**Project Team 9**

**Project 1 :Documentation**

**Robot Design:**

The robot design considerations made were regarding the choice of motors, wheels and a simple design approach to it.

We majorly considered a simplistic approach to the robot design and a proper weight distribution on the wheels of the robot.

**Motors Selection**:

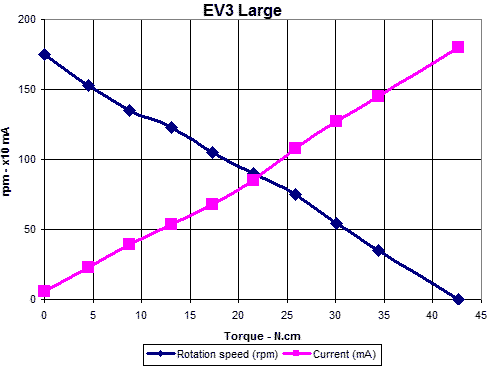
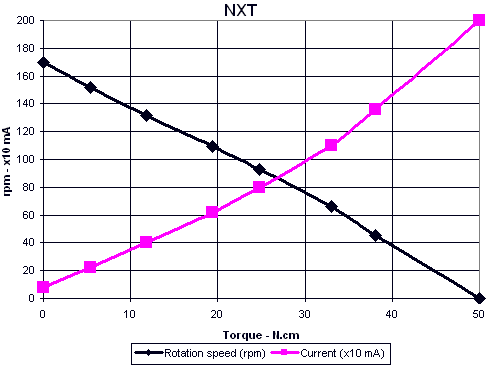
Motors we selected were Ev3 Large over NXT due to the following features:

1) NXT motor delivers a high torque thanks to its internal speed reduction gear train. Because of that, it also turns slowly and efficiency is somewhat reduced.

2) Ev3 Large motor is very similar to NXT motor, but with slightly better fixing capability (compatible with Technic frames, holes aligned with hub).

Curves showing rotation s

peed vs. torque and current consumed vs. torque, both at nominal 9V voltage for both NXT and Ev3 Large motors:



**Physical Structure Selection:**

It is a two wheel drive robot attached with a ball bearing and a steel ball.

Our robot has the following pieces assembled:

1) 2 Ev3 Large Motors

2) 2 Double bevel gears

3) 2 Low Profile Tires and 2 Hubs attached to it.

4) 1 ball bearing and a steel ball attached to support the Two wheel drive robot.

5) 4 Frames attached in order to mount the Ev3 Brick.

6) 2 Double Angular Beams mounted for aesthetic purposes.

**Some issues & our approach:**

**Odometry:**

* Finding number of rotation counts for each motor to travel 1 block distance: We were using either time or distance to move one block which was not as accurate as given the number of counts. Which was hence corrected by giving the number of rotation counts.
* Making 90 degree turns: It would either overshoot or undershoot . This was corrected by giving exact number of rotation counts which still is not perfect.
* Incrementing forward in a straight line: The robot was moving towards the right direction and this had to be corrected by jerking(rotation of 1.5 degrees for correction) towards left a bit after every one block.
* Weight Distribution: The steel ball would get stuck while making turns on some surfaces like tapes. This was corrected by concentrating the weight distribution mostly on wheels.

**Implementation of code:**

* MoveRight:  This counts the number of rotations and calculates the distance traveled with each wheal with respect to radius.  It then rotates the right wheel backwards and the left wheal forwards with the correct distances and corrects if needed (sometimes it is 1 degree of wheel rotation but it is still very accurate).
* MoveLeft: same as right but right wheel move forward and left wheel moves backwards.
* MoveForward:  This rotates the wheals with respect increments to a distances that = 1 block.  Initially it accelerates to the desired velocity in order to remove the initial jerk.  Also since there is a slight move movement to the right. it jerks the wheals so that the reference frame of the robot is 1.5 degrees to its relative left.  This makes the robot go “straight”. although after 10 blocks it is off by 1⁄4 of a block. This is compensated by a global variable. The jerk is also oscillates between 1-2 rotations for the right wheal in order to create that 1.5 degree of relative rotation.  This is done by calculating the relative degree and averaging it to the global degree. If the relative degree is greater than the global then go 1 degrees otherwise go 2 degrees.

All distances are calculated relative to the number of rotations.  We initially calculated an approximate distance and figured out how many degrees each wheel rotates.  We then altered the degrees by which it needs to rotate to go “exactly” 1 block or a distance needed for rotations.  This allows for a more accurate representation of the system and allows us to alter the distance more directly to change it for what it is.  That way if it needs to move more right then left just change the number of degrees the right wheal goes or reduce the number of degrees on the left.

**Pathfinding:**

**Algorithm used: Wavefront Pathfinding Algorithm**

**Reason for choosing this Algorithm:**

Wavefront Algorithm uses the weights of each cell in order to choose the best path to take to the goal. Thus wavefront seemed to be the simplest and most effective code possible to reach the goal. This Algorithm is not only the most efficient but also takes the shortest path possible to reach the goal.

Steps for implementing the algorithm:

* The algorithm requires a map of the entire plane the robot will traverse through. The map is usually a grid of different numbers used to depict different entities. For this instance we used these values: 99- Robot, 1- Obstacles, 0-Free space to move in, and 2-Goal/Destination.

#define NOTHING 0

#define OBSTACLE 1

#define GOAL 2

#define START 99

* The next part of the algorithm was to form a weight matric of all the free spaces on the map. For this we had to find the goal or destination on the map and then move outward from it by updating each cell with a certain value. The algorithm is called wavefront algorithm since it goes on increasing in value as it moves further thus creating a weight matrix of the free space on the entire map.
* The navigation() method in the code is used to navigate on the map using the weight of the cell states its present in. This work in a very simple way. The robot checks for the least weight value of the cells around it in all 4 directions to choose which path to follow. The least value cell around it is preferred. Once it starts moving it follows to least weight cells until it reaches the goal. This way it can choose the most efficient and the least cost path to reach the destination.