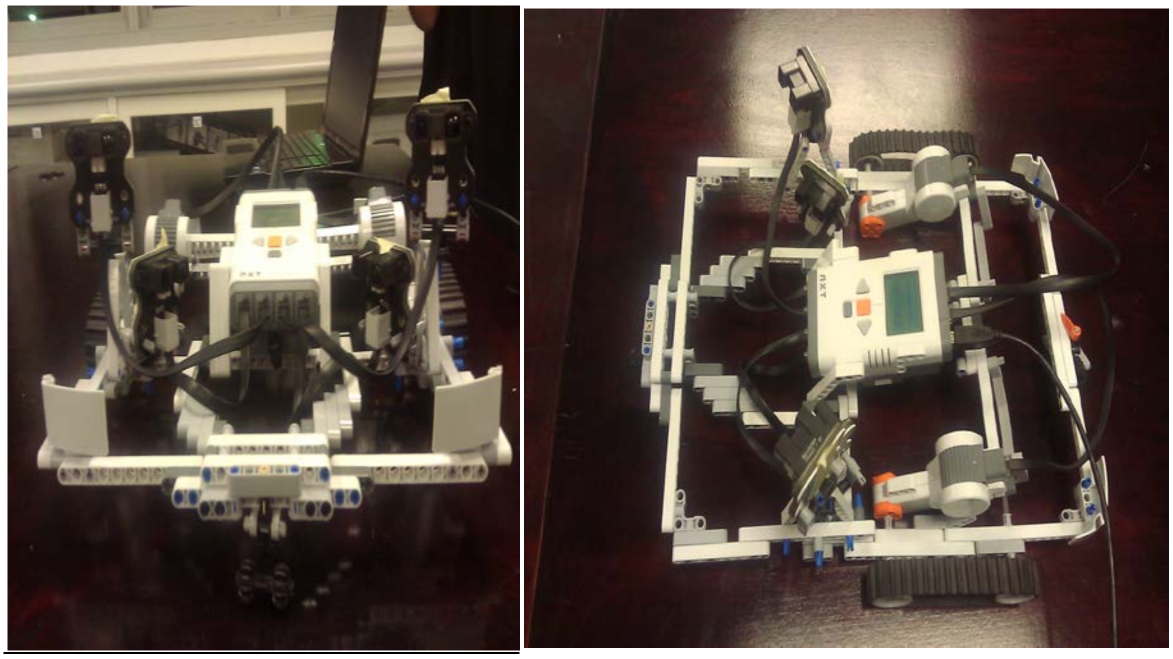


Figure 5 - Differential Drive (Continuous Tracks) Front and Top-Side View

It must be noted that the sensors could not be mounted onto the platform due to its late arrival (Sensors were received in the 12th week). Figure 6 below indicates the required 6 sensor placements but with the current system that can only implement forward motion (No rear sensors). (Note side sensors not used. Refer to IR Sensors in Components Chapter)

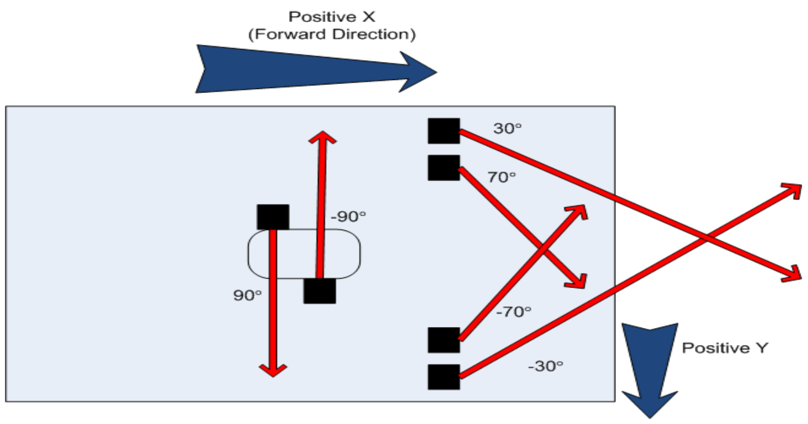


Figure 6 - Current Sensor Placements

Phase 2 - Tuning Parameters

Communication:

In order for the robot to receive an initial target heading or the new wheel velocities, wireless Bluetooth communication is required. This is achieved by MATLAB's RWTH Mindstorms NXT Toolbox specifically designed to control NXT's Intelligent Brick for LEGO robots. It should be noted however that Bluetooth communication is not recommended for real-time robot control due to the higher latency when compared with a USB connection and this is especially evident when implementing a complex program.

Kinematics:

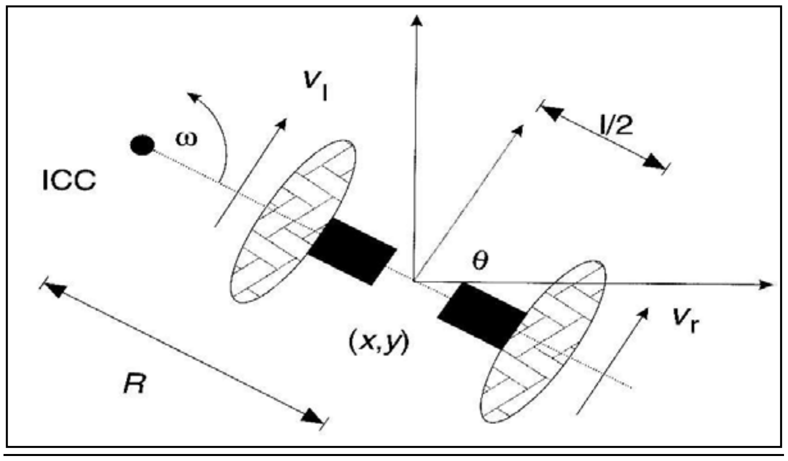


Figure 7 - Differential Drive Kinematics (Dudek and Jenkin, *Computational Principles of Mobile Robotics*)

In order for the robot to travel in an orderly manner without colliding with obstacles the robot's *kinematics* needs to be properly tuned. This is absolutely imperative since the modified dynamic window approach and the polar histogram technique both comprise of utilising the current velocities and accelerations of the robot. Therefore, since the robot is of a differential drive type, the system needs to know the left and right wheel velocities which would provide an indication of its current status, that is, whether it was turning left ($v_r > v_l$), turning right ($v_l > v_r$) or moving in a linear fashion ($v_r = v_l$).

The kinematic equations used are as follows;

$$v_r = \omega (R + L/2)$$

$$v_l = \omega (R - L/2)$$

(Where: v_r = Right Wheel Velocity, v_l = Left Wheel Velocity, ω = Angular Velocity, R = distance between the Instantaneous Centre of Curvature (ICC) to the mid-point between the wheels and L = distance between the centres of the two wheels.)

Hence, equating the equations yields the following values at any instance in time;

$$\omega = (v_r - v_l)/L \text{ and } R = \tfrac{1}{2}(v_l + v_r)/(v_r - v_l)$$

Moreover, an aspect that needs to be dually noted is the fact that one cannot directly acquire the left and right wheel velocities using the RWTH Mindstorms NXT Toolbox. However one is able to ascertain the power levels of each motor and this can in turn be translated into the required velocities through experimentation (Refer to Experimentation and Results – Maximum Linear Velocity).

Dynamics:

The dynamic constraints of maximum linear wheel acceleration (deceleration) and angular acceleration (deceleration) are tied to the linear and angular velocities and are therefore calculated using these kinematic constraints.

Equations applied are:

$$a_t = (V_f - V_i)/t \text{ and } a_t = r\alpha \text{ therefore } \alpha = (1/r)\,(V_f - V_i)/t$$

(Where: a_t = tangential linear acceleration of the wheel, r = radius of the wheel, α = angular acceleration, V_f = final linear velocity, V_i = initial linear velocity, t = time.)

Safety Margins:

The *safety margins* of the system are basically introduced to the perimeter of the robot to allow for deceleration, stopping or reversing when encountering an object. In broad terms, the kinematics or more specifically the current velocity of the robot is directly tied to the safety margin and together they play an essential role in obstacle avoidance. Non-circular robots such as the one used in this project are able to have autonomous values for safety margins for each of the sides. Simply put, when the distance to an object infringes the safety margin (distance to object $\leq$ Safety Margin) a flag is raised telling the algorithm to decrease the current velocity. Furthermore, the safety margin may increase proportionally from its minimum value due to an increase in the velocity of the robot and the introduction of a proportionality constant, k (measured in seconds). An example of how the velocity and safety margin work in unison is as follows. If the robot approaches an obstacle at a high velocity, the chances of it being able to stop in time are drastically low. However, incorporating a safety margin which grows as robot speed increases allows for the system to understand when the robot needs to slow down (or stop) and maneuver out of the obstacle's way, well before any collisions can occur.

The safety margin at any point in time is given by:

Safety Margin = Safety Margin Minimum + (Proportionality Constant *Current Velocity)

Process of Selecting an Optimal Velocity Pair

The basic scheme of attaining the optimal velocity for the robot to move at while utilising the sensory inputs is as follows;

To begin with, the user inputs a *target location* for the robot to make its way to. From the target heading, suitable values for the initial *linear* and *angular velocity* pair (v,w) are chosen (Refer to Figure 1 of Introduction). It is assumed that the robot will start with its maximum linear and angular velocities taking into account that there are no obstacles in its line of sight upon start-off. These values of linear velocity and angular velocity, along with the sensory inputs are then taken as inputs to the reactive control navigation system. A new set (or the same set) of v and w, depending on the sensory inputs, are given as outputs and it is this set of data that is passed through the *kinematic equations* stage (Refer to Kinematics section above) in order to find the *left* and *right* wheel *velocities* (V_L and V_R) of the robot. These values of V_L and V_R are then fed to the robot to dictate its heading. As the robot travels on its course the feedback from the sensors is then tested against the current velocities of the robot and adjustments are then made depending on the feedback data. Note, the velocity (V_L and V_R) of the robot needs to be converted back into a form that the algorithm understands, that is, in the form of v and w, hence, the robot linear velocity is passed through the *inverse kinematics* stage. Figure 7 shows the process described above.

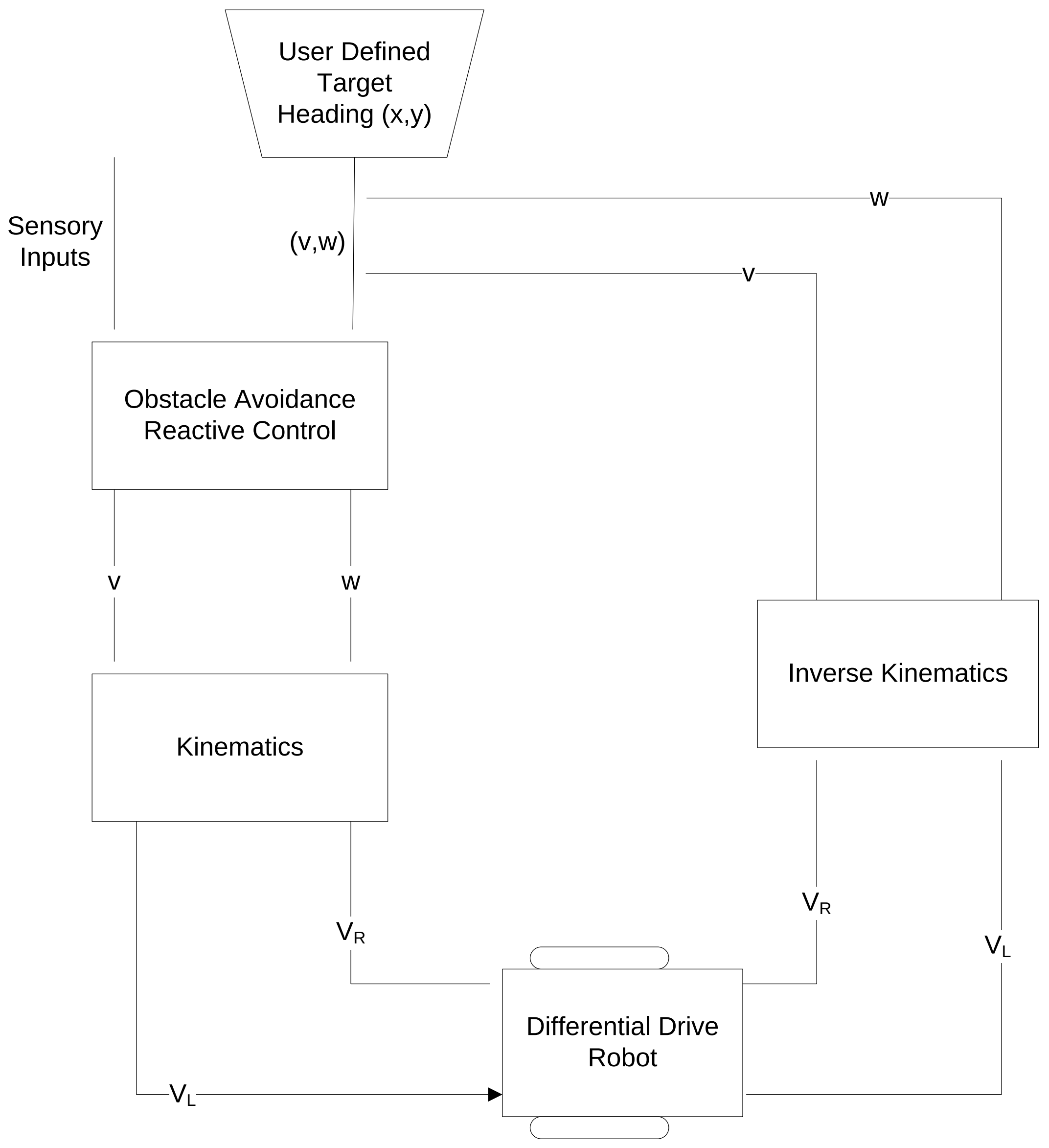


Figure 8 - Block Diagram Showing Simplified Process of Obtaining Velocity Pairs for Robot Motion

Velocity Pairing:

A problem that arose while acquiring the maximum velocity was that the velocities were different for both motors while running at same power level. An experiment, to corroborate this inconsistency, was conducted to determine the velocities at their individual power levels. Moreover, this dilemma would result in the robot not being able to travel in a straight line, that is, it would curve slightly in one direction. It was also found that the motor started rotating at a power level of 15% and this was assumed to be due to the load.

Therefore, to equate the speed and power relationship, a solution was devised to pair the individual wheel velocities according to their corresponding power levels.

The internal motor encoder was used and the velocity was recorded starting from 15% power to 100% power at increments of 1%. A while loop was incorporated to implement this in MATLAB. The code is shown below.

```
Power = 15;
j = 1;
while (Power <=100)
    SetMotor(MOTOR_B);
    SetPower(Power);
    SendMotorSettings();
    SetMotor(MOTOR_C)
    SetPower(Power);
    SendMotorSettings();
  pause(2);
  i=0;
  a=0;
  c=0;
  while(i~=3)
  b = MotorFeedback();
  a = a + b(1);
  c=c+b(2);

  i = i+1;
  end


  P2V(j,:) = [Power a/i c/i]
  j = j+1;
  Power = Power + 1;
  StopMotor('all','off');
end
```

The results of this experiment was logged into an array which showed the left wheel, right wheel velocities and their corresponding power level.

```
                     Power        Vr         Vl

Vel2PowerArray=     [15.0000     0.0013     0.0151;
                     16.0000     0.0024     0.0194;
                     17.0000     0.0140     0.0221;
                     18.0000     0.0167     0.0241;
                     19.0000     0.0174     0.0253;
                     20.0000     0.0217     0.0288;
                     21.0000     0.0233     0.0309;
                     22.0000     0.0249     0.0329;
                     23.0000     0.0265     0.0351;
                     24.0000     0.0300     0.0385;
                     25.0000     0.0318     0.0407;
                     26.0000     0.0340     0.0425;
                     27.0000     0.0357     0.0444;
                     28.0000     0.0391     0.0480;
                     29.0000     0.0408     0.0501;
                     30.0000     0.0426     0.0521;
                     31.0000     0.0447     0.0539;
                     32.0000     0.0485     0.0582;
                     33.0000     0.0497     0.0601;
                     34.0000     0.0521     0.0621;
                     35.0000     0.0535     0.0636;
                     36.0000     0.0574     0.0676;
                     37.0000     0.0589     0.0699;
                     38.0000     0.0606     0.0717;
                     39.0000     0.0625     0.0732;
                     40.0000     0.0662     0.0774;
                     41.0000     0.0675     0.0796;
                     42.0000     0.0696     0.0814;
                     43.0000     0.0710     0.0831;
                     44.0000     0.0749     0.0868;
                     45.0000     0.0774     0.0889;
                     46.0000     0.0790     0.0908;
                     47.0000     0.0812     0.0926;
                     48.0000     0.0847     0.0965;
                     49.0000     0.0865     0.0983;
                     50.0000     0.0888     0.1002;
                     51.0000     0.0907     0.1024;
                     52.0000     0.0943     0.1062;
                     53.0000     0.0958     0.1081;
                     54.0000     0.0986     0.1102;
                     55.0000     0.1003     0.1121;
                     56.0000     0.1047     0.1161;
                     57.0000     0.1070     0.1179;
                     58.0000     0.1084     0.1200;
                     59.0000     0.1102     0.1219;
                     60.0000     0.1144     0.1259;
                     61.0000     0.1170     0.1279;
                     62.0000     0.1190     0.1299;
                     63.0000     0.1214     0.1320;
                     64.0000     0.1251     0.1360;
                     65.0000     0.1274     0.1379;
                     66.0000     0.1298     0.1397
                     67.0000     0.1315     0.1418;
                     68.0000     0.1354     0.1456;
                     69.0000     0.1375     0.1478;
                     70.0000     0.1394     0.1497;
                     71.0000     0.1413     0.1517;
                     72.0000     0.1455     0.1554;
                     73.0000     0.1474     0.1574;
                     74.0000     0.1497     0.1594;
                     75.0000     0.1518     0.1612;
                     76.0000     0.1551     0.1650;
```

```
  77.0000    0.1572    0.1668;
  78.0000    0.1592    0.1687;
  79.0000    0.1613    0.1706;
  80.0000    0.1657    0.1744;
  81.0000    0.1674    0.1763;
  82.0000    0.1696    0.1784;
  83.0000    0.1715    0.1802;
  84.0000    0.1755    0.1840;
  85.0000    0.1775    0.1861;
  86.0000    0.1794    0.1880;
  87.0000    0.1814    0.1900;
  88.0000    0.1855    0.1937;
  89.0000    0.1874    0.1953;
  90.0000    0.1893    0.1974;
  91.0000    0.1913    0.1994;
  92.0000    0.1954    0.2034;
  93.0000    0.1973    0.2053;
  94.0000    0.1993    0.2073;
  95.0000    0.2014    0.2093;
  96.0000    0.2053    0.2133;
  97.0000    0.2070    0.2153;
  98.0000    0.2088    0.2172;
  99.0000    0.2107    0.2190;
 100.0000    0.2168    0.2252];
```

As seen from the logged data, the velocities of both wheels are different at the same power level. This pairing method would solve the differences in speed. For example, if a linear velocity of .1060m/s and an angular velocity of 0 rad/s were applied, the corresponding speed for the right wheel would be .1070m/s and power would be 57%. And speed for the left wheel would be 0.1062m/s and power would be 52%. This would prevent the robot from breaking its linear path since both wheel velocities are now the same. For reverse direction (robot moving in reverse), the same pairing method was used but with the signs changed to negative.

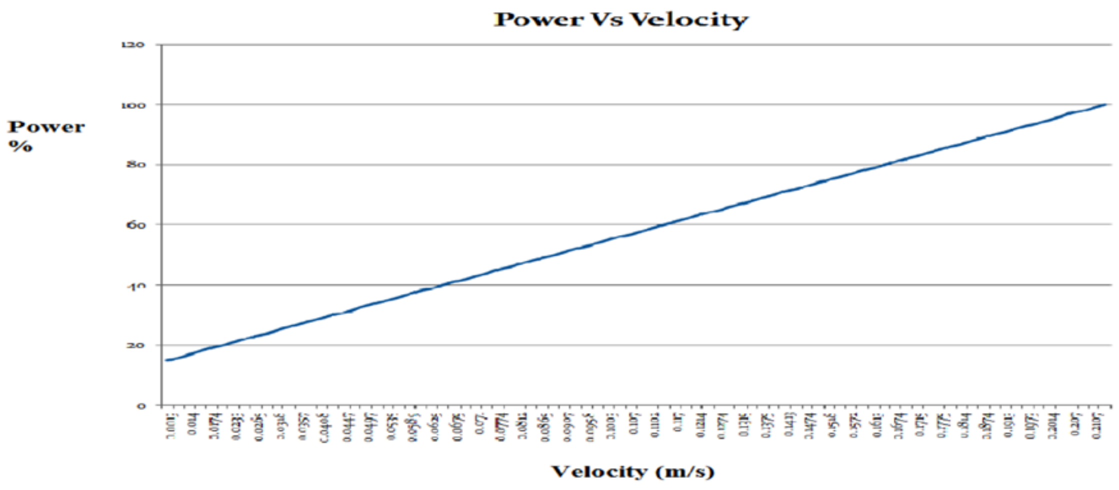


Figure 9 - Graph of Power versus Velocity

Components

Lego NXT robot

The Lego Mindstorm NXT is a programmable robotics kit released by Lego. The brain of this robot is a brick shaped computer called the NXT Intelligent Brick. It can accommodate up to 4 sensors and three motors. Communication between the Brick and the sensors/motors are done via RJ12 cables, which are similar to the RJ11 phone cords but incompatible with the Brick. The Brick utilizes a 100x64 LCD screen and has four buttons for navigation of the user interface which uses hierarchical menus. 6 AA batteries are used to provide power to the whole system. Communication can be done by USB or Bluetooth.

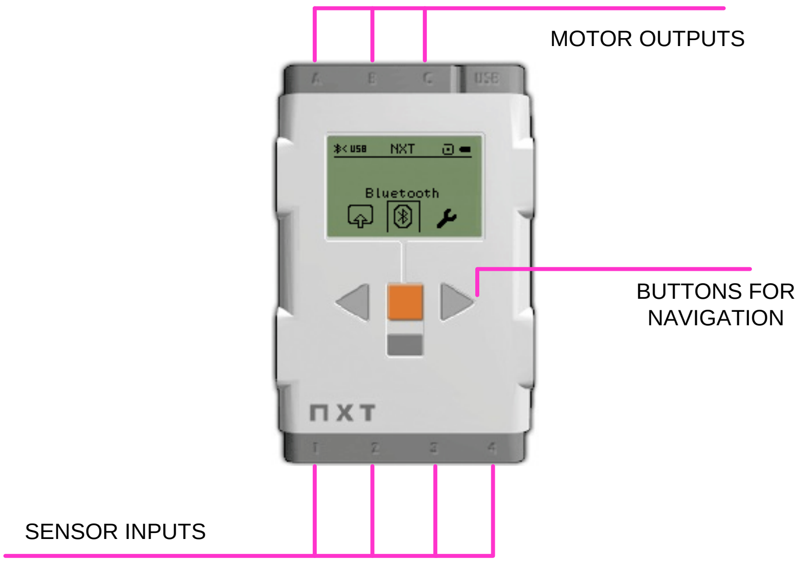


Figure 10 – NXT Brick Overview

NXT MOTOR

The Servo Motor has a built-in rotation sensor that measures speed and distance. This allows for precise complete motor control within one degree of accuracy.

Figure 11- NXT Servo Motor (http://mindstorms.lego.com)

NXT Motor Internals

in order to get accurate speed information back from the motor, its encoder information had to be made available. This could not be done in the lab without breaking apart the motor. So information about the encoder was gathered from the internet.

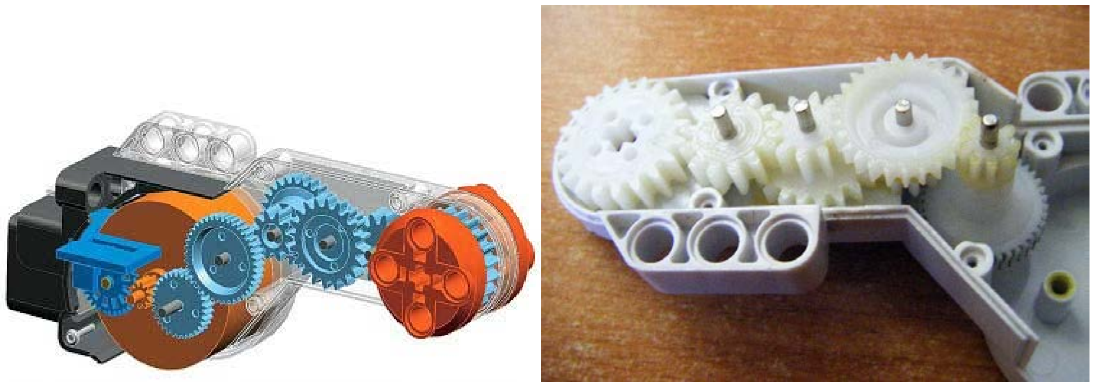


Figure 12- NXT Motor Exploded View (*http://www.philohome.com/nxtmotor/nxtmotor.htm*)

The above figure shows an actual Lego NXT motor internals.

Gear Ratio, from motor to output shaft.

10:30:40	= 1:4
9:27	= 1:3
10:20	= 1:2
10:13:20	= 1:2
Overall	1:48

From this the encoder information was found.

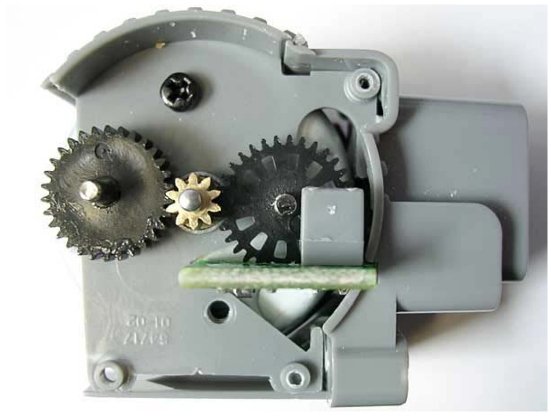


Figure 13-NXT Motor Exploded View Showing Encoder and optical fork (*http://www.philohome.com/nxtmotor/nxtmotor.htm*)

There are 12 slits in encoder, motor to encoder gear reduction is 10:32. So for 1 turn of output hub, encoders turn 48*10/32 =15 turns, and 15*12=180 slits. Using both sides there is 360 slits.

NXT SENSORS

Ultrasonic Sensor

This sensor allows the Lego NXT robot to detect obstacles. Uses ultrasound, work on a similar principle as radar or sonar. These generate high frequency sound waves and use its echo to calculate the distance from an obstacle.

Figure 14- NXT Ultrasonic Sensor (*http://www.legoeducation.us*)

This sensor was used briefly in this project after the initial unsuccessful attempt at reading information from the IR sensors. However, testing showed that the Ultrasonic sensor was not a suitable applicant for this project due to its broad range. Further research suggested that the uses of multiple ultrasonic sensors may hinder with distance readings. A possible fix for this is mentioned in the Problems and solutions section if an application demands use of multiple ultrasonic sensors.

Touch Sensor

Another sensor which comes with the Lego NXT package is the Touch sensor. It is a very basic sensor which utilizes a push button and is either on or off. Applications can use the sensors state to make decisions. This sensor was not used for this project.

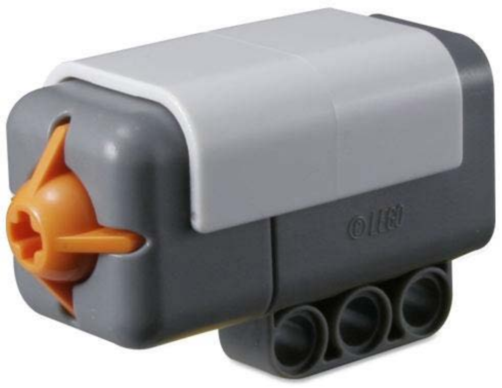


Figure 15- NXT Touch Sensor (*http://www.legoeducation.us*)

Light Sensor

The light sensor detects the light intensity in one direction. Includes an LED to illuminate an object. This sensor is also able to read the reflection from the infrared transmitter.

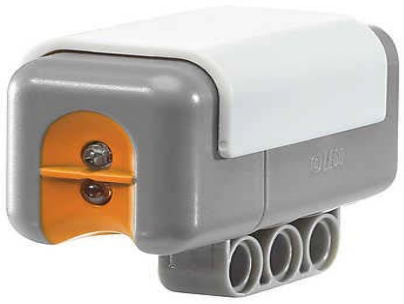


Figure 16- NXT Light Sensor (*http://www.legoeducation.us*)

Sound Sensor

The sound sensor measures volume level on a scale of 0 to 100, 100 being very loud, 0 being completely silent. This sophisticated sensor provides stimulus for directing a robot's actions.

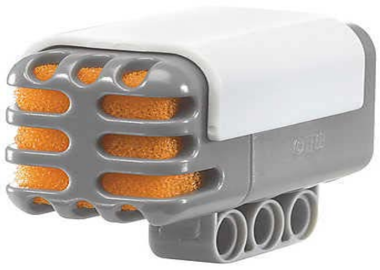


Figure 27- NXT Sound Sensor (*http://www.legoeducation.us*)

IR SENSORS

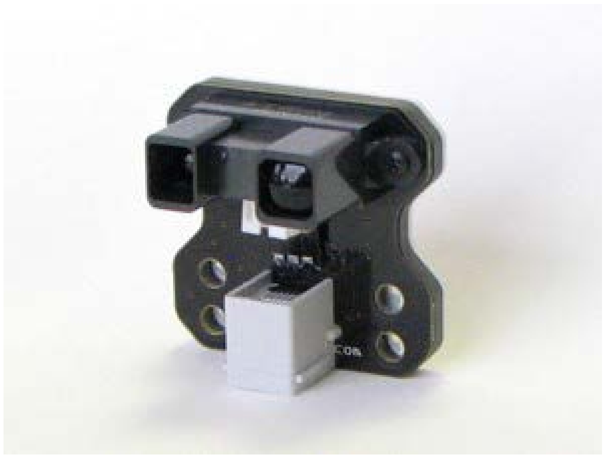


Figure 18- Dist-Nx-V3 Infrared Sensor (*image taken from the datasheet*)

The sensors opted for use in this project is the Dist-Nx-V3 Long Range Infra red sensor. This sensor has a range of 150cm with a minimum range of 30cm. However testing showed that the sensor could be operable within a minimum distance of 15-20cm. the sensor uses a SHARP GP2Y0A02YK sensor and provides accurate readings in millimeters. It emits an IR light which is then reflected from the obstacle. The angle of the received IR light is measured and used to calculate the distance to the obstacle.

Note: Only four Sensors were used for this project, reason for this is discussed in the Problems Encounter and Recommendation

The I2C registers

The DIST-Nx appears as I2C registers as follows:

Register	Read	Write
0x00-0x07	Firmware Version – V3.00	-
0x08-0x0f	Vendor Id – mndsnsrs	-
0x10-0x17	Device ID- Dist-S(short range) - Dist-L(Long range) - Dist-M(medium range)	-
0x41	-	Command
0x42	Distance data LSB	-
0x43	Distance Data MSB	-
0x44	Voltage Data LSB	-
0x45	Voltage Data MSB	-

Table 1 - The DIST-Nx I2C registers

Data is read from the corresponding data register. Register 0x41 is for writing to the sensor. **Current consumption is 38mA while the sensor is Energized and 5mA while the sensor is De-energized**

Port Splitters

Figure 19- SPLIT-Nx-V2 Port Splitter (*image taken from the datasheet*)

The SPLIT-Nx_v2 allows for connection of multiple I2C compliant devices into a single NXT port.

Simulation

Once the parameters of the robot have been fine-tuned the process of coordinating the algorithm with these quantities is to be realized. In turn, the testing of a virtual model of the robot in a simulated world can begin as a means to gauge the performance of the physical robot in a real world environment.

The codes presented were seen to be very well organised which made access to them very easy. The codes were divided into separate functions making pin-pointing which areas of the code needed to be manipulated uncomplicated. In saying this, explanations concerning what certain areas of the code indicated were still a necessary requirement from the supervisor. For simulations to begin, each of the current robot parameters essential to the codes needed to be incorporated. This section will introduce the main functions and indicate which parameters within these functions needed alteration.

Function 1:

“get_robot_parameters” (Appendix A1)

As the name suggests, in this function the main parameters of the robot are to be inserted.

The first aspect that one needed to look at was the case type. There were several cases available each pertaining to the drive type (Tricycle, Differential) and shape of the robot (Circular or Non-circular). The case type chosen for this project was **“case {3,6}”** (Line 611) which represented a **“rectangular robot with differential drive”**.

Secondly, the maximum kinematics and dynamics (Lines 622 – 623 and 634 – 637) of the robot were set and each value was to be given in standard SI Units. (Refer to Experimentation and Results Section for obtaining of values). Maximum Linear Velocity is shown by **“robot.velLinMaxPhysical = 0.2168;”** and the Maximum Angular Velocity given by **“robot.velAngMaxPhysical = 1.377;”.**

The Maximum Linear Acceleration and Maximum Angular Acceleration were given by;

"robot.linAccel = 1;

robot.linAccelPerReactiveCycleMax = 0.1;

robot.angAccel = 0.85*pi;

robot.angAccelPerReactiveCycleMax = 0.085*pi;"

(These parameters were obtained via calculation coupled with experimentation of the simulated and physical robot. Also, the linear and angular accelerations per reactive cycle are found by multiplying the linear acceleration and angular acceleration by the simulation step time. In this case, the simulation step time used was 0.1s)

The next factor to be looked at is the robot radius. The algorithm requires the robot, regardless of whether it is circular or non-circular in orientation, to have a specified robot radius. This case type required two sets of robot radius', a circumscribed radius (maximum) and an inscribed radius (minimum). The syntax is as follows robot.radiusActual = [min,max] = "**robot.radiusActual = [0.14 0.27];".**

Next comes the defining of the virtual equivalent of the physical robot for simulation. The physical robot is of a rectangular shape whose center was taken to be the mid-point between the two tires (or midpoint of axel) since the robot turns about the center. The dimensions of the robot were measured to be 0.34m x 0.28m (L x W). Figure 19 below shows the proportions of the robot.

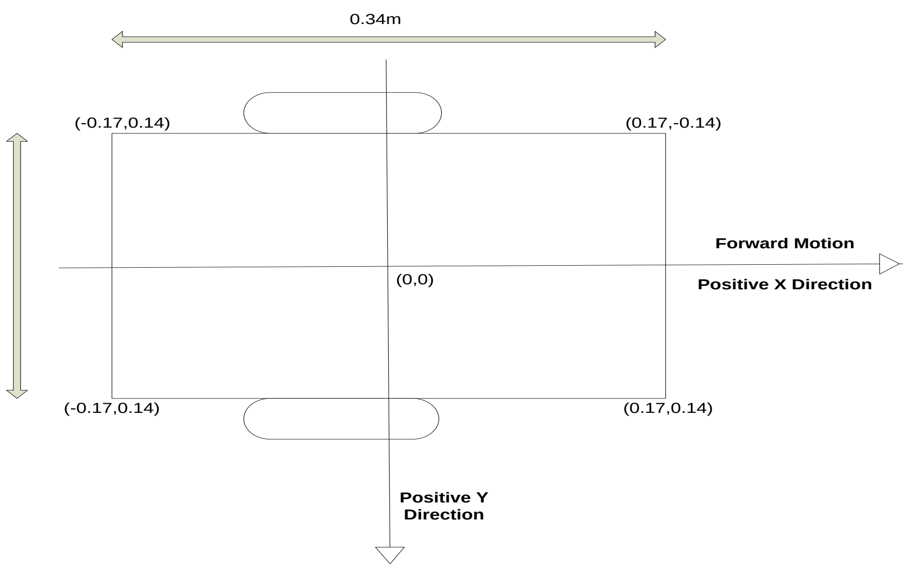


Figure 20 - Robot Dimensions

Therefore, as seen, taking the center to be the origin (0,0), each of the 4 corners coordinate's can be evaluated. It is these very coordinates that the algorithm uses to represent the virtual appearance of the actual robot. Starting from one corner and moving to the next point (clockwise or anticlockwise) until the initial point is again reached, the coordinates are placed in an array to be translated by the program. The syntax of the program is as follows;

```
"robot.polygonCoords = [-0.17 -0.14; (Starting Position)
                         0.17 -0.14;
                         0.17 0.14;
                        -0.17 0.14;
                       -0.17 -0.14];" (Back To Start)
```

The next set of parameters was introduced to the algorithm in a similar manner. Once the virtual robot coordinates have been entered the process of entering the sensor locations onto the virtual platform begins. The data concerning the placement of the sensors is also input into an array and reads as follows [x-coordinate y-coordinate angle at which sensor faces in radians;]. Since only 4 out of the 6 sensors were used, there would be 4 sets data. One must note that the angle of the sensor placements has a slight twist to it due to the positive y-axis being in the downward direction as seen in Figure 19 (Robot forward motion towards the right) .

```
"robot.irRelPoses = [0.08   0.10   -70*pi/180;
                     0.08   0.13   -30*pi/180;
                     0.08   -0.10   70*pi/180;
                     0.08   -0.13   30*pi/179];"
```

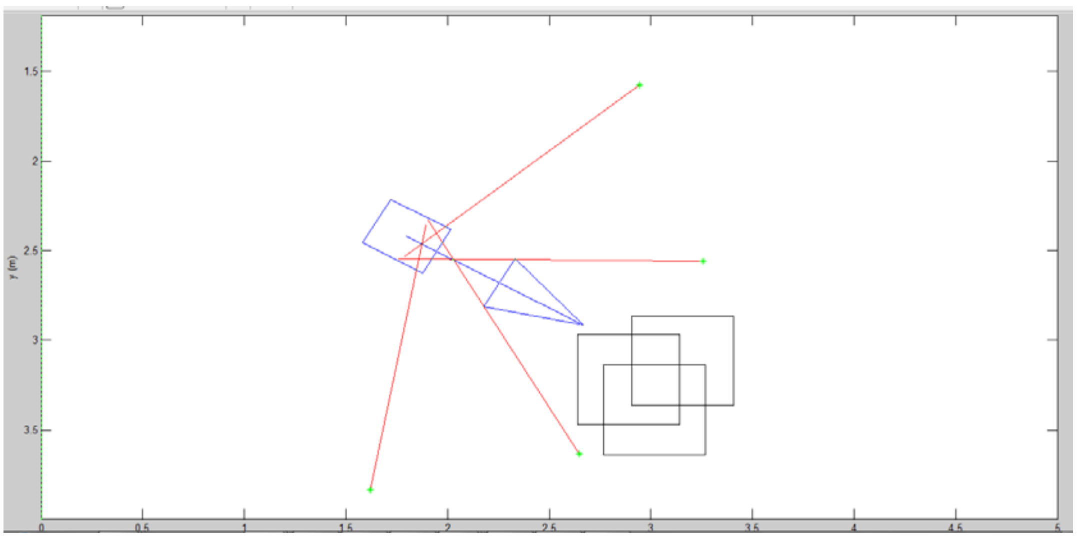


Figure 21 - Screenshot of Virtual Robot Representation with Sensor Placements (Red Lines Show Sensor Range)

Linked to the sensor positions are the sensor designations (Lines 708 – 713). This portion of the code basically differentiates the front sensors from the back sensors and hence from the side sensors. As previously mentioned there were only 4 IR sensors utilised and all were in a forward orientation signifying only forward motion being implemented, as shown in Figure 20. The algorithm, however, also required rear, immediate rear, left and right sensors. Without these parameters it was found that the algorithm would not function, hence, arbitrary values (found using testing) were used to compensate for this dilemma. An extra piece of information needed came in the form of **"robot.reversingParam = [0.6 0.8 0.85 1.05 0.5 0 1.25];"** which as the name indicates is essential for the robot to employ reversing. These values, however, did not need alteration because these were values specifically calibrated by originators of this code. The following shows the syntax of these parameters within the code;

"robot.frontSensors = [2 4]; (sensors 2 and 4 = sensors facing 30°)
robot.rearSensors = [1]; (sensor 1 = sensor facing -70°)
robot.immRearSensors = [1];
robot.leftSensors = [3]; (sensor 3 = sensor facing 70°)
robot.rightSensors = [1];
robot.reversingParam = [0.6 0.8 0.85 1.05 0.5 0 1.25];"

The final parameter to be tuned is the safety margin (Line 668). The values found are estimated by a circle or rectangle and since the robot being used is already in a rectangular orientation the values entered are close to the robot's dimensions. The quantities used are entered as follows;
" **robot.safetyMarginParam"** = [extra value added to safety margin limits limit on negative-x side limit on the positive-x side limit on the negative-y side limit on the positive-y side]; = "**[0.03 -0.22 0.22 -0.17 0.17];".**

Function 2:

"dynamic_window" (Appendix A2)

This function basically altered the type of control used in the simulation, that is, track planning or reactive control. Since reactive control is required in this project the track planning portion is removed from the code (Line 80) leaving only the **"pure reactive"** portion (Line 82).

Function 3:

“get_environment_data” (Appendix A3)

In this portion the maps used in simulation can be chosen since not all environments are suitable for the robot. For example, if the map becomes too complex by having too many obstacles the robot is unable to maneuver itself around the obstacles appropriately even if all six sensors were to be used. The selecting of maps is done by simply removing the lines of code relating to the unwanted maps.

Function 4:

“sim_obstacle_detect” (Appendix A4)

This function along with all the following are the sections that deal with the feedback of the system.

In “sim_obstacle_detect” the distances to obstacles detected by the sensors are returned and read here. Refer to Line 49 of Appendix A3 for code representation.

Function 5:

“sim_robot_motion” (Appendix A5)

Here, the *Motor Settings* portion is inserted into the code (Line 21). For more information concerning the motor settings code refer to the “Motor Settings” section in the Appendix B (B2).

Function 6:

“gui_main” (Appendix A6)

This final section contains the proportions having to do with “Bluetooth connection” the “Motor Feedback” and the “Pose prediction”. The piece of code **“connect BT();”**, added in Line 64, is included to keep the Bluetooth connection established with the robot alive and persistent. The “Motor Feedback” (Lines 747 – 748) and “Pose Prediction” (Line 751) codes are shown and discussed in the Appendix B of this report. (Motor Feedback – Section B2 of Appendix, Pose Prediction – Section B5 of Appendix)

<u>Simulation Screen Shots:</u>

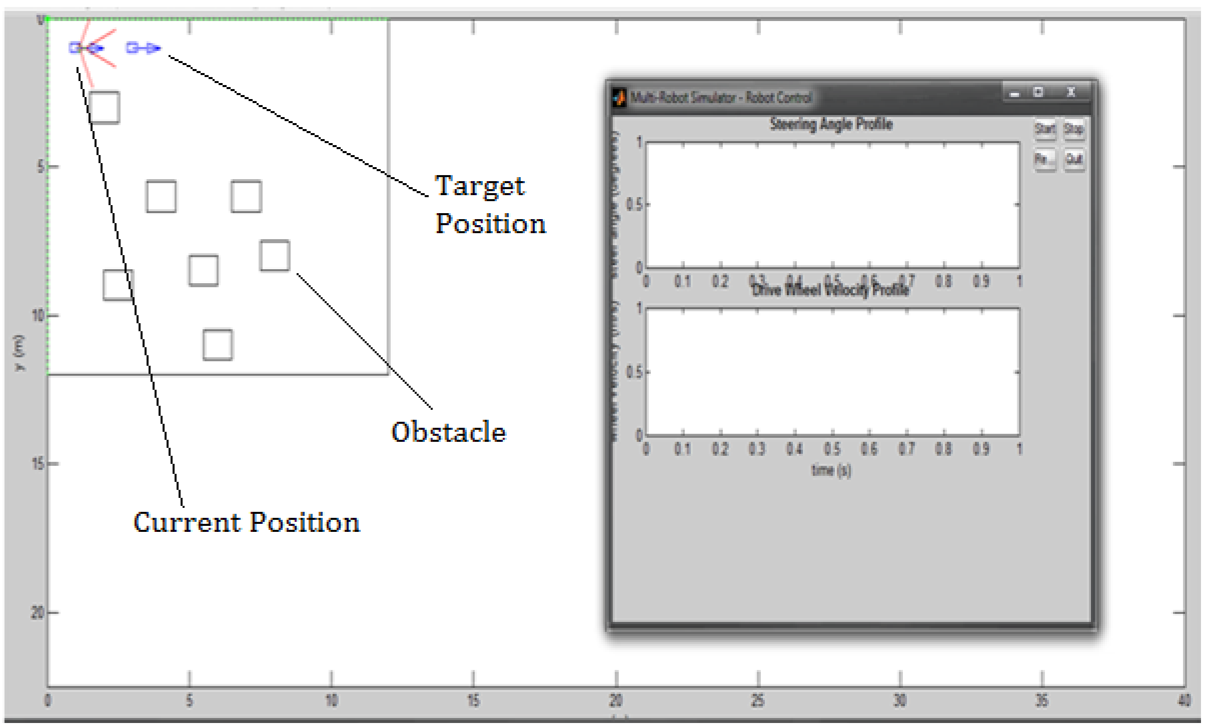
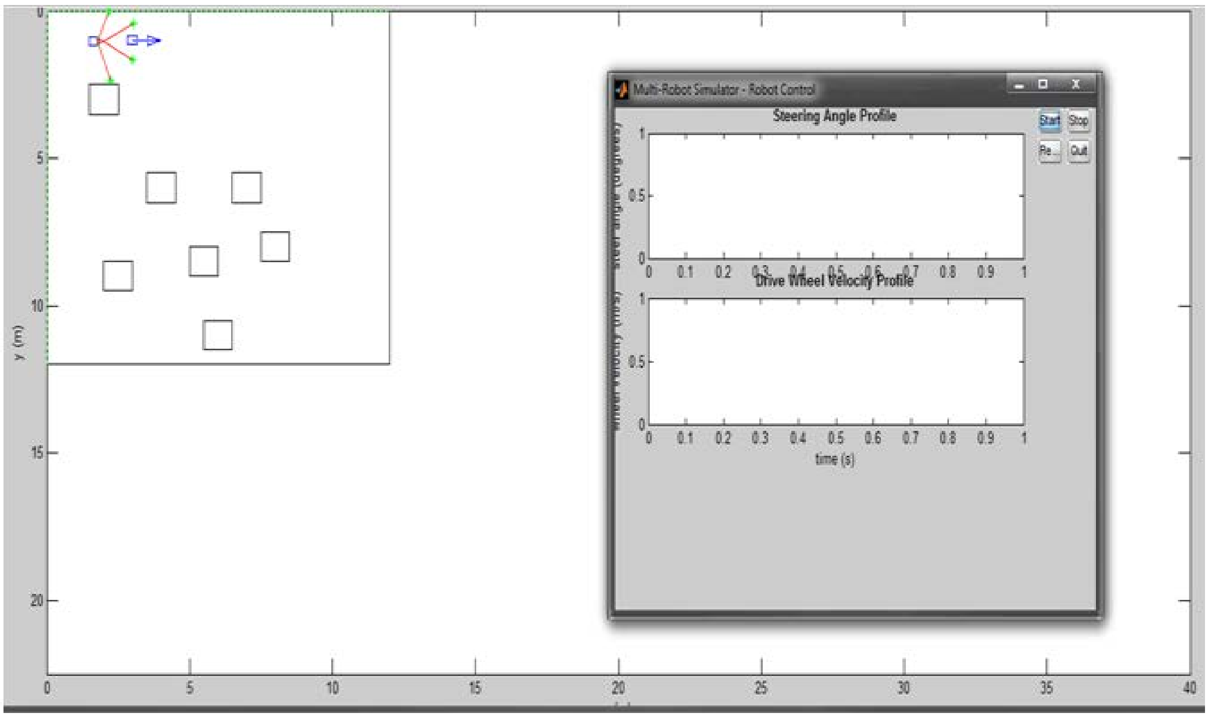


Figure 22 - Simulation Start Window with GUI



Multi-Robot Simulator - Robot Control
Steering Angle Profile
Drive Wheel Velocity Profile
time (s)
Start
Stop
Re...
Quit
y (m)

Figure 23 - Simulation at Work

Experimentation and Results

Maximum Linear Velocity

Trial 1:

Parameters: Power Level = 100%, Smooth Table Top Surface

Distance (cm)	***10***	***15***	***30***	***45***	***60***	***80***	***100***
Time(s)	*0.371*	*0.893*	*1.580*	*2.346*	*2.867*	*4.203*	*5.143*

Table 1 - Trial 1 to Determine Maximum Linear Velocity

Trial 2:

Parameters: Power Level = 100%, Smooth Table Top Surface

Distance (cm)	***10***	***15***	***30***	***45***	***60***	***80***	***100***
Time(s)	*0.508*	*0.674*	*1.693*	*2.37*	*3.16*	*4.29*	*5.12*

Table 2 - Trial 2 to Determine Maximum Linear Velocity

Trial 3 – Encoder Readings:

Encoder Count = 360 slots = 1 revolution
Turns = (Final Encoder Count – Initial Encoder Count) ÷ 360
Velocity = ((Distance/rev) x Turns) ÷ Time

Parameters: Power Level = 100%, Distance/rev = 0.106 meters, Unloaded(Suspended)

Shaft Encoder Trials	***Revolutions***	***Time(s)***	***Velocity (m/s)***
1	*20*	*10.647*	*0.2065*
2	*20*	*10.398*	*0.21105*
3	*20*	*10.05*	*0.2168*

Table 3 - Trial 3 to Determine Maximum Linear Velocity Using the Motor's Internal Tachometer

Trial 1 = 20.19cm/s = 0.2019m/s

Trial 2 = 19.40cm/s = 0.1940m/s

Trial 3 = 0.21145m/s (more accurate assessment)

Maximum Linear Velocity = 0.21m/s @ 100% Motor Power

Infrared Sensor Testing

Since there was no proper procedure described for the extraction of data, a test was conducted to see the values read from register 0x42. The readings are taken from 10cm to 150cm at intervals of 10cm. Matlabs RWTH toolbox was used to read the appropriate data. 6 sets of readings were taken at every interval.

Distance(cm)	Reading LSB						Average reading
10	227	231	231	229	197	197	218.6666667
20	227	231	231	229	229	223	228.3333333
30	76	76	76	76	76	76	76
40	141	142	140	138	141	141	140.5
50	7	7	7	7	7	7	7
60	103	103	103	103	103	103	103
70	218	218	218	218	214	214	216.6666667
80	24	24	27	24	24	24	24.5
90	114	107	114	114	114	107	111.6666667
100	213	207	213	213	213	213	212
110	242	242	228	242	242	245	240.1666667
120	74	74	74	74	74	70	73.33333333
130	194	194	188	194	194	146	185
140	188	188	194	194	243	228	205.8333333
150	66	66	66	66	55	66	64.16666667

Table 5 - Initial Infrared Sensor Testing Results

The results obtained were clearly not the correct distance readings, hence, after further investigation the importance of the Distance MSB register was realised. This particular register is imperative in the calculation of the actual distance. Looking at the results obtained above, the LSB data only reached a value of 255 or less and upon further testing the MSB data was discovered to be an integer below 6. Using the data from both registers the following formula was derived;

Distance (mm) = DistanceMSB * 255 + DistanceLSB

Another test was then conducted to assess the validity of the formula shown above. To do this, a small code was written and implemented. The code, in essence, initializes and configures the sensor for data retrieval. The correct I2C register (in this case register 42 and 43 for distance) is accessed to retrieve data, then converted to distance (in mm) and displayed.

```
    status = NXT_SetInputMode(SENSOR_1, 'LOWSPEED_9V', 'RAWMODE', 'dontreply');

                    D1 = COM_ReadI2C(SENSOR_1, 1,uint8(2), uint8(66))
                    D2 = COM_ReadI2C(SENSOR_1, 1,uint8(2), uint8(67))
                     range=(double(D2)*255+double(D1(1)))
```

The measured distance and sensor readings were recorded in Table 6 below.

Distance(mm)	Reading(mm)
100	378
200	213
300	306
400	416
500	502
600	606
700	698
800	788
900	912
1000	1021
1100	1097
1200	1212
1300	1306
1400	1421
1500	1478

Table 6 - Second Round of IR Sensor Testing Results

The results of the sensor readings compared to the measured values are seen to be quite accurate. It must be noted that inaccuracies become significant below the 20cm mark and this is due to the sensor blind spots described in the IR sensor datasheet.

Physical Robot Testing

Once the various parameters have been tuned and accounted for in the codes, the actual robot testing could begin. The following are the experiments conducted to test the robot's reactive control system.

Test 1 – Against Simulation:

In this test, the connection between the physical robot and its simulated version was examined. The test basically compromised of hindering the robot's line of sight by blocking certain sensors and observing how the virtual robot reacted. This test was a success in that the simulated version was clearly seen to respond to the effects of the real world manipulations.

<u>Test 2 – Real World:</u>

This next test was conducted in a make-shift real world environment using cartons as obstacles. With the absence of sensors, the robot was just tested in a linear point to point journey. Basically, the robot was given a target heading (defined by the simulated target heading) and an obstacle was placed to obstruct its path to the desired location.

The tests revealed a myriad of anomalous results. For instance, in one case the robot would travel to the desired location successfully bypassing the obstacle. However, repeating the experiments using the exact same parameters revealed a less satisfactory result. This test was repeated a number of times, altering the various parameters, with the sole intention of attaining consistent results for the robot's reactive motion.

Note: These experiments were carried out with the maximum linear velocity being halved to compensate for MATLAB's real time application latency (Refer to the Problems Encountered Chapter). It was found that by using the maximum velocity, the system was moving with a constant jerking motion so that the proper velocities and other parameters could be calculated (Seen due to the "PAUSE" line in the coding to account for recalibrations done by the algorithm). Altering the velocity lessened the latency predicament.

<u>Final Analysis:</u>

The robot was able to effectively detect obstacles but the response to these obstacles was quite sporadic. The best possible configuration is shown in the codes located in the Appendix A of this report. This system gave the most consistent results concerning the obstacle detection and avoidance ability of the robot. Troubleshooting to find solutions to the reactive control irregularities was made more difficult because factors such as correcting latency issues proved unsuccessful.

Problems Encountered And Recommendations

This section will highlight the difficulties faced during the course of this project. Some of which are solved, some of which are solved but not used, while others still need attention

<u>IR sensor</u>

Due to the delayed arrival of these sensors, extensive testing and experimentation could not be performed to find a proper solution to some issues that were faced.

This project required the use of six infrared sensors. In attaching all six sensors we found that only four could work. This was due to the current limitation from the NXT brick as removing the motors could accommodate a fifth sensor.

<u>Bluetooth communication</u>

Bluetooth communication was achieved but it did not hold feasible results due to the high latency of Bluetooth. Testing was done via a USB at all times.

In using Bluetooth to communicate, the robot would be jerky, or not move at all.

<u>Real-time to simulation</u>

Through the course of this project a major problem that kept hindering progress the algorithm ability to work in real-time with the NXT robot. The complex nature of the algorithm made finding all the correct parameters difficult.

Possible Solutions

<u>IR sensors</u>

Insufficient current issue could be solved creating a program in which 1 sensor would take readings at any time. This would solve the current issue, as the data sheet suggests decrease from 38mA to 5mA, doing this might allow for use of a much higher number of sensors then the requirement suggests.

The same could be done with the Ultrasonic sensors to avoid interference.

Bluetooth Communication

Bluetooth communication is not advised for this application.

Real time solutions

Some parameters that could be looked to make the robot work in real-time.

SimulationStepTime *(line 54 of Get_Test_Configurations)*

- the control loop time of the algorithm. Was changed to get reasonable results. Could be adjusted more for a more desirable result.

robot.comm.transferRate(*line 138 of Get_Robot_Parameters)*

- suggests a transfer rate between the robot and the computer.

robot.comm.processorTime(*line 138 of Get_Robot_Parameters)*

- time to send or receive a byte of data from buffer (without factor to improve accuracy).

robot.comm.processorTimeFactor (*line 138 of Get_Robot_Parameters)*

- comm processor time factor to convert to real time.

Conclusion

A steady, agile and relatively quick mobile robot platform was successfully designed and assembled with all necessary onboard components (IR Sensors, NXT intelligent brick) added. In turn, the *modified dynamic window method* and *polar histogram technique* were incorporated from the code by finding and inserting the appropriate parameters required by these systems to achieve reactive control.

Experiments conducted on the physical robot while practicing reactive control ascertained that the robotic system was able to detect obstacles and then exercise a fair degree of obstacle avoidance. This was attributed to the lack of IR sensors and problems that arouse with MATLAB's real time application.

Works Cited

Chand, P. and Carnegie, D.A. (2011) ‘Development of a navigation system for heterogenous mobile robots’, Int. J. intelligent Systems Technologies and Applications, Vol. 10, No. 3, pp. 250-278

Mataric, M. J. (n.d.). *The Basics Of Robot Control*. Retrieved 09 19, 2011, from http://robotics.usc.edu/~maja/robot-control.html

High Precision Long Range Infrared distance sensor for NXT (DIST-Nx-Long-v3). (n.d.). Retrieved September 20, 2011, from mindsensors.com: http://www.mindsensors.com/index.php?module=pagemaster&PAGE_user_op=view_page&PAGE_id=73

Atorf, L., Behrens, A., Knepper, A., Schwann, R., Schnitzler, R., Ballé, J., et al. (n.d.). *RWTH - Mindstorms NXT Toolbox for MATLAB*. Retrieved September 20, 2011, from http://www.mindstorms.rwth-aachen.de/

"Philo"Hurbain, P. (2000-2009). *NXT® motor internals*. Retrieved November 25, 2011, from philohome: http://www.philohome.com/nxtmotor/nxtmotor.htm

9842 Interactiv Servo Motor. (n.d.). Retrieved November 15, 2011, from Lego Mindstorm: http://mindstorms.lego.com/en-us/products/interactiv+servo+motor/9842.aspx

Dixon, J., & Henlich, O. (1997). *Mobile Robot Navigation.* London.

Lego Education. (n.d.). Retrieved November 23, 2011, from Lego Education: http://www.legoeducation.us/eng/product/

Appendix A

A1- Get_Robot_Parameters

the commented(code in green) are used.

the uncommented functions are used

```
case {3,6}
        % make a rectangular robot with differential drive
        % robot name
        % set processor benchmark
        robot.cpuBMMul = 60/10;%dellProcBM/robotMicroBM
        i = robotID-3;
        robot.name = ['Tank' num2str(i)];

        % robot physical specifications and constraints
        robot.isBogCapable = 1;

        robot.velLinMaxPhysical = 0.2168%0.2168 ;%0.4;
        robot.velAngMaxPhysical  =1.377%0.7108 % was pi/8
        robot.velAng2velLinRatio = 6.35%3.278%pi/3;

        robot.shape = 'non-circular'; % circular or non-circular
        robot.drive = 'differential'; % differential or tricycle

        robot.thetav = 0;
        robot.Yoff = 0;%-0.27; % -0.27 or 0
        robot.B = 0.62;
        robot.wheelSep = 0.305; %0.66;

        robot.linAccel = 1;%0.75;%1;% linear acceleration in m/sec2
        robot.linAccelPerReactiveCycleMax = 0.1;%0.3015;
        robot.angAccel = 0.85*pi%3;% angular acceleration in rad/sec2
        robot.angAccelPerReactiveCycleMax = 0.085*pi;%1.2;%0.085*pi;

        % actuator noise (assume constant for all speeds except zero)
        robot.driveNoiseMax = 0;%0.01;
        robot.steerNoiseMax = 0;%(2*pi/180);

%         robot.distancePerReactiveControlCycle =
robot.velLinMaxInitStore/robot.reactiveControlFrequency;% relate this to grid
resolution

        % constrain the velocity limits
        [robot.velLinMaxInitStore,robot.velLinMaxInit,robot.velAngMax] =
constrain_lin_ang_vel(robot.velLinMaxInitStore,robot.velLinMaxPhysical,robot.velAng
2velLinRatio,robot.velAngMaxPhysical);

        % constrain the acceleration limits
        [robot.linAccelPerReactiveCycle,robot.angAccelPerReactiveCycle] =
constrain_lin_ang_acc(robot.linAccel,robot.linAccelPerReactiveCycleMax,robot.angAcc
el,robot.angAccelPerReactiveCycleMax,robot.reactiveControlFrequency);


        % prescribe a circle with radius r around the robot (for both circular and
non-circular)
        % could adapt this radius based on the obstacle memory
        % size
        robot.radius = 0.14%0.67;%(0.65-bigrobot) (0.45-smallrobot); % radius used
by control algorithms (includes a small safety margin for circular robots 0.45)

        %robot.radiusActual = 0.35;%(0.55-bigrobot) (0.35-smallrobot); % radius
used by plot commands and path planner for circular robot.
        %if (strcmp(robot.shape,'non-circular'))
        % specify min and max radius
        robot.radiusActual = [0.14 0.27];%[0.4 0.67];
        %end;
        robot.polygonCoords = [-0.17 -0.14;
            0.17 -0.14;
            0.17 0.14;
            -0.17 0.14;
            -0.17 -0.14];
        % safety margin approximated by a circle or rectangle
        robot.safetyMarginParam =[0.03 -0.22 0.22 -0.17 0.17]; %[0.03 -0.22 0.22 -
0.17 0.17];%[0.07 -0.67 0.67 -0.47 0.47];%[0.1 -0.7 0.7 -0.5 0.5];
        robot.irRelPoses = [
            %0.42    0.09    0*pi/180;
            %0.42    -0.09   0*pi/180;
            %0.43    0.16    35*pi/180;
            %0.43    -0.16   -35*pi/180;
            %0.405   0.205   90*pi/180;
            %0.405   -0.205   -90*pi/180;
            %0.09    0.18    90*pi/180;
            %0.09    -0.18    -90*pi/180;
            %-0.32   0.18    90*pi/180;
            %-0.32   -0.18    -90*pi/180;
            %-0.32   0.15    145*pi/180;
            %-0.32   -0.15    -145*pi/180;
            %-0.33   0.12    -180*pi/180;
            %-0.33   -0.12    -180*pi/180];
              0.08    0.10   -70*pi/180;
              0.08    0.13    -30*pi/180;
              0.08    -0.10     70*pi/180;
              0.08    -0.13     30*pi/179];
         %    0.0   -0.05      -180*pi/180];
%               -0.08    -0.005     90*pi/180];
        robot.numberOfSensors = size(robot.irRelPoses,1);
        robot.obstacles(1:size(robot.irRelPoses,1),1:3) = 0;
        robot.rangeStore(1:size(robot.irRelPoses,1)) = 0;
        robot.obstacleMemorySize = 0;
        robot.obstacleMemoryStore = [];%robot.obstacles;

        % reactive control parameters
        robot.dirSensorParam = [1.5 0 41 9 25 0.5 1];
        robot.dynamicWindowParam = [-robot.linAccelPerReactiveCycle ...
            robot.linAccelPerReactiveCycle ...
            -robot.angAccelPerReactiveCycle ...
            robot.angAccelPerReactiveCycle ...
            0.4 0 0.2 robot.velAngMax 1e99 2 2.5 5 11 ...
            0.5 ...
            2 0.5 0.2...
            0 0 0]; %(last param 0.7-bigrobot) (last param 0.5-smallrobot)
        % curvatureMax = 1e99 for diff drive   2.135 for tricycle
        %essential parameters for implementing reversing
        robot.frontSensors = [2 4]%[1 2 3 4];
        robot.rearSensors = [1]%[11 12 13 14];
        robot.immRearSensors = [1]%[13 14]; robot doesn’t work if there are no rear
                                 sensors, due to the important reversing parameters

         robot.leftSensors = [3]%[5 7 9];
        robot.rightSensors = [1]%[6 8 10];
        robot.reversingParam =[0.6 0.8 0.85 1.05 0.5 0 1.25];%[0.6 0.75 0.75 0.95
0.5]-bigrobot; %[0.5 0.7 0.65 0.85 0.5]-smallrobot;
        % round(robot.velLinMaxInitStore*20)
        %goal homing parameters
        robot.goalHomingParam = [1.5 2 0.3162 0.3 0.15*pi 0.2 0.05*pi];

        %parameters for adapting robot clearance in path
        %planning
        robot.poseCurPrevPlan = [];

        %robot stuck state detection tunable parameters
        robot.task.mapBuilding.exploration.pathPlanningParameters.replanWaitTime =
20; %seconds @ 0.6 m/sec for replan wait time
```

```
robot.task.mapBuilding.exploration.pathPlanningParameters.noProgressDetectionTime =
[20 60]; %[stuck disabled] seconds @ 0.6 m/sec for stuck time

robot.task.mapBuilding.exploration.pathPlanningParameters.noProgressDetectionVeloci
tyFactor = [0.2 0.1]; % factor of maximum speed [stuck disabled]

        robot.velArraySize(1) =
robot.task.mapBuilding.exploration.pathPlanningParameters.noProgressDetectionTime(1
)*robot.sensingFrequency*0.5/robot.velLinMaxInitStore; %to be tuned for each robot
(first num = time)
        robot.velArraySize(2) =
robot.task.mapBuilding.exploration.pathPlanningParameters.noProgressDetectionTime(2
)*robot.sensingFrequency; %to be tuned for each robot (first num = time)
        %replanning wait time
        robot.replanCounterMax =
robot.task.mapBuilding.exploration.pathPlanningParameters.replanWaitTime*robot.sens
ingFrequency*0.5/robot.velLinMaxInitStore; %(first num = time) check this.. perhaps
should be function of sensing time


robot.task.mapBuilding.exploration.navigationGoals.endTourPointGoalThresholdDistanc
e = robot.radius;

        % get the robot's ideal achievement rates for resource utilisation
        robot.idealAchievementRate = get_ideal_achievement_rates(robotID);

    otherwise
        message = ['Reference to non-existent parameters for robot'
num2str(robotID) '!'];
        error(message);
end;

%exchange map data flag
robot.isMapExchanged = 0;

% set initial positions of robots  - depends on environment type

% if ((robotID == 1)&& (testConfiguration{4} == 1))
%     robot.poseCurrent(1) = 1.46; %36
% else
%     robot.poseCurrent(1) = 1.35;
% end;
% robot.poseTarget(1) = 5;
% robot.poseCurrent(2) = 1;% + (robotID - 1);
% robot.poseTarget(2) = 1;% + (robotID - 1);
% robot.poseCurrent(3) = 0; %pi
% robot.poseTarget(3) = 0;


robot.poseCurrent(1) = 6;
robot.poseTarget(1) = 3;
robot.poseCurrent(2) = 3;% + (robotID - 1);
robot.poseTarget(2) = 1;% + (robotID - 1);
robot.poseCurrent(3) = 0; %pi
robot.poseTarget(3) = 0;
```

A2-Dynamic Window

Pure reactive used instead of a planned path.

```
%[angleTarget,iNodePrev,xTrack,yTrack] =
track_path(poseCurrent,poseTarget,pathX,pathY,velCurrent(1),GRIDRESOLUTION,radius,i
NodePrev,velLinMaxInitStore); %track path

%[dwaLineHnd] = plot_path_track(xTrack,yTrack,'c',parentHnd,dwaLineHnd);
    angleTarget = atan2((poseTarget(2)-poseCurrent(2)),(poseTarget(1)-
poseCurrent(1))); %pure reactive
```

A3 – Get_Environment_Data

```
switch (world)
    %case 1
    %
[globalEnvData.obstVertex,globalEnvData.obstEdge,globalEnvData.unmappedObstVertex,g
lobalEnvData.unmappedObstEdge,globalEnvData.bogVertexEdge] = load_map();
    case 1

[globalEnvData.obstVertex,globalEnvData.obstEdge,globalEnvData.unmappedObstVertex,g
lobalEnvData.unmappedObstEdge,globalEnvData.bogVertexEdge] = load_map_rect();
    case 2

[globalEnvData.obstVertex,globalEnvData.obstEdge,globalEnvData.unmappedObstVertex,g
lobalEnvData.unmappedObstEdge,globalEnvData.bogVertexEdge] = load_map_rect_empty();
    %case 3
    %
[globalEnvData.obstVertex,globalEnvData.obstEdge,globalEnvData.unmappedObstVertex,g
lobalEnvData.unmappedObstEdge,globalEnvData.bogVertexEdge] = load_map_rect_new();
    %case 4
    %
[globalEnvData.obstVertex,globalEnvData.obstEdge,globalEnvData.unmappedObstVertex,g
lobalEnvData.unmappedObstEdge,globalEnvData.bogVertexEdge] = load_map_office();
    %case 2
     %
[globalEnvData.obstVertex,globalEnvData.obstEdge,globalEnvData.unmappedObstVertex,g
lobalEnvData.unmappedObstEdge,globalEnvData.bogVertexEdge] = load_map_random_5();
    %case 3
    %
[globalEnvData.obstVertex,globalEnvData.obstEdge,globalEnvData.unmappedObstVertex,g
lobalEnvData.unmappedObstEdge,globalEnvData.bogVertexEdge] = load_map_rect_empty();
    otherwise
        message = ['Reference to non-existent world ' num2str(world) '!'];
        error(message);
end;
```

A4- Sim_obstacle_detect

```
%[rangeline] =
sim_irobstacle_range_submex(irLines(:,1),irLines(:,2),irLines(:,3),irLines(:,4),pos
eCurrent,obstVertex(:,1),obstVertex(:,2),obstEdge(:,1),obstEdge(:,2),unmappedObstVe
rtex(:,1),unmappedObstVertex(:,2),unmappedObstEdge(:,1),unmappedObstEdge(:,2),other
RobotMapTemp,rangeMin,rangeMax);

[rangeline] = SensorRange();      %actual sensors used instead of simulation
```

A5- Sim_Robot_Motion

```
function
[velProf,steerProf,time,poseCurrent,totalForwardDistance,totalReverseDistance,velCu
rrent,steerAngCurrent,thetav,runVelMem] =
sim_robot_motion(velProf,steerProf,driveNoiseMax,steerNoiseMax,drive,velTarget,stee
rAngCurrent,thetav,Yoff,B,dynamicWindowParam18,steerAngTarget,runDWA,time,simulatio
nStepTime,posCurrentHnd,totalForwardDistance,totalReverseDistance)

velProf(1,2:100) = velProf(1,1:99);
steerProf(1,2:100) = steerProf(1,1:99);
velProf(2,2:100) = velProf(2,1:99);
steerProf(2,2:100) = steerProf(2,1:99);
%simulate noise
[driveNoise,steerNoise] = sim_steer_drive_noise(driveNoiseMax,steerNoiseMax);
% if (strcmp(drive,'tricycle'))
%     [velProf(2,1),steerAngCurrent,thetav,runVelMem,velCurrent] =
convert_velocities(velTarget,steerAngCurrent,thetav,Yoff,B,dynamicWindowParam18,ste
erAngTarget,runDWA,driveNoise,steerNoise);
if (strcmp(drive,'differential'))
    [velProf(2,1),steerAngCurrent,thetav,runVelMem,velCurrent] =
convert_velocities(velTarget,steerAngCurrent,thetav,Yoff,B,0,steerAngTarget,runDWA,
driveNoise,steerNoise);
end;
steerProf(2,1) = steerAngCurrent*180/pi;
time = time + simulationStepTime;
velProf(1,1) = time;
steerProf(1,1) = time;
%pause(.05);
velocity=velCurrent;

  MotorSettings(velCurrent) %Motor Setting being sent


 poseCurrent = gui_move_pose('GetCoords',posCurrentHnd);
 [poseCurrent,distance,reverseDistance] =
predict_pose(poseCurrent,velCurrent,thetav,Yoff,simulationStepTime)
totalForwardDistance = totalForwardDistance + distance;
totalReverseDistance = totalReverseDistance + reverseDistance;
```

A6 – Gui_main

```
for iRobot = 1:nWorkerRobots
            if (~robot{workerRobotIDs(iRobot)}.nDelaySteps)
            if
(robot{workerRobotIDs(iRobot)}.sensingLoopCounter==robot{workerRobotIDs(iRobot)}.se
nsingLoopCounterReset)
                tocTimesArray(3) = toc;
                [velCurrent]=MotorFeedback();  %get feedback using motor Encoder
%                [velCurrent]=feedback() %get feedback using motor power
                % bluetooth commands to receive current speed

[robot{workerRobotIDs(iRobot)}.velCurrent,robot{workerRobotIDs(iRobot)}.steerAngCur
rent,robot{workerRobotIDs(iRobot)}.velCurrentStore,robot{workerRobotIDs(iRobot)}.ve
lLinMaxInit] =
sim_velocity_filtering(robot{workerRobotIDs(iRobot)}.driveNoiseMax,robot{workerRobo
tIDs(iRobot)}.steerNoiseMax,robot{workerRobotIDs(iRobot)}.drive,robot{workerRobotID
s(iRobot)}.velCurrent,robot{workerRobotIDs(iRobot)}.steerAngCurrent,robot{workerRob
otIDs(iRobot)}.B,robot{workerRobotIDs(iRobot)}.Yoff,robot{workerRobotIDs(iRobot)}.v
elCurrentStore,robot{workerRobotIDs(iRobot)}.velLinMaxInit,robot{workerRobotIDs(iRo
bot)}.velLinMaxInitStore);

 % predict pose commands from sim_robot_motion (lines 19 &
 % 20) or use your functions


 % poseCurrent =
gui_move_pose('GetCoords',robot{allRobotIDs(iRobot)}.posCurrentHnd);
 % [poseCurrent,distance,reverseDistance] =
predict_pose(robot{allRobotIDs(iRobot)}.poseCurrent,velCurrent,robot{workerRobotIDs
(iRobot)}.thetav,robot{workerRobotIDs(iRobot)}.Yoff,simulationStepTime);
                  tocTimesArray(4) = toc;

robot{workerRobotIDs(iRobot)}.executionTime.sensing.obstacleDetection =
robot{workerRobotIDs(iRobot)}.executionTime.sensing.obstacleDetection +
tocTimesArray(4) - tocTimesArray(3);
            end;
            end;
        end;
```

Appendix B

B1-MOTOR FEEDBACK

```
function [velCurrent]=MotorFeedback()
     L=.305;                                                %Length from wheel to wheel

     Data =  NXT_GetOutputState(MOTOR_B) ;
     tictic(1)
     Data2 = NXT_GetOutputState(MOTOR_C) ;
     tictic(2)
     initialRight=Data.TachoCount;
     initialLeft=Data2.TachoCount;
     pause(.7);
     RotationTimeR1=toctoc(1);                   %needed to calculate the time delay of
                                                 %running the NXT_GetOutputState()command;


     Data =  NXT_GetOutputState(MOTOR_B) ;
     RotationTimeR2=toctoc(1);
     RotationTimeL1=toctoc(2);                   %needed to calculate the time delay of
                                                 %running the NXT_GetOutputState()command


     Data2 = NXT_GetOutputState(MOTOR_C);
     RotationTimeL2=toctoc(2);

     FinalRight=Data.TachoCount;
     FinalLeft=Data2.TachoCount;

     TurnsRight=(FinalRight-initialRight)/(360);   %gets the number of right motor turns
     TurnsLeft=(FinalLeft-initialLeft)/(360);      %gets the number of left motor turns

     NewVr = .1065*TurnsRight/(RotationTimeR2-(RotationTimeR2-RotationTimeR1)) ;
%feedback right wheel velocity without the time delays from the commands


     NewVl = .1065*TurnsLeft/(RotationTimeL2-(RotationTimeL2-RotationTimeL1))  ;
%feedback left wheel velocity without time delays from the commands


    velCurrent(1)= (NewVr+NewVl)/2;
    velCurrent(2) = (NewVr-NewVl)/L;
```

B2-MOTOR SETTINGS

```
function[]=MotorSettings(velCurrent)


Vmax = .21;
Wmax = 1.3;
L=.305;



C = velCurrent(1)/velCurrent(2);
 if (velCurrent(2) ==0)                  %if angular speed is 0

 Vr = velCurrent(1) ;
 Vl = velCurrent(1) ;

 else
 Vr = velCurrent(2)*(C+L/2);                        %right wheel velocity
 Vl = velCurrent(2)*(C-L/2)  ;                      %left wheel velocity


 end

[VrPower, VlPower] = Vel2Power(Vr,Vl)               %velocity paring

        rightMotor=MOTOR_B;
        leftMotor=MOTOR_C;
        ResetMotorAngle(MOTOR_B);
        ResetMotorAngle(MOTOR_C);

        SetMotor(rightMotor);                       %target motor data sent
        SetPower(VrPower);

        SendMotorSettings();

        SetMotor(leftMotor)
        SetPower(VlPower);

        SendMotorSettings();


end
```

B3-Velocity to power conversion

```
function [Pr, Pl] = Vel2Power(Vr,Vl)

%                   Power      Vr        Vl
 Vel2PowerArray=   [
                     -15.0000   -0.0013    -0.0151;
                    -16.0000    -0.0024    -0.0194;
                    -17.0000    -0.0140    -0.0221;
                    -18.0000    -0.0167    -0.0241;
                    -19.0000    -0.0174    -0.0253;
                    -20.0000    -0.0217    -0.0288;
                    -21.0000    -0.0233    -0.0309;
                    -22.0000    -0.0249    -0.0329;
                    -23.0000    -0.0265    -0.0351;
                    -24.0000    -0.0300    -0.0385;
                   -25.0000    -0.0318    -0.0407;
                    -26.0000    -0.0340    -0.0425;
                    -27.0000    -0.0357    -0.0444;
                    -28.0000    -0.0391    -0.0480;
                    -29.0000    -0.0408    -0.0501;
                   -30.0000    -0.0426    -0.0521;
                    -31.0000    -0.0447    -0.0539;
                    -32.0000    -0.0485    -0.0582;
                   -33.0000    -0.0497    -0.0601;
                    -34.0000    -0.0521    -0.0621;
                   -35.0000    -0.0535    -0.0636;
                    -36.0000    -0.0574    -0.0676;
                    -37.0000    -0.0589    -0.0699;
                    -38.0000    -0.0606    -0.0717;
                    -39.0000    -0.0625    -0.0732;
                    -40.0000    -0.0662    -0.0774;
                    -41.0000    -0.0675    -0.0796;
                    -42.0000    -0.0696    -0.0814;
                    -43.0000    -0.0710    -0.0831;
                    -44.0000    -0.0749    -0.0868;
                    -45.0000    -0.0774    -0.0889;
                    -46.0000    -0.0790    -0.0908;
                    -47.0000    -0.0812    -0.0926;
                    -48.0000    -0.0847    -0.0965;
                   -49.0000    -0.0865    -0.0983;
                    -50.0000    -0.0888    -0.1002;
                    -51.0000    -0.0907    -0.1024;
                    -52.0000    -0.0943    -0.1062;
                    -53.0000    -0.0958    -0.1081;
                    -54.0000    -0.0986    -0.1102;
                    -55.0000    -0.1003    -0.1121;
                    -56.0000    -0.1047    -0.1161;
                    -57.0000    -0.1070    -0.1179;
                   -58.0000    -0.1084    -0.1200;
                    -59.0000    -0.1102    -0.1219;
                    -60.0000    -0.1144    -0.1259;
                    -61.0000    -0.1170    -0.1279;
                    -62.0000    -0.1190    -0.1299;
                    -63.0000    -0.1214    -0.1320;
                    -64.0000    -0.1251    -0.1360;
                    -65.0000    -0.1274    -0.1379;
                    -66.0000    -0.1298    -0.1397
                    -67.0000    -0.1315    -0.1418;
                    -68.0000    -0.1354    -0.1456;
                   -69.0000    -0.1375    -0.1478;
                    -70.0000    -0.1394    -0.1497;
                    -71.0000    -0.1413    -0.1517;
                    -72.0000    -0.1455    -0.1554;
                    -73.0000    -0.1474    -0.1574;
                    -74.0000    -0.1497    -0.1594;
                    -75.0000    -0.1518    -0.1612;
                    -76.0000    -0.1551    -0.1650;
```

```
-77.0000    -0.1572    -0.1668;
 -78.0000    -0.1592    -0.1687;
 -79.0000    -0.1613    -0.1706;
 -80.0000    -0.1657    -0.1744;
 -81.0000    -0.1674    -0.1763;
 -82.0000    -0.1696    -0.1784;
 -83.0000    -0.1715    -0.1802;
 -84.0000    -0.1755    -0.1840;
 -85.0000    -0.1775    -0.1861;
 -86.0000    -0.1794    -0.1880;
 -87.0000    -0.1814    -0.1900;
 -88.0000    -0.1855    -0.1937;
 -89.0000    -0.1874    -0.1953;
 -90.0000    -0.1893    -0.1974;
 -91.0000    -0.1913    -0.1994;
 -92.0000    -0.1954    -0.2034;
 -93.0000    -0.1973    -0.2053;
 -94.0000    -0.1993    -0.2073;
 -95.0000    -0.2014    -0.2093;
 -96.0000    -0.2053    -0.2133;
 -97.0000    -0.2070    -0.2153;
 -98.0000    -0.2088    -0.2172;
 -99.0000    -0.2107    -0.2190;
-100.0000    -0.2168    -0.2252;

 0         0.0       0.0;
 15.0000    0.0013    0.0151;
 16.0000    0.0024    0.0194;
 17.0000    0.0140    0.0221;
 18.0000    0.0167    0.0241;
 19.0000    0.0174    0.0253;
 20.0000    0.0217    0.0288;
 21.0000    0.0233    0.0309;
 22.0000    0.0249    0.0329;
 23.0000    0.0265    0.0351;
 24.0000    0.0300    0.0385;
 25.0000    0.0318    0.0407;
 26.0000    0.0340    0.0425;
 27.0000    0.0357    0.0444;
 28.0000    0.0391    0.0480;
 29.0000    0.0408    0.0501;
 30.0000    0.0426    0.0521;
 31.0000    0.0447    0.0539;
 32.0000    0.0485    0.0582;
 33.0000    0.0497    0.0601;
 34.0000    0.0521    0.0621;
 35.0000    0.0535    0.0636;
 36.0000    0.0574    0.0676;
 37.0000    0.0589    0.0699;
 38.0000    0.0606    0.0717;
39.0000    0.0625    0.0732;
 40.0000    0.0662    0.0774;
 41.0000    0.0675    0.0796;
 42.0000    0.0696    0.0814;
 43.0000    0.0710    0.0831;
 44.0000    0.0749    0.0868;
 45.0000    0.0774    0.0889;
 46.0000    0.0790    0.0908;
 47.0000    0.0812    0.0926;
 48.0000    0.0847    0.0965;
 49.0000    0.0865    0.0983;
 50.0000    0.0888    0.1002;
 51.0000    0.0907    0.1024;
 52.0000    0.0943    0.1062;
 53.0000    0.0958    0.1081;
 54.0000    0.0986    0.1102;
 55.0000    0.1003    0.1121;
 56.0000    0.1047    0.1161;
 57.0000    0.1070    0.1179;
 58.0000    0.1084    0.1200;
 59.0000    0.1102    0.1219;
```

```
                    60.0000    0.1144    0.1259;
                    61.0000    0.1170    0.1279;
                    62.0000    0.1190    0.1299;
                    63.0000    0.1214    0.1320;
                    64.0000    0.1251    0.1360;
                    65.0000    0.1274    0.1379;
                    66.0000    0.1298    0.1397
                    67.0000    0.1315    0.1418;
                    68.0000    0.1354    0.1456;
                    69.0000    0.1375    0.1478;
                    70.0000    0.1394    0.1497;
                    71.0000    0.1413    0.1517;
                    72.0000    0.1455    0.1554;
                    73.0000    0.1474    0.1574;
                    74.0000    0.1497    0.1594;
                    75.0000    0.1518    0.1612;
                    76.0000    0.1551    0.1650;
                    77.0000    0.1572    0.1668;
                    78.0000    0.1592    0.1687;
                    79.0000    0.1613    0.1706;
                    80.0000    0.1657    0.1744;
                    81.0000    0.1674    0.1763;
                    82.0000    0.1696    0.1784;
                    83.0000    0.1715    0.1802;
                    84.0000    0.1755    0.1840;
                    85.0000    0.1775    0.1861;
                    86.0000    0.1794    0.1880;
                    87.0000    0.1814    0.1900;
                    88.0000    0.1855    0.1937;
                    89.0000    0.1874    0.1953;
                    90.0000    0.1893    0.1974;
                    91.0000    0.1913    0.1994;
                    92.0000    0.1954    0.2034;
                    93.0000    0.1973    0.2053;
                    94.0000    0.1993    0.2073;
                    95.0000    0.2014    0.2093;
                    96.0000    0.2053    0.2133;
                    97.0000    0.2070    0.2153;
                    98.0000    0.2088    0.2172;
                    99.0000    0.2107    0.2190;
                   100.0000    0.2168    0.2252];

[min_difference, array_position] = min(abs(Vel2PowerArray - Vr));
Pr = Vel2PowerArray(array_position(2),1)
[min_difference, array_position] = min(abs(Vel2PowerArray - Vl));
Pl = Vel2PowerArray(array_position(3),1)
end
```

B4-Bluetooth Connection

```
function [handle]=connectBT()

persistent Connection
if isempty(Connection)

COM_CloseNXT('all');
Connection=1;
handle = COM_OpenNXT('bluetooth.ini', 'check');
  COM_SetDefaultNXT(handle);
else
    disp('Connection already open')
end
end
```

B5-Sensor Range

```
function [Range]=SensorRange()

    status = NXT_SetInputMode(SENSOR_1, 'LOWSPEED_9V', 'RAWMODE', 'dontreply');
%Initialises and configures Sensor 1

%for Data retreival

    D1= COM_ReadI2C(SENSOR_1, 1,uint8(2), uint8(66));          %reads data from
distance LSB register
    D2 = COM_ReadI2C(SENSOR_1, 1,uint8(2), uint8(67));         % reads data from
distance LSB register
    range1=((double(D2)*255+double(D1))/10)/100;               % Converts the
available data into distance in meters

    status = NXT_SetInputMode(SENSOR_2, 'LOWSPEED_9V', 'RAWMODE', 'dontreply');
    D1= COM_ReadI2C(SENSOR_2, 1,uint8(4), uint8(66));
    D2 = COM_ReadI2C(SENSOR_2, 1,uint8(4), uint8(67));
    range2=((double(D2)*255+double(D1))/10)/100;

    status = NXT_SetInputMode(SENSOR_3, 'LOWSPEED_9V', 'RAWMODE', 'dontreply');
    D1= COM_ReadI2C(SENSOR_3, 1,uint8(4), uint8(66));
    D2 = COM_ReadI2C(SENSOR_3, 1,uint8(4), uint8(67));
    range3=((double(D2)*255+double(D1))/10)/100;

    status = NXT_SetInputMode(SENSOR_4, 'LOWSPEED_9V', 'RAWMODE', 'dontreply');
    D1= COM_ReadI2C(SENSOR_4, 1,uint8(2), uint8(66));
    D2 = COM_ReadI2C(SENSOR_4, 1,uint8(2), uint8(67));
    range4=((double(D2)*255+double(D1))/10)/100;

    Range = [range1 range2 range3 range4];

end

% Sensor 2 and Sensor 3 are using bus 4...this can easily be changed by
% using chnageadd executable for the NXT software
```

B6-Predict Pose

```
%this part of the project was not used since another version exists which
%which does the exact same..

function [poseNew,distance,reverseDistance] =
predict_pose(poseCurrent,velCurrent,time)

x=poseCurrent(1);        %initial x position
y=poseCurrent(2);        % initial y position
theta=poseCurrent(3);    % initial theta angle

dt=time;
L=.305;
R = velCurrent(1)/velCurrent(2);
W=velCurrent(2);

if(velCurrent(2)==0)
    Vr=velCurrent(1);
    Vl=velCurrent(1);
else

Vr = velCurrent(2)*(R+L/2);
Vl = velCurrent(2)*(R-L/2)  ;
end
```

```
distance = time*velCurrent(1);

if (Vr==Vl)                          % if robot is moving in a straight line
    D=dt*Vr;
    y1=sin(theta)*D;
    x1=cos(theta)*D;
    poseNew=[x1+x y1+y theta];
else


ICC = [(x-(R*sin(theta))),y+(R*cos(theta))];  % if robots turning

Pose= ([cos(W*dt), -sin(W*dt),0;
           sin(W*dt),cos(W*dt),0;
           0, 0, 1]* [x-ICC(1);y-ICC(2);theta])+[ICC(1);ICC(2);W*dt];
       poseNew=permute(Pose,[2,1,3]);
end
    if (distance >= 0)
    reverseDistance = 0;
    else
    reverseDistance = abs(distance);
    distance = 0;
    end
end
```

Appendix C

Datasheets (See attached)