Milestone 2: Robot Motion Control and Line Tracking

Engineering Design (ME) I EEE3099S

Department of Electrical Engineering

14 September 2023

Joseph Stewart I STWJOS003, *Nüvit Ilkin Demirtaş* I DMRNUV001, *Harry Papanicolaou* I PPNHAR001

Introduction:

For milestone 2 we were tasked with creating algorithms and then simulating the solutions to certain problems using Simulink and MATLAB. Files were provided to us to model real-life components which has allowed us to accurately simulate the real-life environment. The different sensors will be used to demonstrate some potential functionality for the robot. Some functionality that will be demonstrated includes distance measurement, angular movement, line following, object detection as well as maze solving.

Distance and Angle Control Algorithm:

Model:

Distance:
The wheel encoders will keep track of the number of rotations each of the respective wheels have completed. For the specific task the robot must move 1 meter in a straight line. Both wheels will move an equal amount and then the average of both distances added together will be the distance traveled by the robot. The distance the robot has travelled will be used to check if the robot has moved 1 meter. If it has then the motors will stop.

Angle:
The wheel encoders keep track of the distances that both wheels have travelled and then convert each of the encoder ticks into distance travelled. Because the robot is only spinning counterclockwise the left encoder values need to be negative to account for this. The distances are then converted into radians and then need to be converted into degrees. The robot will continue spinning counterclockwise until the degrees that the robot has rotated is greater than or equal to a predetermined value. When this happens, the robot will stop rotating.

Operation:

Distance:
The ticks from each of the encoders are both fed into a respective gain block which converts the number of ticks into distance in meters using the formula $d = ticks \times \frac{2 \times \pi \times wheelR}{ticksPerRot}$. The two distances which have now been given by the two respective gain blocks are now added together in a summing block. The average of the distance is now taken by multiplying the addition of the two distances by 0.5, this is done using a gain block. The output is now fed into a state block, which takes in the variable distTraveled which is the distance traveled in meters by the robot. Inside the state block the variable distTraveled is constantly checked to see if it is greater or equal to 1 (distTraveled>=1) if it is greater than the velocity of the wheels is set to zero (v = 0) if it is less than 1 then the velocity of the wheels is set to 1 (v = 1).

Angle:
The right and left wheel encoders are both fed through gain blocks which convert the number of ticks to distance in meters using the following formula $d = ticks \times \frac{2 \times \pi \times wheelR}{ticksPerRot}$. Both outputs to the respective gain blocks are then fed through a summing block, however since the robot is always rotating counterclockwise the left encoder distance needs to be inverted hence the summing block will subtract the left encoder distance and add the right encoder distance. The output of the summing block is then fed into a gain block that

converts the distance travelled by both wheels into rotations in radians using the following formula $rot_{rad} = distance \times \frac{1}{axle\ length}$. The output of that gain block is fed into another gain block to convert the rotations in radians into degrees using the following formula $rot\_\deg = rot_{rad} \times \frac{180}{\pi \times R}$. This value is then fed into a state block as the variable angle_covered. Inside the state block, the value for angle_covered is constantly being checked if it is greater than or equal to the predetermined rotation value of ref_ang (angle_covered >= ref_ang). If it is greater than or equal to ref_ang then the robot will stop rotating (w = 0), however, if it is less than ref_ang then the robot will continue rotating (w = 1).

Line Following Algorithm:

Model:
The data from the line sensors is split up into individual channels for each sensor. The robot is set to a constant speed, and the data from the front left and right line sensors is constantly monitored. If one of the sensors is no longer over the line, then the robot will correct its heading such that it returns in the direction of the line.

Operation:
The line sensor data is broken into separate channels for each of the four line sensors (sensor back right and left (SBR, SBRL) as well as sensor front right and left (SFR, SFL)), this is done using a demux block. SFR and SFL are then fed into a state block as variables. The entry point to the state block sets the angular velocity to zero (w = 0), from there is monitors the values for SFR and SFL and waits for one of the sensors to go from a logic level of 0 to 1 which indicates that respective line sensor is no longer directly over the line. When this happens an opposing angular velocity is applied which corrects the course of the robot. The velocity is held constant at 0.075 as the only thing that is required from this task is to get the robot to remain on course.

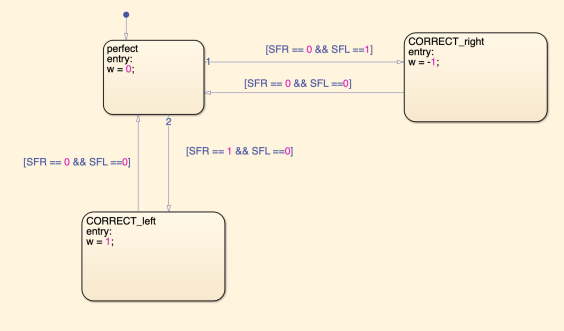


Figure 1: The State Chart Responsible For Keeping The Robot Over The Line

Object Detection Algorithm:

Model:
The output of the ultrasonic sensor is constantly monitored. The robot will move forward with a fixed velocity until the distance measured is less than a predetermined threshold, which his then when the motors stop, and the robot comes to rest. This is to ensure an accurate reading as all ultrasonic sensors have an operating distance that they are rated for.

Operation:
The data from the ultrasonic sensor is inputted into a state block as the variable ultrasonicValue as well as a predetermined distance called threshold. The default value for the robots forward velocity is 0.2 (v= 0.2). The distance measured from the ultrasonic sensor is constantly monitored to see if the distance is less than or equal to the threshold (ultrasonicValue <= threshold), at which point the velocity of the robot is set to zero (v = 0). The angular velocity is held at zero for the whole period of this simulation (w = 0).

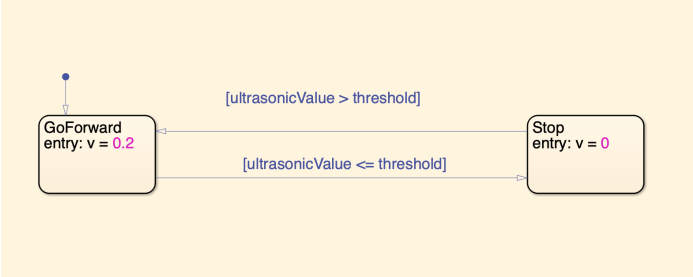


Figure 2: The State Chart That Monitors The Value of The Ultrasonic Sensor's Measurement

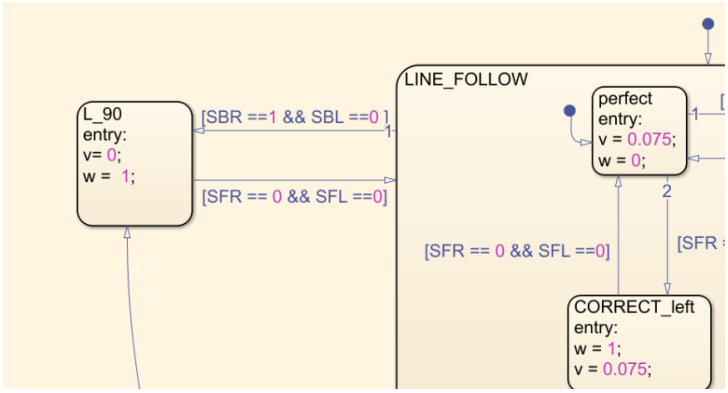
Localisation Algorithm:

Model:
With every object detection instance that the robot experiences, the distance of the given object is measured and recorded if it is the smallest distance for that treasure hunt instance. The robot will continuously navigate through the maze, navigating through each branch in search of viewing points, updating the shortest distance with each subsequent object detection. Once the robot has reached its original starting point, it will loop through the entire maze again until it reaches the closest object, the desired destination.

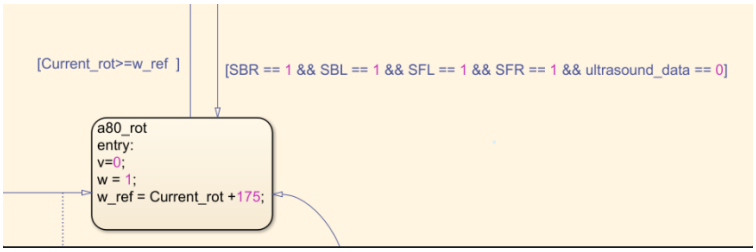
Operation:
The LINE_FOLLOW state is the original entry point for this algorithm. This state is identical to the line following algorithm in the heading above. This state is remained in until a left turn in the maze is detected.

The detection is done via a pair of secondary sensors on the back of the robot. They are placed at a width such that they are usually outside of the line detection width, so that they are usually returning values of 1 (white).

When the left rear sensor detects a protruding line to the left (SBR == 1 && SBL == 0), the robot enters the L_90 state, signifying a left turn is required. The robot will stop (v = 0) and begin to rotate counterclockwise (w = 1). The state will be maintained until the front left sensor and the front right sensor detect the branching route (SFR == 0 && SFL == 0) and the robot will return to LINE_FOLLOW mode until another exit case occurs.



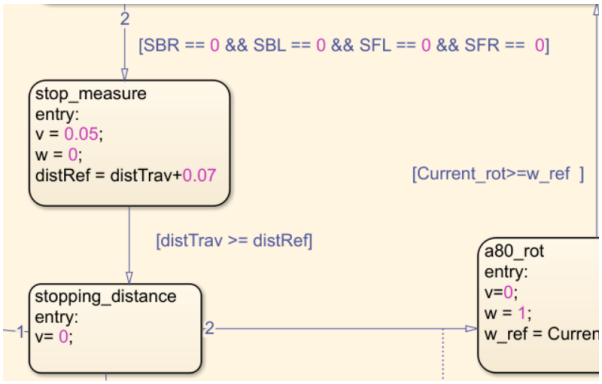
In the event that the robot meets a dead end, ie: The path just stops, it will be detected if all of the sensors read white (0) and the ultrasound sensor doesn't detect anything (SBR == 1 && SBL == 1 && SFL == 1 && SFR == 1 && ultrasound_data == 0). The robot will enter the a80_rot state. The robot will stop (v = 0) and begin to rotate counterclockwise (w = 1). A reference angular position, Current_rot, is calculated using the wheel's rotation from the encoder's values, and is incremented by 175 to give the value w_ref. As the robot rotates, Current_rot is incremented, and once the desired reference angle is achieved, (Current_rot >= w_ref) the robot returns to the LINE_FOLLOW state.



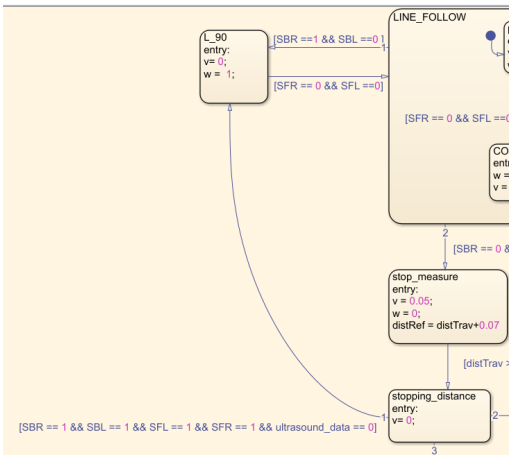
Note the state entered/exited to above this image is the LINE_FOLLOW state.

The final exit case from LINE_FOLLOW is the event that the robot detects a viewing point (SBR == 0 && SBL == 0 && SFL == 0 && SFR == 0). In this case, a viewing point causes all of the sensors to detect black simultaneously. The robot's speed is reduced by 33% (v = 0.05) and it is set to be not rotating (w = 0).

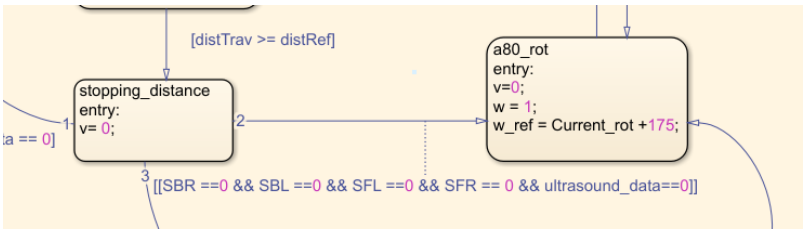
A measure of how far the robot has moved is also recorded, in the same vein as the Current_rot metric. The variable of how far the robot has moved is distTrav. After the speed and rotation adjustments, the robot will move forward until it has incremented distTrav by 0.07, at which point it will enter the stopping_distance mode.



At this point the robot is made to halt all movement (v = 0). From here there are three possible events that the robot is able to handle. Firstly, in the event the robot reaches a pathless location without any obstacles being detected (SBR == 1 && SBL == 1 && SFL == 1 && SFR == 1 && ultrasound_data == 0) it will enter the turn left mode, after which it will find the path from where it came and LINE_FOLLOW it back to where it came from.

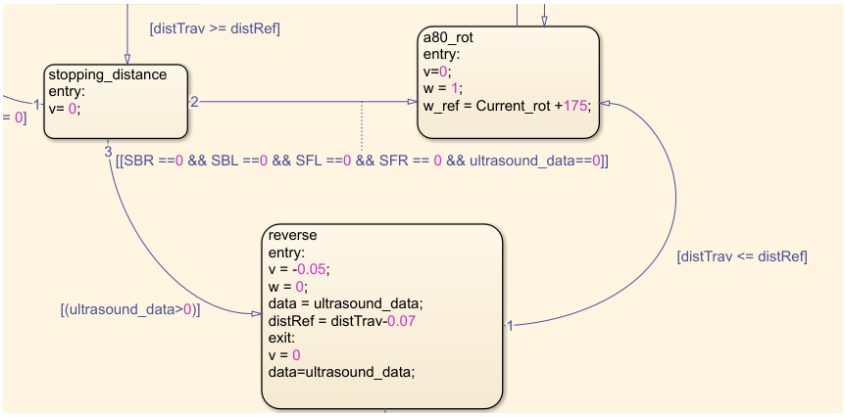


In the event the robot reaches a position while still on a detection point, ie: Line sensors still detect all black, and there are no obstacles being detected ([SBR ==0 && SBL ==0 && SFL ==0 && SFR == 0 && ultrasound_data==0]) the robot will enter the 180 degrees turning mode and path back the way it came. This trigger indicates the “end point” of the maze has been reached, hence the need for a 180 degree turn, to now go back to the start to end the first run through of the maze.

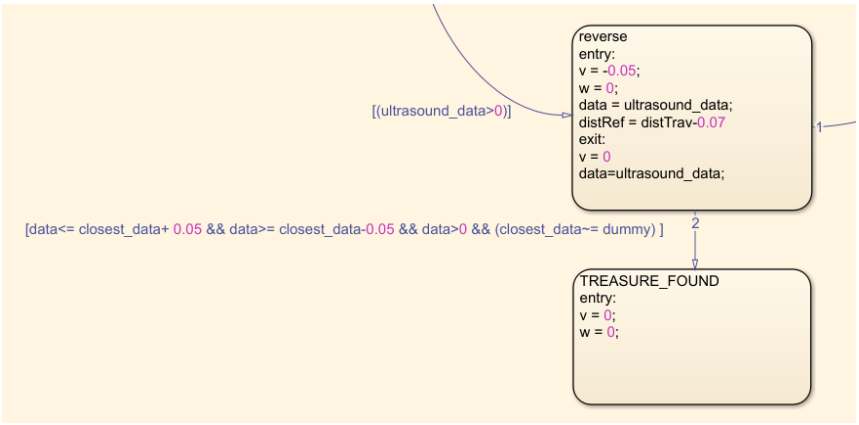


The distinction between the above “turn around when we don’t find an object” cases is that there is a case where the robot may overshoot the viewing point and it needs to be dealt with, and there was a time during testing where the robot will return to the viewing point and try to search again and fail, and so it will constantly reset itself in a loop and get stuck, so the 180 degree turn is necessary.

Finally, if the robot detects an object (ultrasound_data > 0) it will record the distance the object is away from its detection point, reverse and turn 180 degrees to leave the observation point.



When the ultrasound data is recorded, it is compared to the previous smallest recording of the same kind. In the event of it being smaller, the data is recorded as closest_data. In the event that it is the same or incredibly similar to (given inaccuracies with the measurement and reference point) the previous smallest recording, ie: In the event the robot is looking at the same object after going through the entire maze again without finding a smaller one, the robot will enter TREASURE_FOUND mode, as this can only occur if the robot is looking at the same object that was previously recorded to be the closest.



This is done through a series of simple comparison blocks that simply compare the current data, “dummy”, with the previous closest measurement, “closest_data”.