# THE A-TEAM

Bryan Lilly, Zach Lancy
Josh Wilkins

MAY 19, 2014

Design challenges are essential for engineering students throughout their education. In this instance, our team had to find and create the optimal design for a race. This race consisted of a rectangular track, on which four trials took place; the first and second were without obstacles, while the third and fourth were with obstacles in different configurations. All four trials were comprised of one lap each, the first two had to be completed within 60 seconds each, meanwhile the latter two had 75 seconds each. From the initiation to the termination of the creation of our design, our team encountered a few problems. Since the Lego NXT robotics kits are not the most advanced kits in the field of robotics, we had some problems with reliability, predictability, and repeatability.

Initially, ultrasonic sensors seemed to be the most logical solution to the design problem. We thought we could use the distance from the robot to the wall to steer it, remaining a constant distance from the wall. Before we built and tested our first theory however, the sensor was tested for accuracy and precision. As you can see, the sensors are not accurate under 25cm, but for longer distances it could have been used. Another concern of ours was about ambient noise in the room affecting our ultrasonic sensor readings, making this option less and less viable. An alternative idea we had was to force the robot against a wall and use touch sensors to detect if it was against the wall. Due to the friction caused by this design, a lot of speed was lost, making this option better than the ultrasound, but not optimal. The optimal choice we discovered was to use the drive distance function to program our robot. Through various calculations, we were able to derive a set distance in terms of the rotational amount the motor turned in degrees. With this information we were then able to program the robot to travel the defined dimensions of the track. Evidently the optimal solution from the various and conclusive results of previous tests.

| Sensor (cm) | Actual (cm) |
|---|---|
| 11 | 5 |
| 14 | 10 |
| 18 | 15 |
| 23 | 20 |
| 25 | 25 |
| 31 | 30 |
| 36 | 35 |
| 41 | 40 |

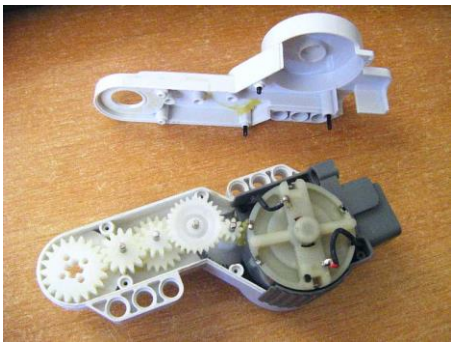
Cost was not an important aspect of this design because a kit was loaned to us by the school with all the parts needed to assemble a functional robot. However, cost was still considered in our analysis and final design choice. The most expensive parts in the Lego NXT kit were the sensors and the brick. By minimizing our design, we could achieve a cost of approximately $240. In order to do this, the front touch sensors on our design would be removed and any excess technic pieces would be removed. Nevertheless, cost was not an issue, as all the parts were already in the kit and so our design remained the way it was (non-minimized), and would amount to a total cost of $286.38, if each part were purchased individually.

The functionality of our robot was in question during the final competition. Despite countless runs without err, on competition day, our robot was only able to complete one trial out of the four. Very shortly after the competition we realized our error; that the starting position of our robot was slightly askew. If the robot had been positioned about an inch further back, then all four trials could have been completed. Our robot was definitely the fastest in the class and when compared to other robots, it was more consistent than all but 2 robots, although would have been just as consistent with the winning robot had we started the robot in the correct position each trial. In retrospect we should have tweaked our program to run efficiently from the designated starting location.

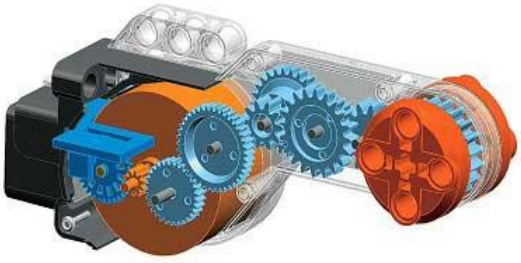
# Table of Contents

Propulsion Control System

The main aspect of the propulsion system within our design is comprised of two motors, each capable of 170rpm at no load, drawing 60mA of current on full power. Stall torque is at 50Ncm, drawing 2A when this occurs. 50Ncm is an enormous amount of torque for these little motors (resulting from the gear system illustrated), but the result of this torque is slower speeds. Thus our propulsion system was designed with a 3:1 gearing system. Each motor is attached with large 36 tooth gear which then drives a



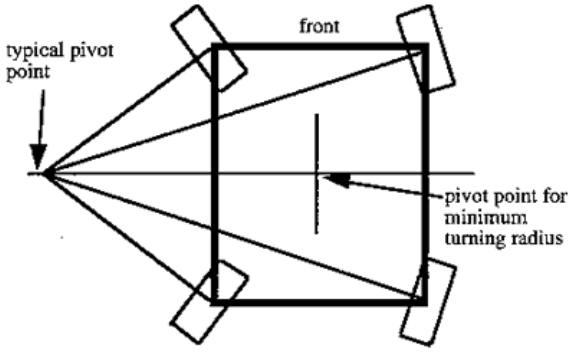
smaller 12 tooth gear attached to a wheel. This in theory would give us a speed of 510 rpm and in conjunction with a 1.625" wheel will give us a speed of 3.62 ft/s with no load. If we consider the perimeter of the inside part track (~23'), while accounting for the length of our robot (~2/3'), the time it would theoretically take for the robot to complete one lap would be about 6.35s. Of course nothing is ideal, with friction loses, and a load on our system, the efficiency of the system is greatly deprecated, resulting in a much smaller speed of approximately 1.92 ft/s. This gives us an actual time around the inside of the track to be approximately 12s. If we then take these values, we can determine the efficiency of our propulsion system by taking the theoretical value and subtracting from it the actual value, then divide it by the larger



number, multiplied by 100. This gives our propulsion system an efficiency of 47%. This seems to be a terrible efficiency, but in actuality, not many systems are ever very efficient, there are just too many variables that cannot be accounted or compensated for. Another aspect of our propulsion system is its ease of navigation, leading into the next section, the navigation control system. Construction of this system is simple: Attach the two motors to the brick, then make support for the gears and align the gears so that they mesh correctly.

Navigation Control System

The navigation control system is simply the product of our propulsion system. The fact that we designed our propulsion system with two motors allows us to control the direction of our robot through the difference in the propulsions of the motors. Put simply, the robot steers by the slowing down or speeding up of one of the motors. If the right motor is turning faster than the left one, the robot will steer left, likewise if the left motor is turning faster than the right one, the robot will steer right. Not only is this advantageous for the simple fact that is replaces alternative steering methods, but it also gives us the ability of a near zero turning radius. This illustration depicts what can be done to minimize the turning radius, with a similar navigation system as ours. With a steering system, the turn radius can only be as small as the angle of which the wheels can turn (the "typical pivot point), however with a dual motor propulsion/navigation system, the turning radius is only restricted by the width of the robot. Our robot is approximately 7" wide, meaning that we can turn on the spot 90 degrees, however the wheel on the outside of the turn has a turning radius of ~7" while the inside wheel has a near zero turning radius. Once again construction is just simply the same as the propulsion control system, by fixating the motors to the brick, aligning the gears for proper meshing, and adding support for the gear train.



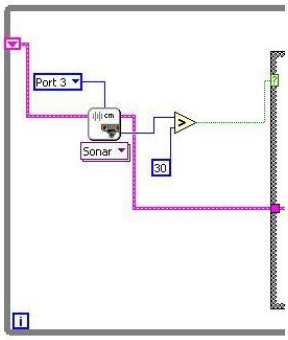
Obstacle Avoidance

In conjunction with the navigation and propulsion systems, obstacle avoidance is an essential piece of the design. Obstacle avoidance is achieved through the use of an ultrasound sensor and through the encoders in the motors of the propulsion system. The ultrasound sensor, located on the back of the robot is used to detect, in the very beginning only, from which we are then able to discern the configuration from and adjust accordingly.

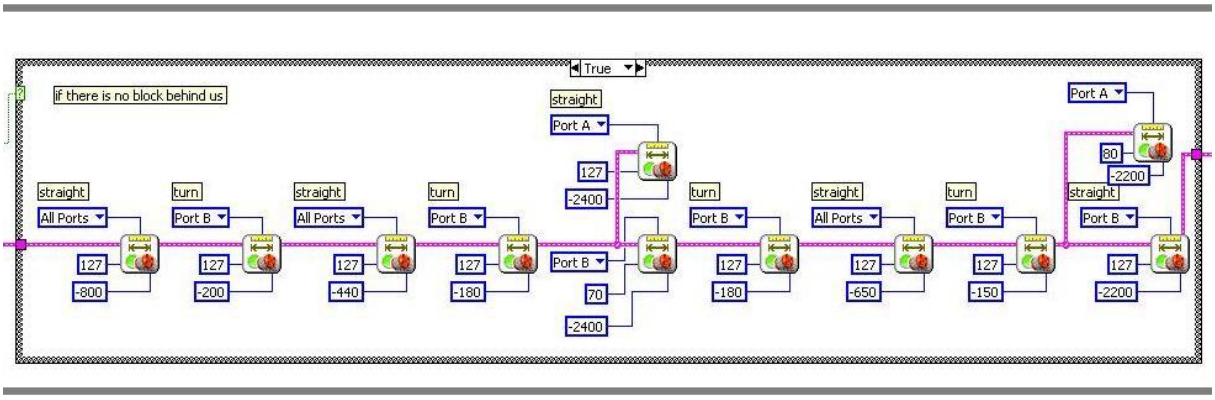
The reading from the sensor is either high or low, meaning the block is there or it isn't. If the block is there, this means we need to go the inner lane after the corner, and then the outer lane after the following corner. If the block isn't there, then we know that we need to go to the outer lane after the first corner, then the inner lane on the following corner. To determine the position of the robot, we use the encoders embedded in the motors. These encoders output the revolution of the motor in degrees, allowing for precise movements around the track. We know the distances around the track, and with a few trials, we determined that for every 8.33 degrees, we travel 1cm. From this we were able to create a reliable obstacle avoidance system. The motors are already constructed, thus no assembly or fabrication for the encoder portion and the ultrasound was simply attached to the rear end of the robot, directly onto the brick with a couple of cross blocks.

# Software Description

Though our code may appear very simple on the surface, it is a highly sophisticated piece of software comprising of parallel thread execution and digital systems integration. LabView's compiler took several minutes to build the hex file and export it to the NXT brick.




The entire program is placed within a loop structure. It has no initial conditions, and its stop condition is held false with a Boolean constant. This forces our program to loop continuously even though we only need it to do one lap/loop. The first part of our program checks the sonar sensor to see if there is any objects within 30 centimeters of the rear of the robot. We chose to do this because at the starting point the robot could check for a block behind it. If the robot knows if there is a block behind it, it can choose which appropriate route to take to avoid the obstacles. So if the sensor picks up anything greater than 30 centimeters, it will send a true value to the case structure input. If the input is a true, it selects a program to navigate the track with the obstacles set up in a left configuration set up. If the value sent forward is false, it will select the case that navigates a route for the right configuration set up.

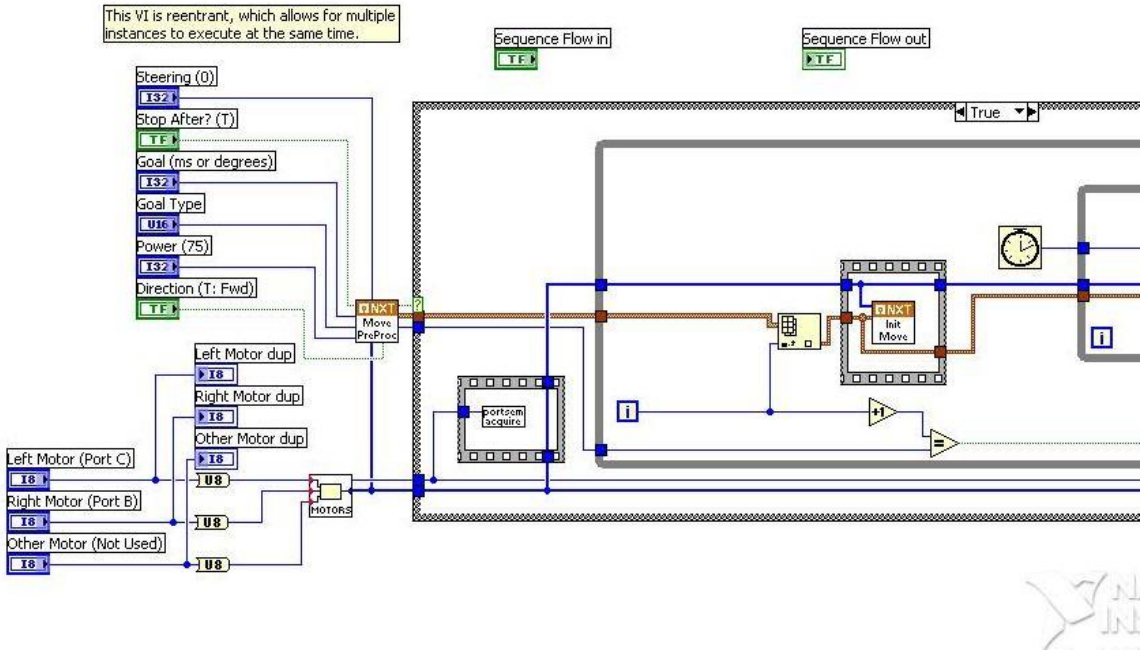


If the initial function of the program sends a true value to the case structure, this is the operation it will follow. We programmed a series of distances for both motors, and one motor if turning, to navigate. The distance values (marked above) are in units of degrees of rotation. Since we had our motors turning gears to our wheels, we had to turn negative degrees to move forward. To formulate the amount of degrees we needed to turn to travel differences, we calculated the circumference of our wheels using (pi X diameter). Then we took that value, adjusted for our gear ratio, and calculated the degrees we would need to turn in order to achieve our desired distance. Since this program is run when there is no block behind it on the starting line, it means that the first obstacle is on the inside wall, so we programmed the appropriate distances to avoid it, and the second obstacle is on the outside wall, therefore we programmed the distances to avoid that as well.

If a false value is sent forward from the initial function, a case of similar format will be run, but with distances appropriate to avoid an alternate track configuration. This approach worked well during all of our testing, but some anomaly caused us to fail three of the four trials during the in-class competition. The NXT pink wire is connected to all NXT vi's to control program flow. The pink wire is also passed back to the main loop through a shift register but this is unnecessary as our code only needs to loop once.

Each case structure contains 11 NXT Drive Distance vi's to control motor rotation rate and distance. Power is one of the variables passed into the vi, it is a signed 32bit integer who's maximum value can be 127 for maximum speed. It is unclear why a 32bit integer is used because it is passed to PreprocessMovement.vi and converted to an 8bit value. Distance is also passed in as an unsigned 32bit integer. It is defined as a goal for the Move vi. PreprocessMovement.vi then bundles our goal, goal type (distance or time), and power along with several other variables. This is sent to a loop within Move.vi to

be run until the goal is reached.




InitializeMovement.vi unbundles our cluster of variables and passes them to various NXT classes. The internal workings of these classes are not known to us because the software is closed source. The method TachoLimit is the key feature we are utilizing. We believe that this class uses a hardware interrupt and counter on each motor to keep track of encoder counts as the motor spins. Some research online has shown that the NXT "custom phone cords" pin out includes two wires for the quadrature encoder built into the motors. The NXT brick does not allow us direct access to the optical encoder on the motor but we can read the total counts as the motor spins, and we can reset that value to zero. Which leads us to believe that the "interactive" motors we used have a hardware interrupt to keeps track of the pulses. The included software manual states that we should pass in an initial value and this will be written to the motor. We then check the RunState method which reports a Boolean value. This value is either 1 meaning that the motor is still counting to our set value or 0 meaning that the counter has finished and the motor has stopped turning. Once the motor has stopped, the code moves on to the next MoveDistance.vi.

Additional research into the NXT “interactive” motors shows that directly off of the main dc motor gear is a special gear with spokes that are used for the optical encoder. There are 12 slits in the encoder wheel and the gear reduction for the encoder is 10:32. The gear reduction for the output hub is 1:48. So for one turn of the output hub we have 48*10/32 = 15 turns. This yields a resolution of 15*12 = 180 slits. If the interrupt subroutine keeps track of both rising and falling edges, this will yield a resolution of 360 transitions per revolution of the output hub. A and B signal lines are present in the NXT motor pin out, which suggests this is actually a quadrature encoder that has two sensors offset from one another so a direction of rotation can be seen. By keeping track of which pin is rising before the other we can ascertain direction of rotation. This additional sensor can double the resolution but since all the methods return and receive Integer values it is unlikely that they are counting in half degree increments. The second sensor is only used to determine count up and count down but we still have an excellent resolution of +- 1 degree.

The Move vi has a sequence that is initiated before the main loop is executed that is called PortSemaphore.Aquire.vi. We were unable to find much information on a Class called NXTSemSystemCall which includes some methods for resource allocation checks. A semaphore is a variable or abstract data type that is used for controlling access to common resources. We believe this vi is used to make sure there are no motor port conflicts during program execution. Since multiple instances of the MoveDistance vi can be ran at the same time, this can be viewed as a parallel process. Multiple sub-vi’s may want access to the same motor ports at the same time. Therefore a semaphore is used to keep track of motor port resource allocation preventing a race condition in which two vi’s are

racing to gain access to the motor port and then locking the other out. This would result in unexpected behavior and may be somewhat unpredictable making debug the problem very challenging.

# Team Dynamics

The first few days of the project were mostly testing and data logging. Josh has brought in his NXT from home to do some initial testing with gear ratios and speed while Zach worked on building another one from the school supplied parts. Josh's NXT served as our test platform throughout the three weeks before the competition.

Josh and I had witnessed last semesters 161 competition in the atrium and saw that most teams had used the ultrasonic distance sensors for a wall follow and how unreliable they were. We decided it would be best to test these sensors with a ruler to see how accurate they were. We found that they were not accurate at all below 20cm. They also had difficulty seeing such a small piece of molding on the track. The field of view on these sensors was nearly 45 degrees, this would mean that they would see the dowels and any bystanders near the track. Based on last semesters 161 competition results, they were also susceptible to noise interference. For a reliable robot we would need reliable data.

After ruling out the ultrasonic sensors, Josh and I continued the discussion on which sensors to use. The light sensors were ruled out because line follow would not help us around the obstacles and would be very difficult with a dashed line. Josh and I both have had significant experience with this track and the ways to navigate it. We decided to model our robot to be similar to team Grinder's from this semester's ENR-259.

We would use a touch sensor for our wall follow subroutines and use the distance sensor to check for the obstacle ahead of us. If the touch switch is depressed, it means that we were against the wall and should continue straight at full speed. If the switch was not pressed,

the robot would slow down the motor on that side. This would either bring the robot closer to the wall to avoid hitting the obstacle or it would force the robot around the 90 degree turn. The majority of the race is an interior wall follow. We would keep checking for the block in front of us and if seen, we would switch over to an outside wall follow for a short period of time then switch back to the interior.

This method proved to work very well, we were running laps in around 20 seconds. After some additional testing we realized that the robot was not traveling at its top speed. We concluded that this was due to an oscillation in the wall follow / turn subroutines. The robot would bounce off the wall ever so slightly that the switch would be released, this would cause the robot to enter the turn function, hit the wall and bounce off. This oscillation was not visible to the naked eye but was proven by printing "turn" to the screen every time we were in the turn function. And it explained why the robot would move so much quicker if we loaded a program that turned both motors on at full speed.

We then attempted to navigate around the track in a square pattern by turning the motors on at a set rate for a set amount of time. This method worked some of the time but required that we maintained a high battery voltage. If the voltage dropped too low the motors would spin at a lower rate and the timings would all be off.

We discussed this problem in a team meeting and concluded that we needed a better solution to the entire course. Zach suggested making use of the encoders built into the motors. Labview includes a sub-vi that turns on the motors at a set speed and reads the encoders. It then shuts off the motors after a set distance has been reached. We then spent one class

period measuring programmed distances and actual travel and found that this system was very reliable and the correlation in our data was linear. With our tire size and gear ratio, 8.3333 units sent to the VI was 1cm of actual travel. Use of the encoders would also solve motor torque issues when the batteries ran low.

We remounted the distance sensor to the rear of the robot to check for the block at the start of the race, this would determine the length of the second straight away. After another class period we had the robot programmed and calibrated to complete the entire course in 11.5 seconds with obstacles. During our final team meeting we concluded that we could have spent more time fine tuning the program but one of the biggest mistakes made by most 259 students is over-practicing the robot and making things worse. The final thing we did was split up the written report sections and oral report sections.

The most difficult part of this project was obtaining reliable sensor data with the sensor we had available to us. After some research we found that the motors contained encoders within them. These encoders would give us consistent readings across any battery voltage. Some minor testing showed that this was the most reliable sensor data that we had available to us.

The size of this team was appropriate for the assignment. Each member was assigned a specific task and accomplished their task within the given deadline. All members were readily available to help the others when needed. No team leader was specifically chosen, however Josh quickly filled that role in order to keep the project on target. Everyone's ideas and opinions were heard during testing and during meetings. There was no bickering amongst

members, which made the project fun instead of a chore. We feel that we did quite well throughout this project. As a group we worked well together and all members contributed. We managed to get third place with only one successful run. Had we had a gyroscope available to us we could have made our turns more precise. Infrared distance sensors would have also been helpful, as they have a narrow field of view.

| Bill of Materials | | | | | |
|---|---|---|---|---|---|
| **Part** | **Description** | **Supplier** | **Quantity** | **Unit Price** | **Total Cost** |
| NXT Intelligent Brick | System Controller | Lego | 1 | $149.99 | $149.99 |
| Ultrasonic Sensor | Ultrasound Distance Sensor | Lego | 1 | $33.99 | $33.99 |
| Push Sensor | Touch Sensitive Button | Lego | 2 | $19.99 | $39.98 |
| Tacho Motor | Servo Motor | Lego | 2 | $19.99 | $39.98 |
| Cables | Data Transference | Lego | 5 | $1.86 | $9.30 |
| Technic Pin | Friction Inducing Connector | Brick Owl | 35 | $0.01 | $0.35 |
| Long Technic Pin | Long Friction Inducing Connector | Brick Owl | 13 | $0.02 | $0.26 |
| Pin to Axle | Friction Based Connector to Axle | Brick Owl | 4 | $0.01 | $0.04 |
| Lego Beam 3 | 3 Hole Technic Beam | Brick Owl | 6 | $0.06 | $0.36 |
| Lego Beam 7 | 7 Hole Technic Beam | Brick Owl | 6 | $0.08 | $0.48 |
| Lego Beam 9 | 9 Hole Technic Beam | Brick Owl | 4 | $0.12 | $0.48 |
| Lego Beam 11 | 11 Hole Technic Beam | Brick Owl | 4 | $0.11 | $0.44 |
| Lego Beam 13 | 13 Hole Technic Beam | Brick Owl | 1 | $0.38 | $0.38 |
| Bent 45 Beam 3 x 3.8 x 7 | Technic L beam with 2 45 angles | Brick Owl | 2 | $0.06 | $0.12 |
| Bent 90 Beam 3 x 5 | 3 Beam Merged with a 5 Beam at 90 | Brick Owl | 2 | $0.07 | $0.14 |
| Wedge Belt Wheel | Narrow Wheel or Pulley Belt | Brick Owl | 4 | $0.10 | $0.40 |
| Lego Angle Connector #2 | Axle to Axle Connector - Straight | Brick Owl | 4 | $0.04 | $0.16 |
| Lego Angle Connector #6 | Axle to Axle Connector - Bent 90 | Brick Owl | 2 | $0.06 | $0.12 |
| Double Bevel Gear Z36 | Conical Bevel Gear 36 tooth | Brick Owl | 2 | $0.86 | $1.72 |
| Double Bevel Gear Z12 | Conical Bevel Gear 12 tooth | Brick Owl | 2 | $0.08 | $0.16 |
| Technic Bushing | Spacer/Technic Bushing | Brick Owl | 8 | $0.02 | $0.16 |
| 1/2 Technic Bushing | Small Spacer/Technic Bushing | Brick Owl | 4 | $0.04 | $0.16 |
| Wheel 49.6 x 28 VR Assembly | 1.625" Diameter Rubber Wheels | Brick Owl | 2 | $0.67 | $1.34 |
| Technic Cross Block 1 x 3 | H - Shaped Connector Pins | Brick Owl | 8 | $0.14 | $1.12 |
| Cross Block Bent 90 | L -Shaped Connector Pins | Brick Owl | 6 | $0.75 | $4.50 |
| Axle 5 | 3.9cm Technic Axle | Brick Owl | 5 | $0.03 | $0.15 |
| Axle 6 | 4.7cm Technic Axle | Brick Owl | 1 | $0.02 | $0.02 |
| Axle 10 | 7.9cm Technic Axle | Brick Owl | 2 | $0.04 | $0.08 |
| | | Total: | 138 | $229.59 | $286.38 |

# Competition and Results

This design competition is comprised of a rectangular track approximately 12' x 4' in which teams must design a Lego NXT robot that can navigate around the track. Each teams' robot must also fit into a box that is 8" x 12" x 10" high. The order in which teams compete is chosen randomly and once your team is called, you then have one minute to start. During this time, it is the responsibility of the team to inspect the track after which they must start behind the joint of the plywood and travel in a counterclockwise direction. The time starts when the judges give the command, at which point the team can activate only one button and may not communicate in any way with the robot thereafter. All parts of the robot must then remain inside the track, no dowels can be knocked over, and the blocks cannot be moved more than an inch. After each trail, a team can make changes or repairs to their robot, but cannot test there robots once the competition has begun.

Scoring is based on the speed of the robot in conjunction with the configuration of the track. Each trial has a time constraint; Trials 1 and 2 have a 60 second time limit each, while trials 3 and 4 have a 75 second time limit. For trials 1 and 2, the teams' robot must simply just navigate around the track, however in trials 3 and 4, the robot must navigate the track as well as avoid preconfigured obstacles. For all four trials, the score awarded to the robot is the time limit for that trial minus the time it took for the robot to complete one lap. If the robot failed to complete the lap however, a score of 0 is rewarded. The final score awarded to the team is then all the points summed up from the four trials and the team with the highest score at the end wins.

## Class Competition Results

| Team Names | Trial 1 (sec) | Trial 1 Points | Trial 2 (sec) | Trial 2 Points | Trial 3 (sec) | Trial 3 Points | Trial 4 (sec) | Trial 4 Points | Overall Score | Rank |
|---|---|---|---|---|---|---|---|---|---|---|
| Jonathan Laura, Nathan | 25.7 | 34.3 | 25.6 | 34.4 | 29.4 | 45.6 | 27.6 | 47.4 | 161.7 | 1 |
| Ryan, Matt O, Matt | 27.3 | 32.7 | 27.1 | 32.9 | 70 | 5 | 0 | 0 | 70.6 | 2 |
| Josh, Zack Brian | 11.6 | 48.4 | 0 | 0 | 0 | 0 | 0 | 0 | 48.4 | 3 |
| Alex S, Kenny Jamaal J | 0 | 0 | 12.3 | 47.7 | 0 | 0 | 0 | 0 | 47.7 | 4 |
| Sarah L, Alex G Kristen E | 37.3 | 22.7 | 35.4 | 24.6 | 0 | 0 | 0 | 0 | 47.3 | 5 |
| Chai, Hope, Jacky, Shiva | 19.3 | 40.7 | 0 | 0 | 0 | 0 | 0 | 0 | 40.7 | 6 |
| Ahmad, Zach Ali Z | 38.8 | 21.2 | 41.4 | 18.6 | 0 | 0 | 0 | 0 | 39.8 | 7 |
| Joshua, Greg, Connor, Joel | 40.6 | 19.4 | 0 | 0 | 0 | 0 | 0 | 0 | 19.4 | 8 |

# Project Assessment

Our robot performed perfectly during testing; having three failed runs in a row during the competition was an unusual anomaly. It was well designed mechanically, and the code was simple enough to have good repeatability. Our robot was the fastest in our class, but there was another team that was not far behind us. Most of the other teams chose to use a direct drive off of the gear boxes or a 1:1 gear ratio. Speed was not a hindrance to our performance. However, it's possible that ramping our speed could have helped with wheel slippage.

Most teams used the ultrasonic distance sensors for wall following. As we expected this proved to be a difficult task. The ultrasonic sensors have too wide of a field of view to reliably sense a ¾" tall piece of molding. Several of the robots lost their readings and drove over the wall resulting in a disqualification. Some teams had successful runs with the ultrasonic sensor but they spent a significant amount of time coding and testing. Additional man hours would have raised the cost of our robot. One option for better readings with the ultrasonic sensors was to mount them to the robot as far from the wall as possible. This was successfully accomplished by another team that went on as an exhibition robot in the ENR-259 competition.

One feature we really liked was stopping and backing up once the obstacle was detected. This was done by one of Professor Kumar's morning teams, they did not win their class competition but they had a fast robot and went on to win second place against ENR-259 as an exhibition robot. Stopping was not something that our team had thought of and we would consider that for future projects.

We feel that our use of encoders was our greatest strength and would encourage other teams to use them in the future. The encoders were the most reliable sensors on the robot, and not many people knew they were there. Coupling the encoders with a set-back ultrasonic would have been the

ideal solution. Most teams were impressed with our speed, gearing up to 1:3 would have helped several of the teams that had good wall follow routines but a slow robot.

The savings in man hours is why our robot was one of the most cost effective in the class. We had minimal parts and we didn't spend an excessive amount of time coding and testing. Any testing we did was necessary to determine which sensors would be the best for us. We did waste some time with our initial touch sensor approach. Had we not tried that route, we could have shaved 20-30% off of the labor on the project. However we feel that it was a valuable learning experience and if we could have spent more time on the project, we might have integrated the touch sensor back in for slightly more external real-world feedback.