A Subsumption Architecture Implementation on LEGO Mindstorms EV3 Robots

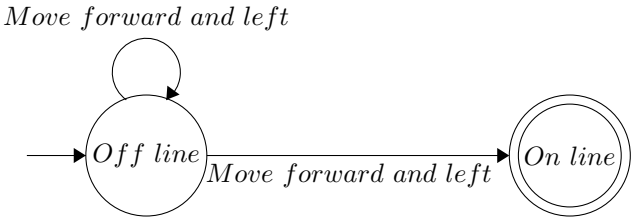
Team B5
Alexander Steenson - psyajs - 4252754
psyajs@nottingham.ac.uk

1 Introduction

Our project implements all the required behaviors within a subsumption architecture design. The program starts at the lowest level in the architecture, the find line task, which wonders about until it finds a line. The robot is able to detect when it has found a line by using a light sensor to detect a significant decrease in light, once the decrease in light has been detected the robot knows it must be a line. Once the robot has found a line the next task in the architecture, follow line, supersedes and stops the find line behaviour and begins following the line. The robot follows the line by continuously zigzagging, this causes the light sensor to go in and out of the line, allowing the robot to keep track of where the line is and enabling the robot to follow it. The robot continues to follow the line until either the avoid obstacle behaviour or the observe behaviour takes over. The avoid behaviour avoids an object it detects in front of it using the ultra-sonic sensor, by backing up and turning round the object. It then then continues to follow the line. The observe behaviour is the highest task in the architecture and stops all tasks below it once it is completed. The observe task measures when the robot has turned and the distance travelled between turns to be able to locate the midpoint of the longest straight and stop there.

2 Foraging (Wonder to find line)

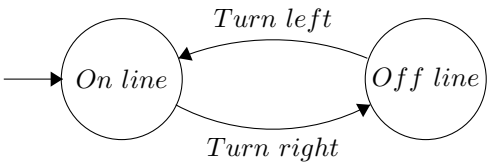
The robot first starts in the lowest state in the subsumption model. To find a line the robot begins by turning in a circle. Whilst turning, the robot is monitoring the light levels to detect if the threshold of a line has been met. The threshold is taken semi-automatically, meaning the threshold can change every time the program is run but the user must physically move the robot in order to take a reading. The user, just after starting the program, must place the robot on the black line closest to the center of the projector - this will get the brightest light value of the black line meaning the robot will never mistake a lighter section of the line as being off the line, the robot then will always be able to follow the line. Once the light value has been taken the user then places the robot where they want it to start. We only took readings and monitored the green light levels as this gave us the biggest value variance between being in the light and in the dark. For example, in the light the green light level might be 2000 and in the dark it might be 300. Whilst the blue light level in the light gives a reading of around 1000 and in the dark 200. We use this extra range to be more accurate in determining if the robot is on the line or not. After the robot has been turning and crosses into the line, the threshold will have been detected and the line has been found. The robot can then enter the next stage of the subsumption model - line follow. We didn't use any specific algorithm to be able to find a line, we just set the motor speeds in a while loop in the arbiter, and monitored light levels.



3 Follow (Follow line)

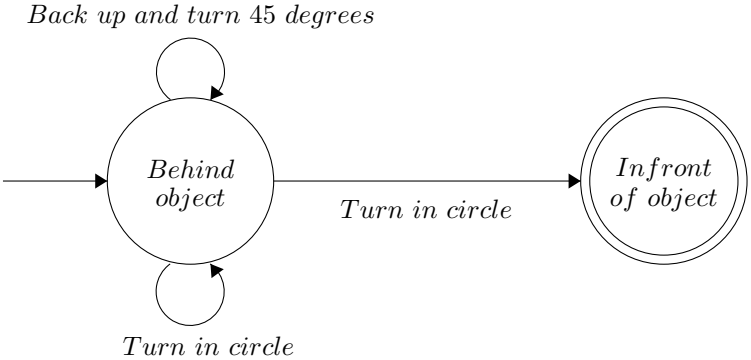
Once the foraging has detected a line the follow line state then takes over. We experimented with a number of different methods to achieve this and decided to use the zigzagging technique. Zigzagging provided a robust method to follow any line given and navigate round tight bends. Another method we tried we tried was, whilst the robot is on the line move forwards giving equal power to both motors. Once the robot is off the line, it moves left on the spot and checks if there is a line there, if not, it checks right. If after checking left and right there is still no line turn, increase the amount the robot turns and check again. Once a line has been found, go forward again. Whilst this would work the zigzagging is able to continuously move around the track without stopping and is therefore, more robust.

Our zigzagging algorithm works by always being in one of two states. The first state being when the light sensor is detecting it is on the line it moves forward but with a reasonable amount of more power on the left motor than the right, making it turn forward and off the line. Once it has recognized it is off the line it goes into the second state, this state makes the robot go straight again but with more power on the right motor than the left, moving forward and back on to the line.
This method makes the robot follow the outside line when going anti-clockwise around the track and the inside line when going clockwise.
We put a lot more power on the left motor when on the line because when the robot is going clockwise round the track it can go right round a sharp corner leaving the light sensor on the black. This is important because if the sensor goes off when turning around the corner it will go off on the outside line. The sensor will then sense it is off the line and turn left to try get back on the line but instead will just move further away.



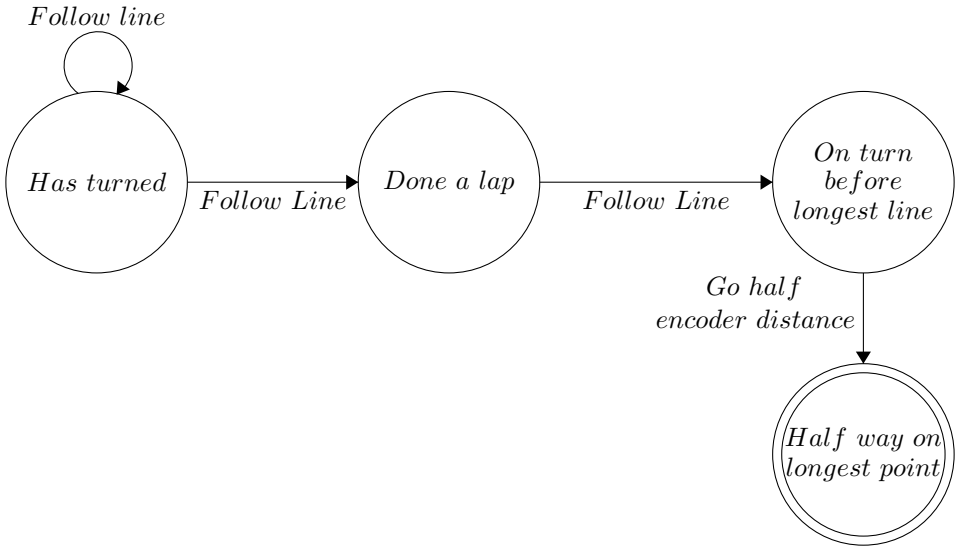
4 Avoid (Avoid obstacles)

Whilst the robot is following the line or looking for a line it is constantly taking ultrasonic (US) readings. Once the US readings detects an obstacle within 5cm in front of the robot the avoid behaviour stops the line following via the arbiter. The behaviour sets a Boolean value, object, to be true. The arbiter can then stop the line following and stop the motors to allow the avoid behaviour to avoid the obstacle. To avoid the obstacle, we make the robot move backwards, turn 45 degrees and move in a circle around the obstacle. The circle radius should be wide enough to allow the robot to move around the obstacle, but if the robot detects another or the same obstacle it will repeat the process ensuring it can always get around.
The robot turns around the obstacle on a timer. We used a timer to ensure that the robot can get all the way around an obstacle. Every time the US sensors detects an obstacle the timer resets, meaning when it is going round an obstacle and it senses another it won't continue for the length of time needed to only get round the first obstacle.
Once the timer has ended the robot should be either on the line or somewhere very close to it. If the robot finishes on the line it will go back into the line following, if the robot didn't finish on the line it will go to the line finding state and look for the line.
We couldn't find any specific algorithms to be able to complete this task but we thought this way was reliable and robust. We tried a couple of other methods such as moving in a square round the obstacle but this method proved to be unreliable. Another method we tried was to move the robot backwards, turn right, move forward and left whilst it didn't detect the object and forward and right when it did. This was unreliable because the sensors didn't work consistently when at an angle to the obstacle.



5 Observe (Detect the mid-point)

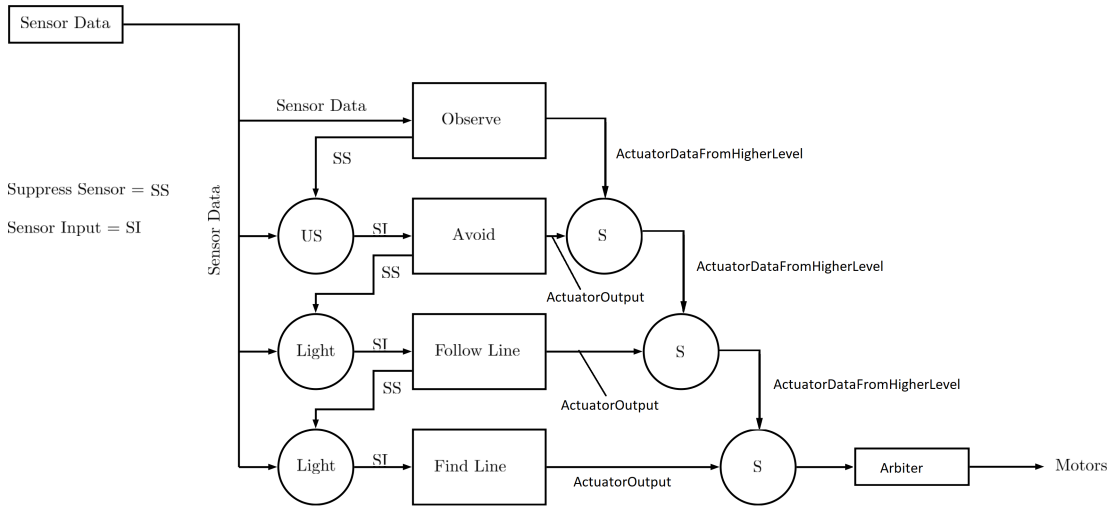
After the robot has wondered and found a line, the observer starts monitoring when the robot goes round a bend and how long the robot has travelled before the bend. The task takes a gyro reading once it is on the line and that is the starting gyro position. It then uses the gyro to recognise when the robot has gone round a bend, a turn of 45 degrees' registers as a bend. It also uses the motor encoders to find out how far the robot has travelled between bends. The total motor encoder value is the left motor encoder value plus the right motor encoder value. If the distance travelled between two bends is further than the previous best, then update the best encoder distance variable and update what corner number is just before the straight line. The robot knows it has done a full lap when the robots gyro reads the starting gyro value plus or minus 360 degrees, depending on if the robot went clockwise or anti-clockwise around the track, plus or minus 45 degrees, and then goes back to the first turn. From there it knows the turn that is before the longest straight and how long the longest straight is. The turn count is then reset and the robot continues forward registering the turns so it can find the right turn number. The robot is then at the start of the longest straight so it travels forward for half the total encoder count and then stops, stopping all other tasks. This algorithm isn't based on any localization or mapping algorithm, my partner and I discussed a couple of ideas that would work the best and settled on this one.
One of the issues we had was when the robot found a line and the robot turned to become parallel with the line, the behavior would count this as a turn on the track when there isn't one. To overcome this, we reset all the variables and use a timer for a second to not count any turns. This gives the robot time to turn parallel with the line and not count it as a turn.



6 Interaction with the arbiter

Each behaviour has its own task which run in accordance to which behaviour is currently being used. For example, whilst the robot is following a line the find line task is not being used but the line follow, avoidance and observe tasks are being used - as the robot needs to follow the line, look out for any obstacle, and be able to stop half way on the longest straight.
Each task only controls the state for that behaviour and the arbiter controls the movement and sets the motor speed.
The arbiter checks what the current state the robot is in and sets the motor speeds accordingly. For example, when the robot is following the line the line follower task updates the state to say if it is on a black line or not. The arbiter can then turn the robot left or right accordingly.
Another example is if the US sensors on the robot detect an obstacle the avoid task updates the current state to say there is an obstacle in the way. The arbiter then stops the line following and moves out of the way of the obstacle, once it is cleared the avoid task updates again to say it is no longer blocked and

the line following can take precedence again.



7 Personal reflections

The workload between my partner and I was split very evenly, we worked at the same time on the same tasks throughout the project. This allowed us to get tasks done efficiently with minimum errors. We would both talk over potential solutions and discuss what would work best before we programmed it. When programming, one of us would write the code and the other would watch in case any mistakes were made.

Both my partner and I put a lot of work into ensuring all behaviors were implemented robustly. I researched different ways we could implement each behavior before implementation. When testing the robot, if it didn't work I made sure we ran it a few times to look as to why it didn't work. We preferred this method instead of just acknowledging that it didn't work and trying figure out why just by looking at the code.

For the most part we programmed together in the lab sessions except for when we were implementing the observe behaviour. When implementing this we discussed methods of doing it and then both programmed it separately at home. The next day we compared code to see which one was better and use that one.

I was in a group with Ben Trusswell, Ben was very good at noticing little details and debugging code I had written. He has a good eye for spotting concurrency issues within the tasks and logic errors that stops the task from performing properly.

8 Conclusion

We were able to implement each behaviour, test it, and ensure they all worked robustly. We faced many limitations throughout the project - such as the line being projected rather than black tape on the floor and only using one light sensor.

The problem we faced with the line being projected is that there was no accurate light threshold we could get and when the sunlight got on the track the robot then sometimes got lost. Another issue with using the projector is that if the robot searched for a line outside of the track it might find a shadow and think it was a line. If we had used tape this wouldn't have been an issue.

The issue with only one light sensor is that we can only know if the robot is on the line or not, we don't know which side of the line it's on. This means that the only robust line following algorithm we could use was zigzagging which isn't very smooth. If we had two sensors we could follow the line with one sensor on the line and one off making it much smoother. By making the robot smoother we could more accurately track when the robot turns for the observe behaviour, because we are currently having to

zigzag we can only count a turn when the robot turns more than 45 degrees, meaning if there is a slight bend it may not register it.
Some improvements that we could make is making the line following behaviour use PID control to make it more accurate. We could also use automatic adaptive light threshold algorithm instead of semi-automatic so you don't have to pick up the robot to get a threshold.
We also found time a big limitation, if we had more time we could have implemented a more accurate algorithm for the observe behaviour, such as dead reckoning.
I am very happy that we were able to implement each behaviour and they all work robustly. The hardest part was the observe behaviour and being able to track accurately and consistently the number of turns the robot made whilst going around the track. Once we had managed to do this we could easily make it stop in the centre of the line by using the motor encoders.