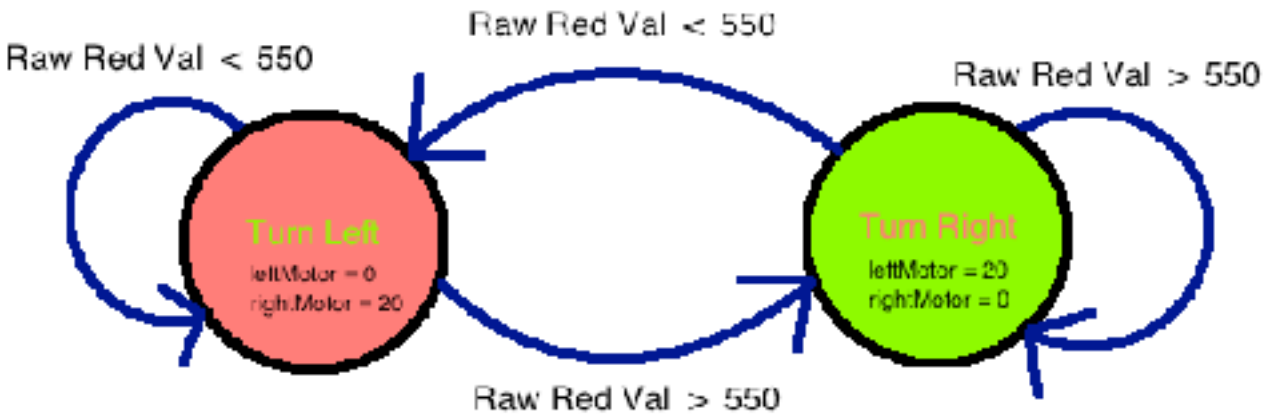
A Subsumption Architecture Implementation on LEGO Mindstorms EV3 Robots

Introduction

We were tasked with the implementation of several autonomous behaviours for the LEGO Mindstorm EV3 Robot. This report outlines the approaches taken and the challenges faced while trying complete each of the four behaviours. The first of the four was “follow”, to use the robot’s light sensor to follow a projected black line as smoothly as possible. This was successfully implemented my using a raw light value threshold to determine whether the robot was under white light or black light from a line. Next we implemented a simple “forage” behaviour, when a robot is placed outside of the projected line, under white light, it should be able to search around its environment for a black line. Our robot radiates in a circular motion from its starting point until it detects a line using the light sensor. The third feature implemented was “avoid”, the robot must use its ultrasonic sensors to detect an obstacle obscuring its line, move around, missing, the object and commence following the line. Our robot does this in three stages, moving away from the line until it is clear to turn past the object, move past the object until it is clear to turn back towards the line, move forward towards until the line is re-found. The robot knows when to move onto the next stage by turning to use its ultrasonic sensors at regular intervals. Unfortunately we were unable to fully complete the final behaviour “observe”. This required the robot to calculate and stop on the middle of the longest stretch of a given track. We recognised that the challenge here was accurately recognising when the robot turns a corner and then recording the length of each straight. Our solution used consecutive Gyro sensor reading outside the recent average to calculate a corner and counted the number of times the robot hit the edge of the line between corners to quantify the length of each straight. Due to the time taken to implement the other three behaviours we didn't have enough time to fully execute and test this feature.

Behaviour 1 - follow



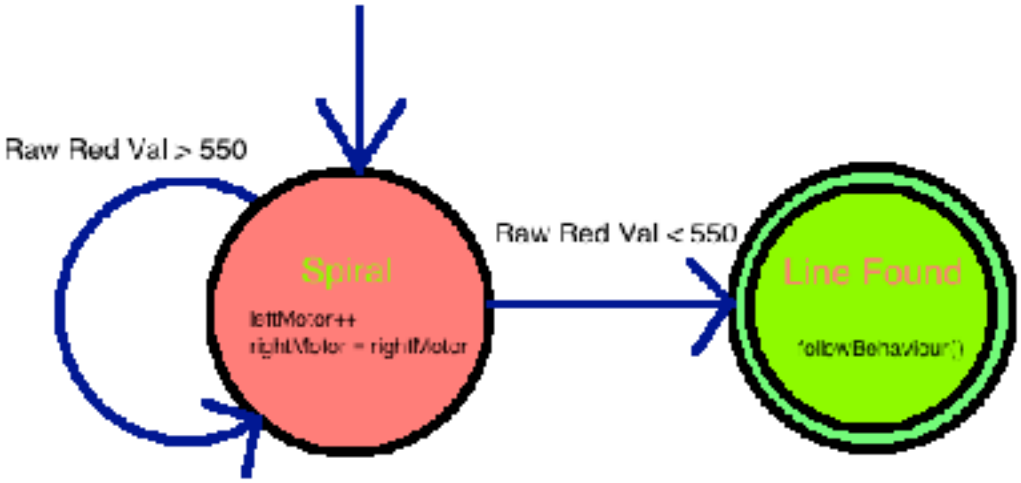
We achieved this by recognizing a difference in the detected light values when the sensor was placed in the white light compared to when placed in the black light of the line. We calculated a threshold for the raw light values whereby we could be confident that if a recorded raw red light value was beneath 550, the sensor was under the black light, above this value, we deemed to be off the line, under white light. We used only the red RGB

value for simplicity, we tested a threshold with all three values but found it difficult to set correctly, we concluded a single value is still capable of detecting a line.

To get the robot to follow a detected line we adjust the robot's trajectory over time if it drifts out of the line and white light is detected. Our issue was that it is difficult to determine whether the robot is drifting to the left or right of the line and therefore which direction to adjust the robots direction. For our initial solution we programmed the robot to turn left, until it hits the left edge, then turn right until it hit the right edge, essentially zigzagging along the line. While this worked to some extent, one of the requirements for this behavior was for the robots motion to be as smooth as possible. Our final implementation simplified our initial attempt to only hit the left edge of the line. Rather than traverse across the line to hit the opposite edge, we turn left if in black, and turn right if in white light. This has the effect of and relies upon following the left edge of the line. If at any point the robot hits the right edge of the line, the robot would move away from the line, as it is programmed to turn right if white light is detected. To ensure the robot never hits the right edge, we had to ensure that our left turn whilst in black was sharp enough to miss the opposite edge, whilst still maintaining as smoother forward momentum as possible. After many tests and tweaks, we settled on a right motor speed of 20 and a left motor speed of 0 to achieve the correct turning circle. These values could be tweaked depending on the thickness of the line the robot is expected to follow. Thinner lines would require a tighter turn whilst a thick line would allow for a lighter turn and smoother forward motion. The robot can go clockwise or anti-clockwise, but will always hug the edge of the line to its left.

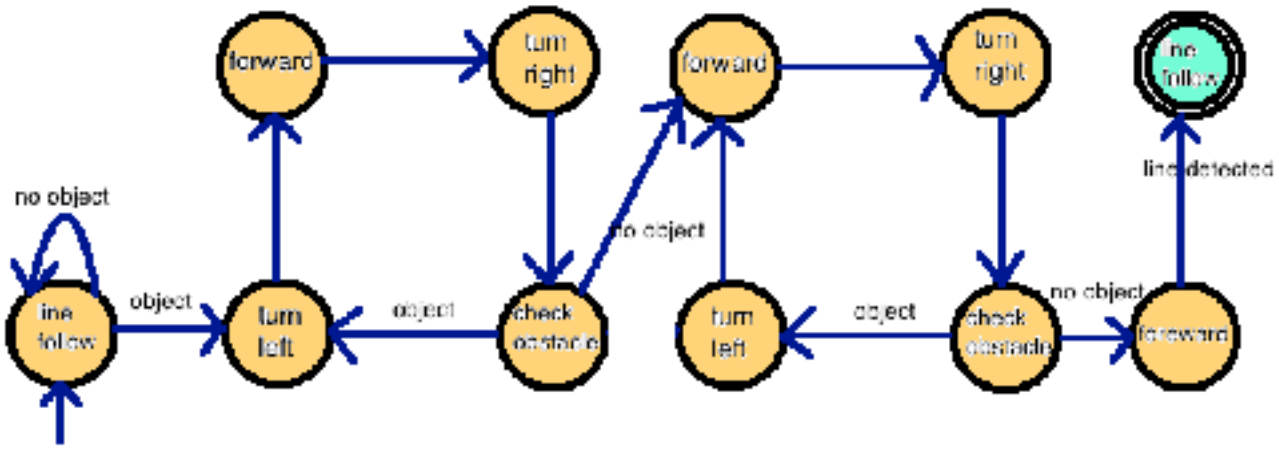
Having experimented with getting the robot to detect moving from white to black we noticed that sometimes the projected line didn't have a sharp enough edge, occasionally the robot would detect multiple lines due to the blur on the edge of the line. To compound this issue, just one sensor reading beneath the light threshold would signify a line, this caused false lines to be detected if there was an erroneous reading or if something momentarily obscured the sensor. To fix this, instead of using just one sensor reading we keep an average of the previous five sensor readings, this reduces the impact of an incorrect light value. This also allows the robot to handle blurred line edges as a line is only detected when the recent average of light values falls beneath the threshold, consecutive readings above and below the threshold, caused by the blurred edge, no longer signifies multiple lines.

Behaviour 2 - forage



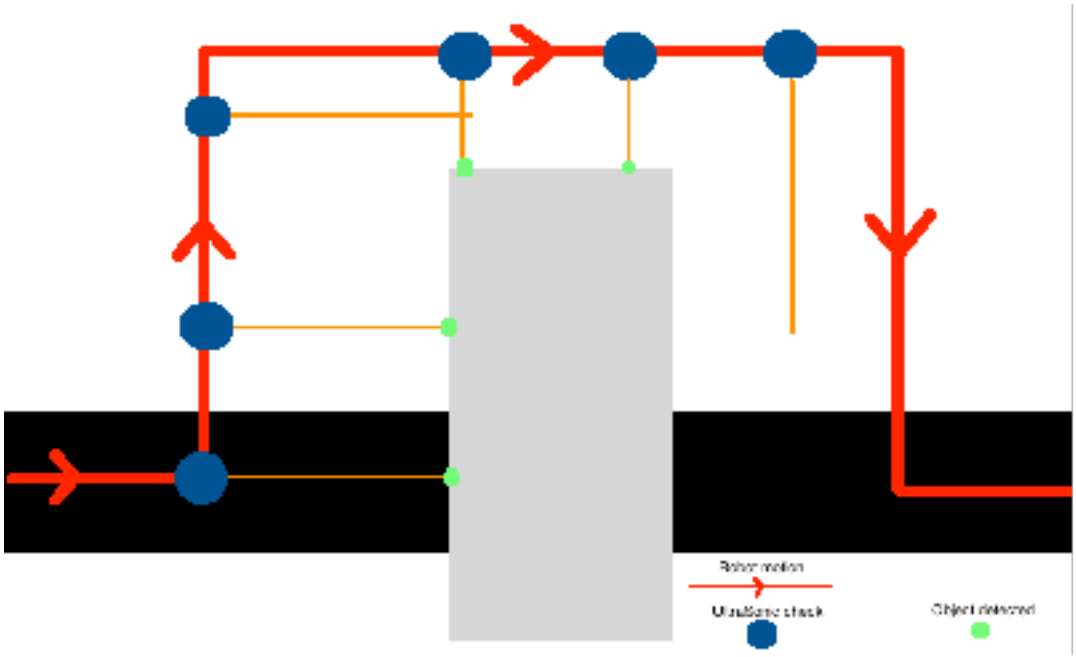
We realized that the only possible way the EV3 could detect a line, given its current set of sensors, was through its color sensor, so there would be no intelligent way to search for a line it hasn't already detected. As far as the robot knows all directions are equally likely to have a line in. So we decided, rather than a completely random search, to cover as much ground as possible from the robots starting position. We have programmed the robot to spirals out from its start position until it finds a line. We achieve the spiral by having a set speed of the right motor, then slowly increase the left motor speed and therefore the robots turning circle. If the left motor speed ever gets equal to the right motor speed we assume that it cannot find a line within its environment so we stop the robot. At all points during the search, the robot is monitoring the light sensor average. If at any point the average falls below the threshold we end the search task, stop robot and hand control over to the follow task.

Behaviour 3 - avoid



Our avoid functionality uses the robots ultrasonic sensors to detect a physical object in front of the robot. The avoidance task is a constant loop that calls a function to check the ultrasonic sensors for an object, if at any point an object is detected then we set the robot to avoidance mode and call the function that handles moving around the object.

We have programmed our robot to move around an object in three stages, one for each side of the object it needs to avoid. We start by moving turning left and moving at right angles to the line, at set intervals, the robot turns 90 degrees to the right to check if its clear to move past the object by rechecking the ultrasonic sensors. If it is not clear then- then we turn left and continue further from the line. If it is clear then we still turn left and move further away from the line, this is to ensure that the robots width doest hit the object as it goes past. Now the robot turns right and starts to move parallel to the line. This exact process is used again to calculate when the robot is clear of the object and can move back towards the line. The final step moves the robot forward towards the line. Using the color sensor threshold to detect the line in the same way as with the “forage” behavior. Once the line was detected the robot makes a sharp turn 90 degrees to the left to ensure when passing over to the line following task that the robot continues in the original direction.



Behaviour 4 - observe

The “observe” functionality was not completed. We recognized that the key aspects of this task were to recognize and detect when the robot turns a corner on the track and be able to measure the length of each straight. Our problem was, because of the way that we had implemented our line following, it was hard to work out the direction that the robot was heading in as its turning left and right along the straight line. Our implementation took the gyro sensor reading when the robot hits the edge of the line. We realized that on a straight, the robot hits the edge of the line at the same, or very similar, angle. Even though this gyro sensor reading would not indicate the direction of the line, it could be used to identify corners and straights. Similar to our light average function we store a list of recent gyro sensor readings, or the last four times the robot hit the edge. A threshold is then created based on the average gyro sensor reading. Two consecutive readings outside this threshold would signify a corner.

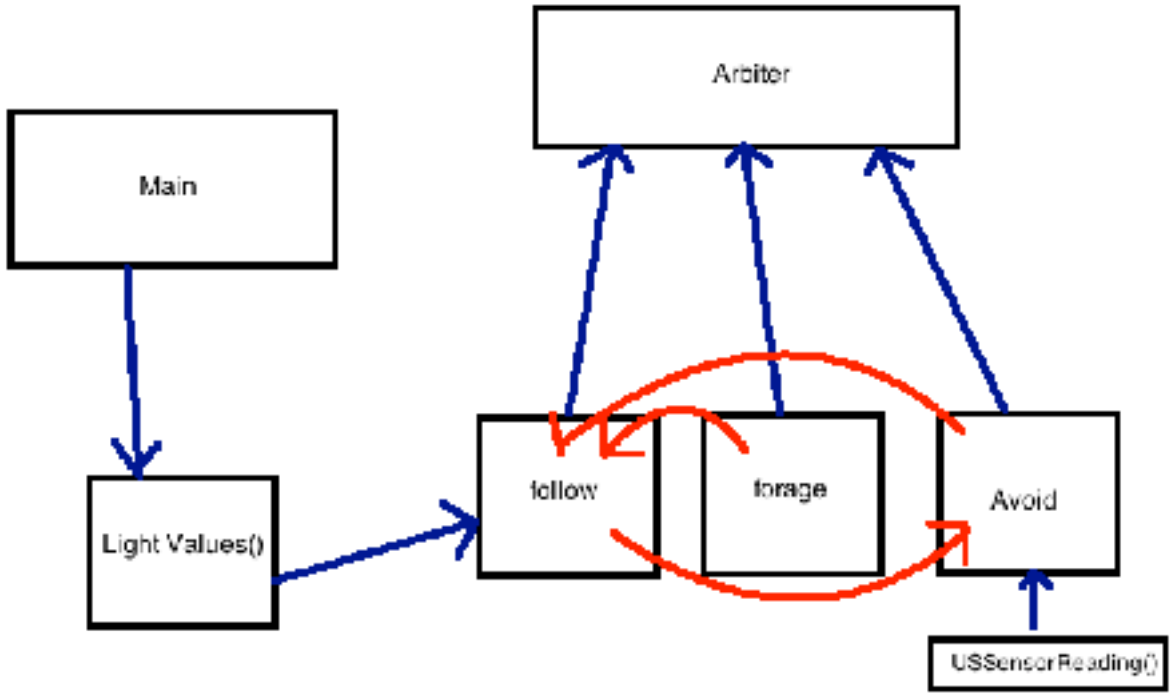
We planned to measure the lengths of each straight by counting the number of times the robot hits the side of the line between corners. At each corner, the length of the straight and the average gyro reading for that straight would be recorded in an array. This essentially maps and creates a representation of the track.

After a new side has been recorded in the array a pattern matching function will be called. This will, if the array has an even number of elements, split the array in two and compare if they are equal. Rather than exactly equal, thresholds would have to be used as the was some variations in both our length measurements and average gyro readings. If the two arrays match, this signifies two laps have been completed, each array represents a lap.

From here, we planned to figure out the longest side based on the data in the array. The position in the array is how many turns to get to the longest side. To stop in the middle we would just have our recorded length.

Behaviour interaction

Our main task calls four subtasks, one for each of the three behaviors implemented and the arbiter. The arbiter is responsible for setting the motor speeds of the robot, using different variables depending on three condition statements signifying which mode the robot is in, lineSearch, lineFollow, and obstAvoid. To switch between behavior modes we can simply switch the boolean variable for that particular mode and the arbiter will set the motor speeds for the specified mode. By default lineSearch is set to true as that is the most likely start state, once a line is found it is set to false and lineFollow is set to true. Likewise, if an obstacle is detected obstAvoid is set to true and the others to false. In the main, our function to get the average light values is constantly called from a loop, this means out average light variables are always up to date and the function doesn't need to be called each time they are needed. As the tasks are always running, the arbiter is essentially in control of which modes changes actually take effect, this allowed for easier development as we didn't have to worry about each task interfering with each other.



Personal reflections

The workload of this project was split equally between the two of us. The vast majority of tasks were completed together as a team, however, we tried to play to each other's strengths as much as possible. I pride myself on being good at solving a given problem using mathematics and logic but I am not as quick at transferring this into code as my team member, particularly when using a C-based language for the first time in a few years. I provided the ideas and logic to our solutions by means of diagrams, explanations, and pseudocode, then much of the typing up and Syntax checking was done by my team member to make sure the code compiled and functioned as expected. Any crashes or unexpected bugs were left to my team mate to fix and up until he dropped our robot on the penultimate week, he was in charge of physically testing the robot whilst we both observed

it in action. All of our work was ultimately completed and checked together but these were roles we naturally fell into as the weeks progressed.

Conclusion

Overall I am quite disappointed with how this task concluded, I feel that the limiting factor was the amount of time it was possible to access and test with the robots rather than what we were capable of implementing. Despite this, I believe that three of the four behaviors were implemented and tested well, despite a few limitations. While the “observe” behavior was not completed in time for the demonstration I believe that our logic was correct and would have worked given some time to test of the robots.

We are aware of an issue with our line following behavior by which very sharp corners or corners with a very specific apex cause the robot to loose track of the left edge of the line. When turning left, the sensor misses the apex of the corner and hits the right edge of the line, detecting white light, causing the robot to turn right, away from the line. To fix this would need to detect when this happens and either moves back to the left edge or switch our behavior to turn in the opposite directions in order to stay on the line. This can also occur if the projected line is too thin, upon finding the line, the angle of the line is often too sharp to turn without hitting the opposite edge of the line, causing the same issue. On rare occasions, one particular projector would cause the robot to behave differently to another. We believe this to be due variations in color and light intensity. We did some research into dynamically setting our light threshold based on the light levels it was receiving, so the robot would improve reliability the longer it spent in one environment, or area of environment, as it fine-tuned its light threshold to distinguish between the two colors more accurately.

There was a number of extra features that we were planning on adding to our behaviors. One of the biggest was to give the robot more abilities to use the ultrasonic sensors to monitor its environment. Currently, the ultrasonic sensors are used only in select scenarios, when following the line to detect obstacles and to ensure the robot moves around a detected obstacle. We had intended to use the sensors to ensure that no obstacles were hit when finding a line during the “forage" stage and to be more aware of other potential objects when moving around a detected object on the line, currently, it focuses solely on the object it is avoiding.

If we were to spend significantly more time with the robots I personally would like to research alternative methods to achieve simple line following. Our current method of constantly turning left or right means that any given moment it is difficult to determine exactly what direction the robot is heading in relation to the line. This made other behaviors even more challenging, in particular, “avoid” ,when leaving the line, and “observe”. If the robots motion was smoother and could somehow calculate the trajectory of the line before embarking on it, it would be easier to recognize corners and calculate the length of a line. One suggested solution I found whilst researching would be to use a second color sensor, to straddle both line edges and minimize sideways movement.