Thymio Karting

Experimental Robotics

Authors: Bella Arsenault - B00918242, Aaron Mai - B00924036, Tristan Seely - B00898894, Evan Thompson - B00929818, Harshil Varia- B00919242   Robot 9

Abstract:

The main objective of this project was to reach the end of a hurdle course while following a path and avoiding obstacles along the way; during this project, our group collaborated and came up with a solution that could successfully perform the objective. The only limitation of our code was that it was unable to follow the course in reverse order, where the robot could not recover from the lost state. This report will introduce key concepts about Aseba and the Thymio robot, outline our process for the project, the results, and how we would improve our code.

Introduction:

Robotics is a broad field that varies from being used in aerospace projects to consumer-based products. This report will outline the use of the educational robot Thymio II and the problems it was tasked to solve in Experimental Robotics. In this midterm project, we were tasked with coding a solution to a hurdles route given to us; our robot had to navigate a winding black line of tape, approximately 1.5 - 2 cm thick, as well as sense and maneuver around five duplo blocks centered on the tape line and placed around 20 cm away from each other [1].

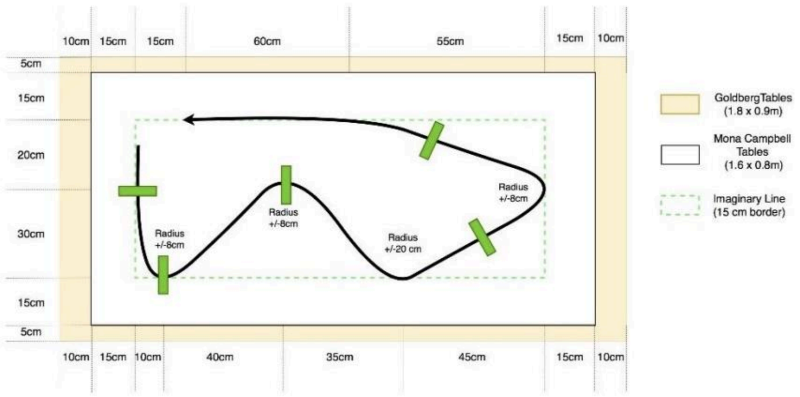


Fig. 1. The track [1]

Two of the main problems that the Thymio II robot had to overcome for this project were getting around the duplo blocks and returning to the line, as well as returning to the line around tight corners. Once the robot had a program where it could adequately navigate around the five duplo blocks, return to the line, and complete the course, the robot would compete with other groups enrolled in CSCI 1108 to see whose robot could complete the course with the fastest time. Our robot was able to complete two of the three courses successfully, whereas, with one of the routes, our code needed to be improved to deal with the variation in the track. To complete the course successfully, the completion time had to be under a minute and a half. This report will outline what the Thymio II robot is, how it is programmed in the ASEBA programming language and the limitations that arise, as well as our group's approach to the problem, the solution that was designed, the results that came out of the competition and finally what we would do differently if we had the chance to redo this project.

Background:

This project makes use of the Thymio II robot and the ASEBA programming language, an open-source robot and language designed by two universities in Switzerland: EPFL (Swiss Federal Institute of Technology Lausanne) and ECAL (University of Art and Design Lausanne), produced by Mobsya [2]. The robot is designed for educational purposes, promoting education for programming robots at all levels. ASEBA uses an event-driven architecture [3]. ASEBA is offered in 4 different forms for the public to program the Thymio II robots. There is a visual programming option, a block coding option, a scratch programming option, and a text programming option [4]. This project makes use of Aseba Studio, which supports text-based programming. The Thymio II robot can be seen below in figure 2.

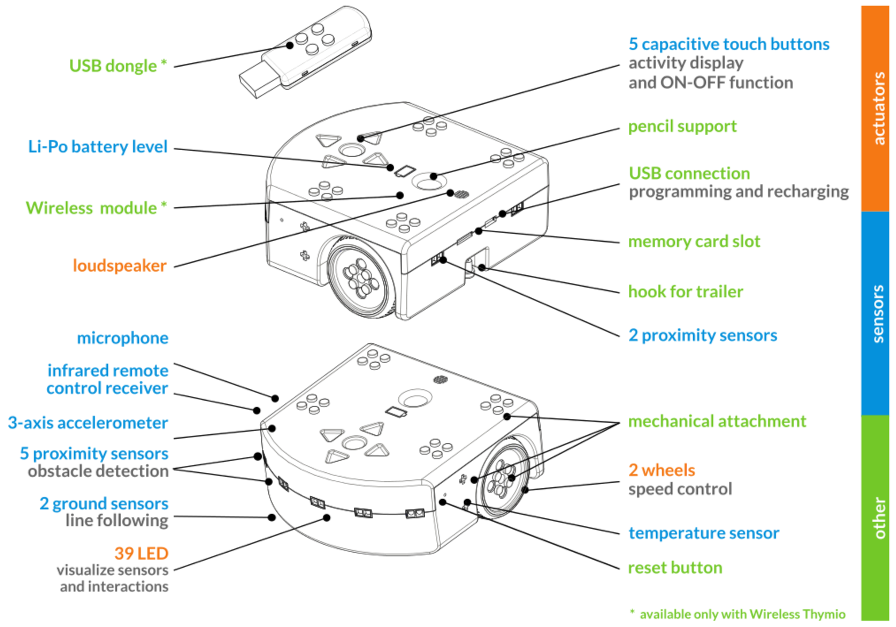


Fig. 2. Aseba Structure [5]

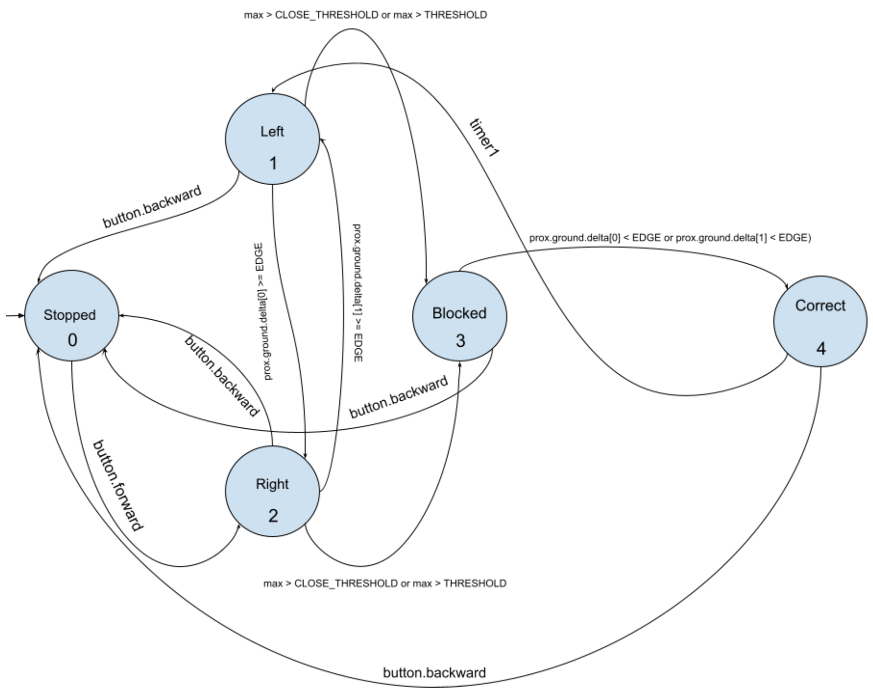
Important components of the Thymio II robot used in this project were the nine infrared proximity sensors, five on the front of the robot, two on the back of the robot, and two ground sensors. The robot also has five capacitive touch buttons; the forward button and backwards buttons were used for this project. Additionally, the robot has thirty-nine LEDs and two DC motors. The Thymio II robot's maximum speed is 14 cm/s [4]. Once code has been written for the Thymio II, it connects to the computer via a USB connection, and the code is loaded onto the robot.

Specific limitations for this project are that the Aseba language is very limited. ASEBA only allows integers and arrays of integers as variable types. The Thymio's limitations are that there are no infrared proximity sensors on the side of the robot and that the robot can only successfully sense in front of itself up to 8 cm. Sensors in the robot itself also have bias, standard deviation from actual values and variability and random deviation from actual values. We did not notice any significant bias for our specific robot, but to deal with variability, we would take a mean of proximity readings to ensure accuracy.

For our project, we used and improved two programs that we had previously seen. The first was *Lab 6 - Modeling the Real World* [5]; this lab used code to make the Thymio robot follow a line using its left ground sensor. This code would make the robot continuously turn left and right depending on whether it could see a black line on the ground. This action emulates the robot going straight following the line. We used this code to modify it to be able to follow lines using both its ground sensors instead of only one to improve accuracy. The second lab that we looked at code to help our project was *Lab 7 - Dealing with Failure* [6], where we needed to deal with the robot not being able to go around sharp turns on its own; it would go into a lost state until it had found the line again, this was used for our project because we wanted to make sure that our robot could also deal with sharp corners while it was going around duplo blocks, it would be able to return to the line. As we only sometimes had access to the Thymio II robots to test our code on, we were using Hammer, a simulator developed by Dr. Alexander Brodsky that allows us to test our code on the robot in a simulation of the course [7]. Some timing issues led to our group not being to test our code on the physical robot, so one of the most significant assumptions that we made for this project was that our robot would perform similarly enough as to how it did in the Hammer environment, and if we ran into minor issues, we would have time to fix them within the thirty seconds we were given between tests. Other assumptions we made for this midterm were that the track we would be using for the competition would be the same or similar to the track we were given to test our program.

Program description:

To start the movement of our robot, button.forward is pressed and in this event handler we start moving the robot to the right (motor.left.target = TARGET and motor.right.target = 0), the state is changed from STOPPED (state = 0) to RIGHT (state = 2). We wanted to improve upon the initial line follower code that we were given which only used the left ground sensor, so to switch between the right and the left states, which check if the associated ground sensor is no longer sensing the line and if so, turn back in the other direction. From RIGHT (state = 2) to LEFT (state = 1) happens when prox.ground.delta[1] >= EDGE and the same thing happens when changing from LEFT to RIGHT except we are looking at the values for prox.ground.delta[0] instead. Our robot changes from either LEFT or RIGHT into blocked if when the prox sensors are polled, if the max of sensors [0:4] and if max > THRESHOLD, then the Thymio state changes to BLOCKED (state = 3). If max > CLOSE_THRESHOLD then the state will also change to BLOCKED but the robot will also start to back up, this is while the robot is still trying to go around the block to avoid the duplo block but it doesn't turn on quite enough of an angle so we want to check and make sure that if it doesn't turn enough, when the it reaches an even closer threshold (CLOSE_THRESHOLD) that the robot will back up and then continue its maneuvre. When the robot is in its BLOCKED state, then it checks to see if the robot is able to see the line or not, if it is not able to see the line, it goes into CORRECT state and timer.period[0] is set to 100 as the robot will drive straight for that time then give itself more time for correction to get on the line, it sets timer.period[1] where it will reset to go back to the left state.
Below attached is our state transition diagram for our code (see appendix for the code).



max > CLOSE_THRESHOLD or max > THRESHOLD
timer1
Left
1
button.backward
prox.ground.delta[0] >= EDGE
prox.ground.delta[1] >= EDGE
prox.ground.delta[0] < EDGE or prox.ground.delta[1] < EDGE)
Stopped
0
Blocked
3
Correct
4
button.backward
button.backward
button.forward
Right
2
max > CLOSE_THRESHOLD or max > THRESHOLD
button.backward

Fig. 3 STD of our code

Our first idea for solving the hurdles problem was to improve the code we were given for line following with one sensor to make our robot follow the line with both its sensors. This way, we could guarantee a better chance of our robot not getting lost and a better chance of our robot being able to make it around steeper corners. Once this part of the problem was solved then, we approached the problem of the robot sensing the duplo block, starting its avoiding maneuvre of the duplo block, and making use of the timers to have a set time for the robot to turn around the duplo block and return to the line, in the case that the robot was not returning to the line we added a check case where if a certain amount of time had passed and the robot still had not seen the line it went into a “recovery” mode to get back to the line.

Problems we ran into happened when the robot was going around the block but would not turn on a wide enough angle, so we had to adjust our code to make sure that if the robot was sensing the block at an even closer threshold value (CLOSE_THRESHOLD), so that it would be able to fix its turn before it hit the block. Another problem that we ran into was with the robot being unable to go around the last two duplo blocks. The robot was able to complete the track if it went to the right of the duplo block to avoid it, but if the robot went to the left, it would get lost and could not return to the line. This was a slightly tricky problem to deal with initially because the robot would not perform the same way every time. On some tests, it would go to the right; on others, without changing the code, it would go to the left and get lost. To solve this problem, we changed the code with the timers for returning from the lost state, which seemed to account for the problem. The final problem we ran into was that the original code we were using had around 14 states, so to clean up the code and the state transition diagram, redundant states were taken out, making the code a lot easier to follow.

Results:

In the first course, our robot could complete the course, only hitting the second duplo block on the course slightly in around a minute and a bit. The second time, our robot got lost going backwards around the steep corner and was not able to go past that section to complete the course. Our robot completed the course the third time without hitting any of the blocks in one minute and 20 seconds. In the end, our program was functional going forward in the track but could be improved for going backwards; this could have been done by making a lost state to return to the line, similar to when it returns to the line after being blocked, but the robot would not be relying on being blocked by a block before it could return to the line. Compared to our peers, from what we remarked, there were a decent number of programs that did not make it to the end going backwards, so from that point of view, we were not in the minority not to have figured out slight improvements to make our code even better.

Conclusion and future work:

In conclusion we reached a good solution which managed to pass 2 / 3 hurdle course i.e. it managed to follow the tape line quickly and got around blocks without moving it off the boundary or losing the line, however in one of the tracks it was essential to add a recovery state so that the robot does what its suppose to do without losing track no matter how we change the hurdle course, Had our group had more time to work on our program for a hurdles course, something that we would've liked to have fixed would be that when the robot went around the course backwards instead of forwards, our robot was unable to return to the line after the first block once it goes around the steep corner, it was unable to return to the line as the sensors weren't able to sense the line but the timers and the routines within the timer were associated with the robot returning to the line, hence making the code more generalized which would work on different tracks, Also another thing that we would do is try and come up with best values of constants like THRESHOLD, SPEED, EDGE etc, resulting in the fastest and safest way for robot to reach the end of course. In future projects, we would implement the things we learned during this project and the concepts we understood more clearly from completing the task of the hurdles project.

References:

1. H. Sajjad. (2023). CSCI 1108 Experimental Robotics. Midterm Project Outline. [Online]. Available: https://dal.brightspace.com/d2l/le/content/248846/viewContent/3529354/View
2. Mobsya. "About." *Thymio by MOBSYA*, 04-Mar-2023. [Online]. Available: https://www.thymio.org/about/.
3. Wikidot. "What is Aseba?" . Available: http://aseba.wikidot.com/en:description.
4. Wikidot. "Specifications - Thymio and Aseba." *Wiki.Thymio.* [Online]. Available: http://wiki.thymio.org/en:thymiospecifications.
5. H. Sajjad. (2023). CSCI 1108 Experimental Robotics. Lab 6 Modeling the Real World. [Online]. Available: https://dal.brightspace.com/d2l/le/content/248846/viewContent/3437550/View
6. H. Sajjad. (2023). CSCI 1108 Experimental Robotics. Lab 7 Dealing with Failure. [Online]. Available: https://dal.brightspace.com/d2l/le/content/248846/viewContent/3437553/View
7. A. Brodsky. "User's Guide to Thor's Hammer A Thymio II Simulator and Visualizer." Dalhousie Computer Science Faculty, Halifax, Nova Scotia, Canada.[Online]. Available: https://web.cs.dal.ca/~abrodsky/other/thor_hammer.pdf

Appendix:
Our midterm code:

```
#*
Filename: Midterm
Title: Midterm Project
Course: CSCI1108-01
Date Created: 2023/02/14

Description: A program so a robot will follow along a line while avoiding objects in the way

Authors:
   Aaron Mai - B00924036
   Bella Arsenault - B00918242
   Evan Thompson - B00929818
   Harshil Varia- B00919242
   Tristan Seely - B00898894
*#
#robot states
const TARGET = 300
const STOPPED = 0
const LEFT = 1
const RIGHT = 2
const BLOCKED = 3
const CORRECT = 4
const EDGE = 400
const THRESHOLD = 2000
const CLOSE_THRESHOLD = 3000
var state = STOPPED
var max

onevent button.forward # start movement on button forward
        motor.left.target = TARGET
        motor.right.target = 0
        state = RIGHT

onevent button.backward # stop movement on button back
        motor.left.target = 0
        motor.right.target = 0
        state = STOPPED
        timer.period[0] = 0

onevent prox
        call math.stat(prox.horizontal[0:4], 0, max, 0)
```

```
	if state != STOPPED then

		#robot finds the line again
		if state == BLOCKED and (prox.ground.delta[0] < EDGE or prox.ground.delta[1] <
EDGE) then
			state = CORRECT
			motor.left.target= TARGET
			motor.right.target = TARGET
			timer.period[0] = 300
		end

	#if moving left and not sensing line on left sensor
		if state == LEFT and prox.ground.delta[0] >= EDGE then
			state = RIGHT
			motor.left.target=TARGET
			motor.right.target = 0
			timer.period[0] = 0
		end

	#if moving right and not sensing line on right sensor
		if state == RIGHT and prox.ground.delta[1] >= EDGE  then
			state = LEFT
			motor.left.target = 0
			motor.right.target = TARGET
			timer.period[0] = 0
		end

		#if too close the robot will back up
		if max > CLOSE_THRESHOLD then
			state = BLOCKED
			motor.left.target = -TARGET
			motor.right.target = 0
			timer.period[0] = 200
			timer.period[1] = 0
		elseif max > THRESHOLD then
			state = BLOCKED
			motor.left.target = 0
			motor.right.target = TARGET
			timer.period[0] = 200
		end

	end
```

```
onevent timer0

        #Extra correction for getting onto the line
        if state == CORRECT then
                motor.left.target = 0
                motor.right.target = TARGET
                timer.period[1] = 200
                timer.period[0] = 0
        #after not sensing a block, it will go straight for a bit
        else
                motor.left.target = TARGET
                motor.right.target = TARGET
                timer.period[1] = 1000
        end

onevent timer1
        timer.period[1] = 0
        timer.period[0] = 0

        #resets to left state
        if state == CORRECT then
                state = LEFT
        #if no block has been seen for a while
        else
                motor.left.target = TARGET
                motor.right.target = -TARGET * 2 / 3
        end
```