Summary

The lab began by familiarizing ourselves with the BotBarduino. The *M452Introswitch* code was used again with the same sequence of flashing LEDs as the previous lab and then modified to incorporate feedback received from the bumper switches. After this the LynxBot was calibrated first by using *M452Lab2UpDown*. The robot was placed on a pedestal and it's speed was adjusted to find the dead zone of the motors between their direction of rotation using the PBA, PBB and PBC buttons on the board. Using the Arduino code *M452Lab2Forward*, the maximum forward and reverse speeds of the motors were determined by gradually increasing the input variables and recording the distance traveled forward and backwards. Each tested input speed had three distance trials before an average was taken. These results are discussed further in section Results.

After calibration, the navigation task was coded by our team by modifying the *void loop()* of the *M452Lab2Forward*. This section of the lab had three main components in the code that each team member was assigned, moving forward and detecting the collision, rotating the robot, and the LED sequence at each stage. The moving forward code was done by using the *runMotorsForward* function and an if statement using a *digitalRead* command to check if the bumpers switch is set to *HIGH* from a collision. When the switch is triggered the turning sequence begins with the robot driving quickly to square itself with the wall, reversing and then turning 90 degrees before adding 1 count to the wall hit counter, and going forwards again. The LED sequence of the robot needed to follow a pattern based on the robot's behavior. When the program is ready, the green LED flashes, moving forward the green LED is solid, when a wall is hit the red LED is solid, when reversing the yellow LED is solid, and lastly, when the robot returns to the starting position, all LEDs are flashing.

Once the navigation path was complete, a trial-and-error method of testing was used to program the correct duration of turning time. If the LynxBot under turned, the turning program was edited to run for longer. If it overturned, the program was shortened. This was done until the LynxBot was able to complete four turns and return back to the home position.

Design

section with description of design and two photos (1 st = close-up of robot showing bumper placement from group 16 and 2 nd = the testing setup, as in Fig. 1).

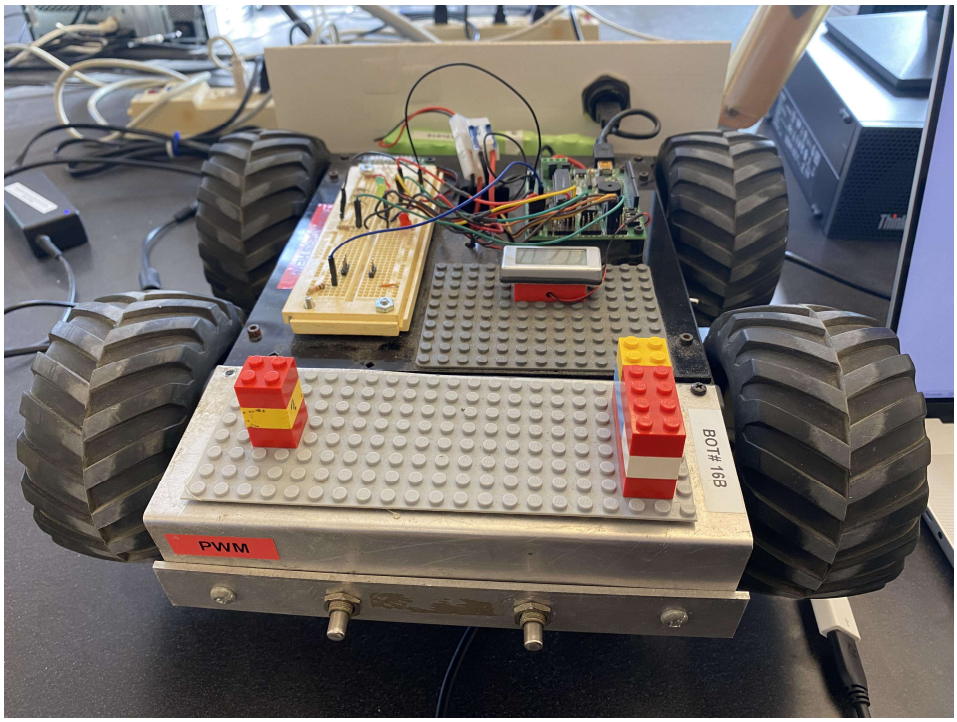


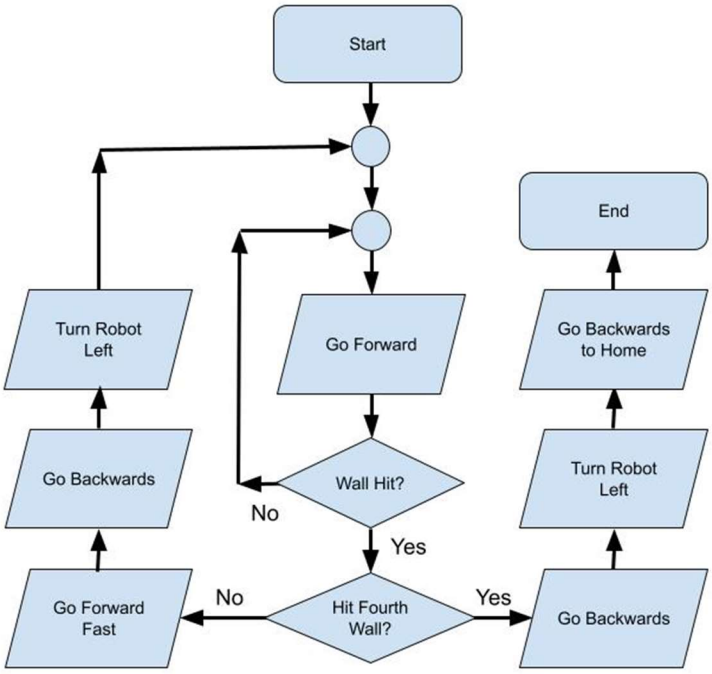
Figure 1: Placement of bumpers on LynxBot. This photo is from Group #16 as the teams photo was unusable; however, the placement of bumpers is still the same as the LynxBox that was used in this lab.

Figure 2: Testing circuit. The LynxBot starts between the red lines then goes in a square pattern before reversing back between the two red lines.

Program

section with flowchart for Group#Lab2BumpTurn and text that explains the flowchart and the sequence of operations. Flowchart should provide more detail than seen in Fig. 3, for
example what “ReversePivotPause” and “GoHome” are about. But not too much detail.

The figure below shows the operating sequence that the car uses in the bump and go trial.



The car beings to move forward using *runMotorsForward* until a wall is hit, triggering the bumper sensors on the front of the car. When a wall is hit the car accelerates to straighten itself with the wall before backing up. Once backed up, the car rotates 90 degrees to the left, and then proceeds to go forward again. In the code there is a *do/while* loop to repeat the forward and turning sequence until the wall has been hit 4 times. After the fourth wall is hit, the robot reverses, turns left, and then reverses into the home position.

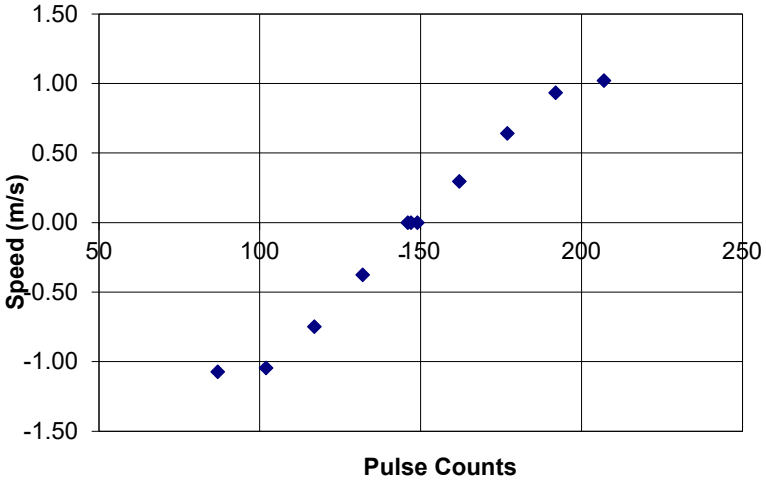
Results

which must include:
i) Calibration results (Table 1 and Table 2 and plot using the template) with supporting text.

The table below shoes the raw speed data found using the *M452Lab2Forward* code and recording the distance traveled forwards and backwards. As expected, as delta increases so does the speed of the motors, but as delta increases the speed begins to encounter an asymptote as the physical limit of the motors are approached.

	Distance (cm)						Average Speed (m/s)	
	1st test		2nd test		3rd test		m/s=distance cm/100	
delta (+/-)	forward	back	forward	back	forward	back	forward	back
60	102	-106	101	-109	103	-107	1.02	-1.07
45	92	-104	93	-104.5	95	-105	0.93	-1.05
30	63.5	-75.5	65	-75	64	-74.5	0.64	-0.75
15	28.5	-38.5	30	-38	30	-36.5	0.30	-0.38

In the table above, the average distance travelled was found and plotted as a function of Pulse Counts to create the following figure. As mentioned above, an asymptote can be seen on the graph as the speed plateaus just above ±1m/s.



The following table contains the calibration data found using the *M452Lab2UpDown* code. The maximum forward and reverse speeds were found here as well as the dead zone of the motors. The pulse counts at maximum forward and reverse speeds using the *M452Lab2UpDown* code found are very close to the pulse counts 60 pulses away from the dead zone.

	Pulse Counts			Speed
Comment	Part a)	Part b)	delta	(m/s)
start point	230			
max forward	216			
		207	60	1.02
		192	45	0.93
		177	30	0.64
		162	15	0.30
upper for stop		149		0.00
deadzone		147	0	0.00
lower for stop		146		0.00
		132	-15	-0.38
		117	-30	-0.75
		102	-45	-1.05
		87	-60	-1.07
max backward	78			
start point	70			

ii) Answer to the question: “What is the biggest source of the deadzone seen in Fig. 2 and why does it vary so much between the LynxBots? Cite a URL to support your answer.

The motors are controlled by a motor controller which operates on the PWM signal from 0 – 255 (0V - 5V). The controller must scale the signal from the 5V input to the 12V output to the motor. The deadzone is increased in the scaling of the voltage – the scaled voltage provides a larger interval for the deadzone. The varied deadzones between LynxBots come from the error in the motor controller to scale the voltage and varying PWM signals from the Arduino controller.

L298N Motor Driver - Arduino Interface, How It Works, Codes, Schematics (howtomechatronics.com)

iii) Discussion on what didn’t work and what did work, as you developed your code.

While developing code, the team realized that the LynxBot would not always make a perfect 90-degree turn. This led to an off-centered contact when bumping into the next wall. To straighten the LynxBot, the team decided to run the motors at extra high speeds after hitting the wall. This would push both sides of the Bot to be square with the wall in contact. This method of running the motors at higher speeds was also used when the robot returns to its home position.

Initially, the team tried to implement a closed loop control if the LynxBot ever got stuck on the side of the wall. This code did not work. It was later taken out.

Appendix

section with listing of Group#Lab2BumpTurn (no need to highlight changes).
As always, before submitting your report, please review the marking rubric for Lab #2 as posted to onQ