ECSE-211
Design Principles and Methods - Lab 5
Section 001

Team 12 - HOLE
Amelia Cui - 260824815
Aiden Gerkis - 260827581
Georgiy Danylenko - 260916341
Taha Mohamed Lazreg - 260923792
David Deng - 260838166
Maxens Destiné - 260926906

Mcgill University
Department of Electrical and Computer Engineering
Fall 2020

Navigation and Torque Measurement: Design Overview and Evaluation

01) Design evaluation

Design Workflow

This lab was completed by a team of six people, which required more collaboration than in previous labs. Knowing that we decided that assigning tasks for our members to complete on their own time was better than holding group work sessions. From lab four we had developed a capabilities and availability document for our team. We used this document, coupled with several preliminary design meetings, to create a distribution of tasks that sought to optimize productivity by assigning each team member to areas of work that best matched their capabilities.

Table 1: Teams and their members related to specific areas of work.

	Hardware Team	Software Team -- Navigation	Software Team --Torque measurement	Documentation / Presentation Team
Members	Maxens, George	Amelia, Taha, David	Aidan	George, Amelia, Aidan
Responsibilities	- Design the hardware structure for the robot - Reuse the localization code in lab3	- Implement simple navigation - Implement a more generalized navigation that can avoid obstacles and push blocks - Implement JUnit testing for simple navigation	- Get the torque measurements from both motors - Determine the heaviest block - Integrate with navigation	- Make presentation slides - Write up reports

Table 2: The lab 5 timeline

Legend

- ■ - Due date
- ■ - Task Completion Deadline

Thursday October 15th	Friday October 16th	Saturday October 17th
- Robot Model due	- Minimal Angle code due - Old localization code tested	

Sunday October 18th	Monday October 19th	Tuesday October 20th	Wednesday October 21st	Thursday October 22nd	Friday October 23rd	Saturday October 24th
	- Pre-programmed navigation due - Torque Measurement code due	Work-In-Progress Presentation				

Sunday October 25th	Monday October 26th	Tuesday October 27th	Wednesday October 28th	Thursday October 29th	Friday October 30th
- General navigation code due	- Code integrated - Test cases ran	Demo Attempt 1	Demo Attempt 2 Code Submission		Report Submission

The above distribution of tasks was meant to organize and optimize work on specific tasks, but team members were also given the flexibility to contribute to other tasks when needed. We found this method of task distribution very useful because it allowed us to focus on certain tasks while maintaining a proactive design process. In our timeline certain tasks, such as navigation implementation and torque measurement, were allocated more time over others. These software tasks used up the majority of our allocated time, while the hardware design was done, by necessity, within the first few days.

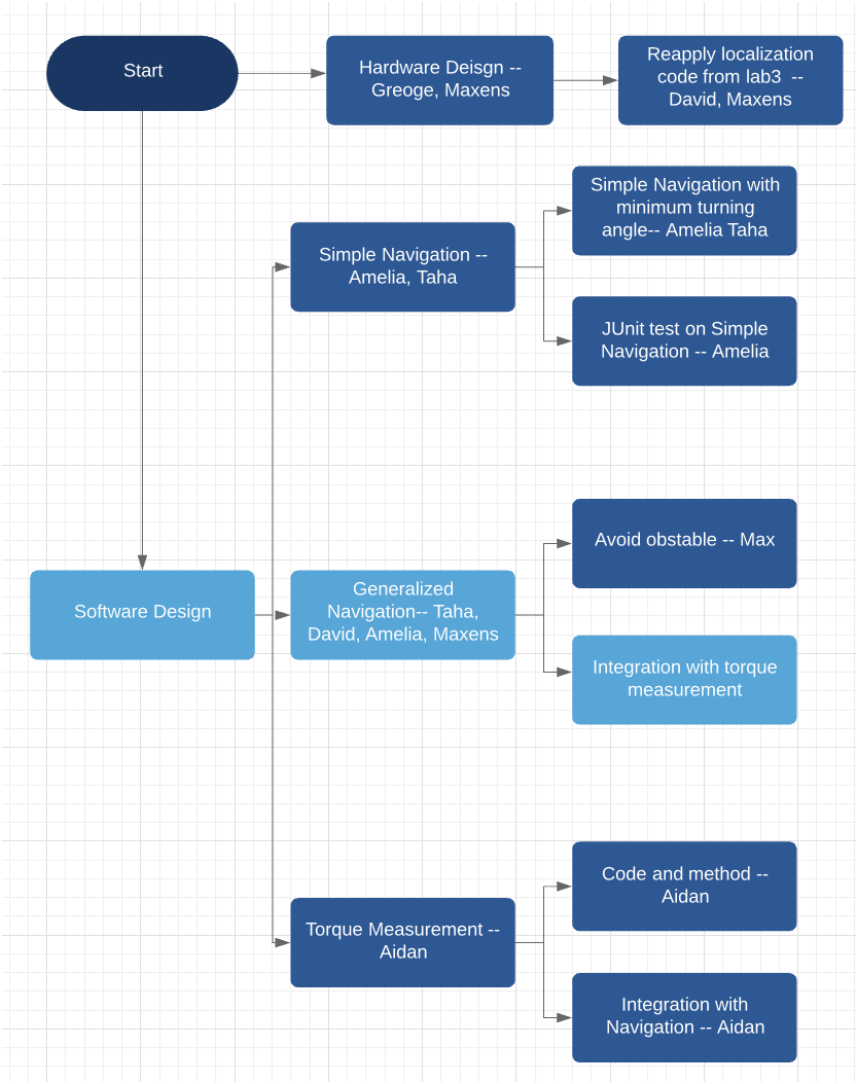

Figure 1: Design Workflow

Hardware Design

Before starting any significant work on the software design our team required a working mechanical model to test our software on. From inspection of the given maps, we realized that we could not reuse a previous model, as the mechanical requirements differed noticeably from those of previous labs. This lab's task required our robot to have a front-facing pusher arm and a much smaller turning radius.

Our robot was expected to navigate between a wall and a block and turn while avoiding collisions. From a preliminary brainstorming session, we developed a multitude of sketches as well as three preliminary mechanical models that required further validation. Additionally, after a few concept designs of differently angled pushers, we decided that a straight forward pusher, as seen in figure 1, would be the most optimal solution for block pushing.

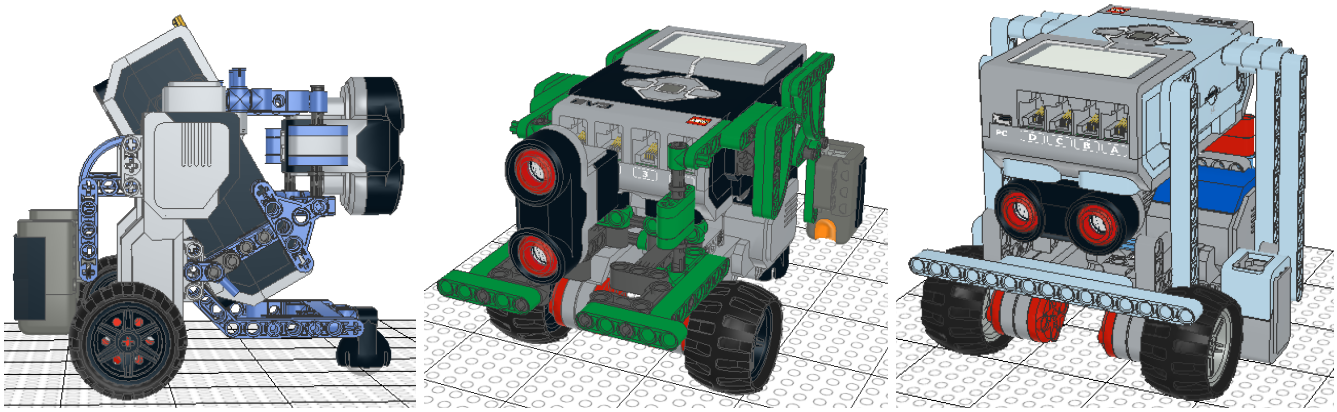

Figure 2: Our three preliminary model designs from left to right: Beans, Tank and TallBoi

The first design had already been suggested in the potential improvements for the previous labs. Its goal, by the means of a standing model, was to reduce the turning radius. Unfortunately, it proved to be a little too wide to avoid collisions. Our second model, Tank, had in mind the goal of being compact and effective. By placing the motors directly under the EV3 brick, we developed the narrowest possible model. Although it successfully turned in confined spaces it often came too close to collision, and we decided against using it. Finally, our third, and favoured, model sought to implement the best aspects of the previous two, while remaining effective and mobile. The concept was to increase compactness by bringing every component as close as possible to the brick's center. This proved to be a successful solution, as TallBoi's size and small turning radius allowed it to maneuver in confined spaces without any risk of collision.

Software Design

Figure 3 provides an overview of the software workflow. Before the demo, we were able to test all four maps with simple navigation and torque measurement. Additionally, in order to achieve the bonus marks, we worked on generalizing our navigation code to detect and avoid obstacles when travelling. However, due to the time constraints, we were not able to integrate torque measurement with the generalized method before our demo. We plan to integrate this code for the final project.

There are three main parts to the software implementation for this lab. First, in order to localize the robot, we decided to reuse Max's and David's code from lab three, since their design was observed to have the most accurate performance after several tests. The localization code implemented used one ultrasonic sensor and two colour sensors to localize the robot.

The simple navigation code is used to move the robot to a certain destination by turning through the minimal angle and travelling to the desired waypoint, it assumes there are no obstacles in between. This behaviour was achieved by implementing the moveStraightFor and turnTo methods. We also implemented the relocalize method to reduce errors in our robot's navigation. This method is called after navigation to a waypoint is complete. It uses both light sensors to realign the robot with the gridlines and correct any odometry error that may have been introduced. To determine which block is heaviest the motor torques are sampled while a block is being pushed, and the average of these values is calculated.

The average torque values are then compared to determine the heaviest block. The torque sampling code is run sequentially with the navigation code by forcing the rotate method to return before movement is complete.

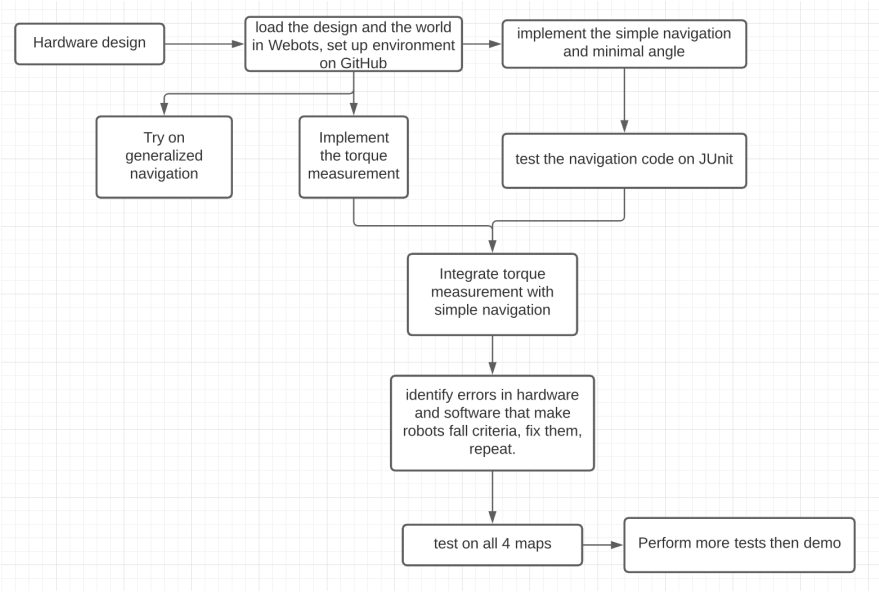

Figure 3: Software design workflow flowchart.

In order to reduce the amount of debugging required in Webots, we decided to test our navigation code on Eclipse prior to running on Webots. Several JUnit tests were implemented to ensure that the robot turned through the minimal angle, and travelled the shortest distance to each waypoint.

In the end, as stated earlier, we ended up "hard coding" the paths for each map as our general navigation code was not ready in time for the demo. As a result, the code for each of the 4 maps differs quite a bit. However, they all follow the same sequence of operations: go to block, relocalise, push block, go back 1 square, relocalise and repeat for remaining blocks. Here is a flowchart representing this sequence of events:

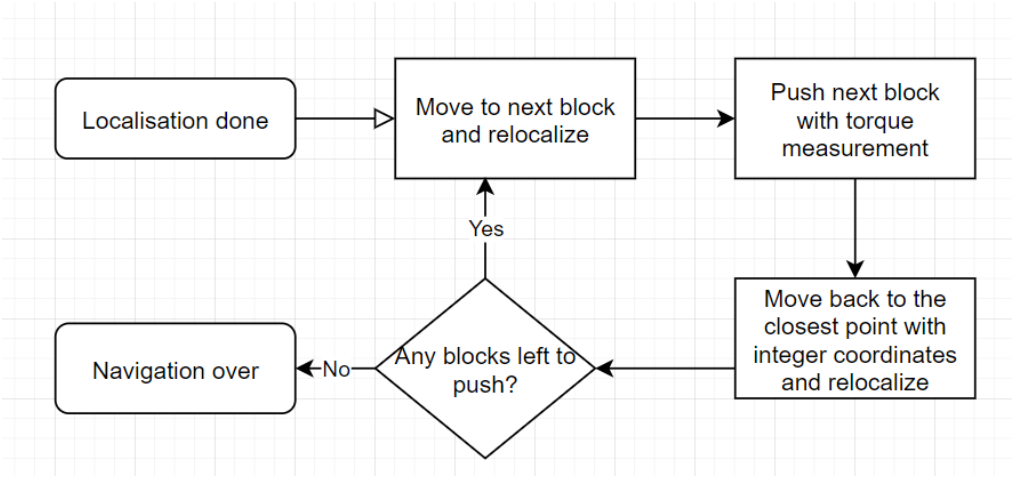

Figure 4: Software flowchart.

Additionally, we decided to add a UML class diagram to illustrate the relationship between our different classes. While not very relevant for this lab, it becomes more important for the final project so we decided to include it as comparative data for our final design.

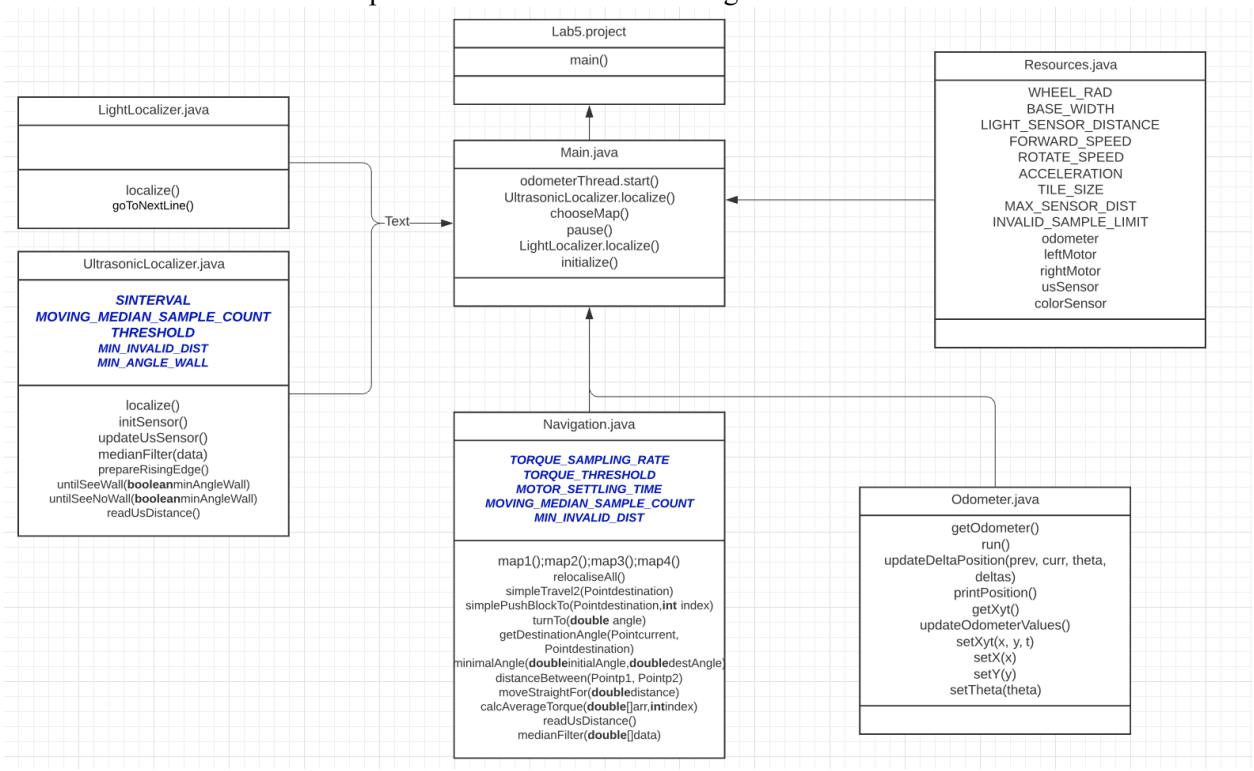

Figure 5: Software design class diagram.

2) Test Data

Table 3: Navigation Test Data

Trial number	Map number	Desired Destination (m)	Actual Destination (m)
1	1	(2.1116, 0.6096)	(2.10943, 0.611924)
2	1	(2.1116, 0.6096)	(2.10955, 0.612044)
3	2	(0.3268, 0.6096)	(0.336819, 0.613247)
4	2	(0.3268, 0.6096)	(0.334219, 0.612007)
5	3	(1.8288,0.9364)	(1.83132, 0.943643)
6	3	(1.8288,0.9364)	(1.83133, 0.943929)

Torque Test Data.xlsx *

**Data is presented in a separate excel sheet due to the amount of data collected*

3) Test Analysis

Table 4: Navigation Test Error Analysis

Trial number	**(X_d, Y_d)**	**(X_f, Y_f)**	**ε**
1	(2.1116, 0.6096)	(2.10943, 0.611924)	0.0031796
2	(2.1116, 0.6096)	(2.10955, 0.612044)	0.00318993
3	(0.3268, 0.6096)	(0.336819, 0.613247)	0.01066213
4	(0.3268, 0.6096)	(0.334219, 0.612007)	0.0077997
5	(1.8288,0.9364)	(1.83132, 0.943643)	0.00766886
6	(1.8288,0.9364)	(1.83133, 0.943929)	0.00794272

Table 5: Error Analysis

ε analysis	
Mean (μ)	0.0067
Standard Deviation (σ)	0.0027

Formulas used:

$$\varepsilon = \sqrt{(Xd - Xf)^2 + (Yd - Yf)^2}$$

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}.$$

$$\mu = \frac{1}{N} * (X_1 + \ldots + X_n)$$

Table 6: Torque Test Data Analysis

Torque Trial Data Analysis						
Trial #	1	2	3	4	5	6
Block Mass	0.5 kg	0.75 kg	1.5 kg	2.0 kg	2.5 kg	3.0 kg
Average Torque	0.551878613	0.585806886	0.662364125	0.7220792	0.7728076	0.8333369
Standard Deviation	0.004086603	0.003844647	0.003460907	0.0031404	0.0027204	0.0025163
Drift (Final Torque - Initial Torque)	0.013485184	0.014598488	0.015847606	0.0118097	0.0110897	0.0091432

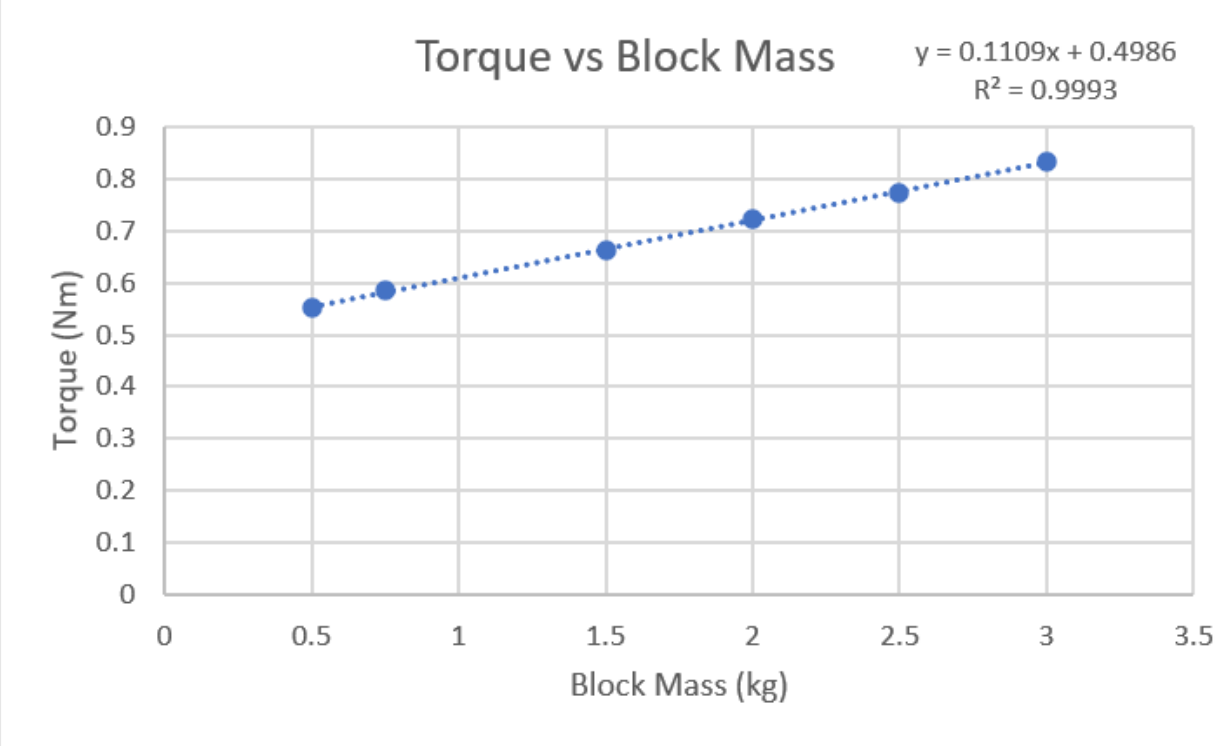

Figure 6: Average torque vs. Block mass over the six trials

4) Observations and Conclusions

Our navigation code determines how far the robot must move to reach the destination based on its current position, as measured by the odometer. The robot then moves to that position using the moveStraightFor() method. Since the navigation class acts as a high-level class which largely performs calculations based on input data, and relies on other classes and methods to interact with the robot, no errors should be caused by the navigation class. The two main sources of potential error in this process are in the current position measurement or in the moveStraightFor() method. An error in current position measurement would arise from the Odometer class, while an error in the moveStraightFor() method would arise from longitudinal wheel slip. Since the robot is being simulated within an 'ideal' environment wheel slip is likely minimal, and most of the error present is presumably generated from the odometer class.

The navigation controller is seen to be very accurate, as the mean error for the six trials is significantly smaller than the distance travelled in these tests, differing by three orders of magnitude. Through testing the navigation code, we determined that the odometer reading becomes inaccurate after our robot has moved straight for three tiles. Therefore, we decided to re-localize our robot after it had moved a distance equivalent to three tiles. Additionally, in order to push the block at a more accurate angle, we relocalize the robot before it begins pushing a block. Each time our robot localizes we are reducing accumulated error by adjusting the real position of our robot to match the desired position. We also set the odometer values each time we localize, which eliminates accumulated odometer error. By localizing often we ensure that the accumulated error does not exceed the maximum range which we can

correct for using localization (half the width of a tile). Localizing often also reduces the chances that the robot will collide with an unexpected object due to an error in its position. However, localizing is a time-intensive process, and localizing at each waypoint may increase the run-time of our code over the five-minute limit.

From the calculations of standard deviation for each of the trials, we see that the torque measurements fluctuate very little. As block mass increases we see that the standard deviation of the measured torque values decreases. We can eliminate environmental variations, such as variations in the coefficient of friction, as the cause of this trend, as they will have the same effect on the robot regardless of mass. The most likely cause of this deviation is sensor drift. The amount of drift experienced during each trial was characterized by subtracting the initial measured torque from the final measured torque. There is a clear decrease in sensor drift as the block mass decreases, and this would account for the decrease in standard deviation that was observed.

The relationship between block mass and torque seems to be linear in the range of values we tested based on inspection of the plot of Torque vs. Block Mass. To verify this assumption the equation for a trend line was calculated using excel. The resulting linear trend line had an R^2 value of 0.9993, so this assumption seems to be valid. From the equation for the torque required to move a mass m ($T = m*g*\mu*r$) it is clear that the expected relationship between torque and block mass should be linear, so the observed trend matches the predicted relationship.

From the lego website the stall torque of a large EV3 motor is 40 Nm, let's assume this is equivalent to the maximum continuous motor torque. Using the trendline calculated by excel we have that the maximum block mass must satisfy the equation $40 = 0.1109*m + 0.4986$. Solving this equation for mass yields m = 39..9, so the maximum mass our robot can move is 39.9 kg with a torque of 40 Nm.

5) Further Improvements

One possible source of error that could be reduced is the error induced in the torque measurement by sensor drift. Although drift does not affect our current application of the torque measurement code it could cause problems in an application where the exact weight of each block needed to be determined. To reduce the impact of this source of error we would first characterize the sensor drift by performing a series of tests, pushing blocks of different weights, and pushing blocks of the same weight sequentially. From these tests, we could use statistical analysis to develop a model of the sensor drift and code a software implementation of this model to eliminate drift from the measured torque values. This would result in a more accurate average torque measurement and would allow us to better analyze the effect of block mass on torque measurement variations.

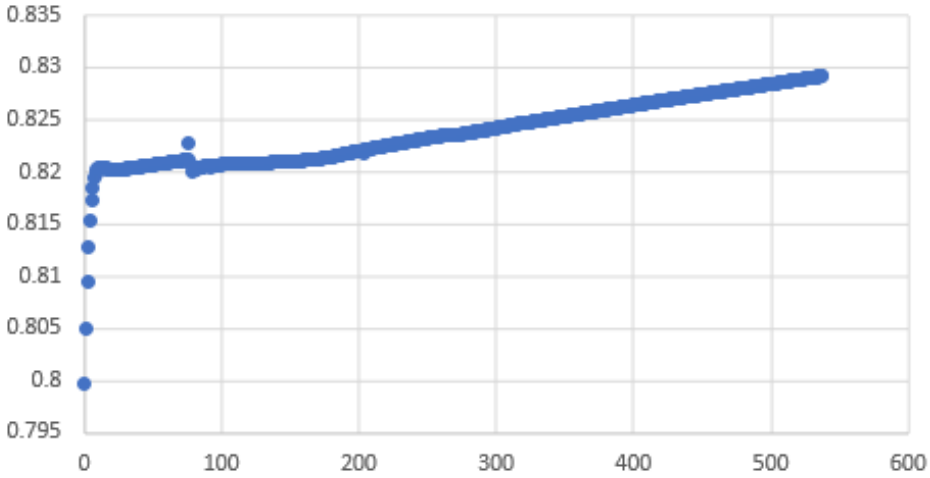

Figure 6: The Torque vs. Time graph for this block clearly illustrates sensor drift

Another possible improvement would be using a touch sensor to determine when our robot is pushing a block. Currently, our robot knows when it is pushing a block because a unique method, pushBlockTo(), is called. It can identify when it stops pushing a block by detecting a change in the measured torque that is greater than the threshold value. This method is very susceptible to errors in the torque measurement, as an outlier value can end the method early. By implementing a touch sensor facing in the direction of motion our robot would be able to more accurately determine when it is pushing a block and would result in a more robust torque measurement algorithm.

One possible navigation improvement for the final project would be to implement a method that automatically relocalizes our robot while navigating to a specific point. For this lab, we hardcoded the relocalization points along the path. However, for the final project navigation will be semi-autonomous, not hardcoded, so a method that allows the robot to self determine when to relocalize during navigation would be useful to increase navigation accuracy. This will also simplify the code significantly, making the code more readable as well as decreasing navigation error.