Electronic engineering project- formula allcode buggy mobile application

Introduction

The aim of my project is to control the flowcode buggy using its API function from a programme which I will create in flowcode. I want the buggy to move forward in steps until it reaches an obstacle. Once the obstacle is reached it will detect it using the IR sensor at the front of the buggy. It will then turn 90° to the right and keep moving forward until the sensor on the left detects there is no longer an object. Once there is no longer an obstacle it will turn 90° to the left back to its original orientation. This process will be repeated for a set number of iterations. This will result in the buggy avoiding obstacles before continuing its path.

The LCD screen will display the iteration the programme is on. The LEDs will all be turned on at the start to indicate the programme is running. Once the programme finishes, the LEDs are turned off and a sound plays at a frequency of 65535 for 100ms to indicate the simulation is complete.

The software I will use is called flowcode. Flowcode is a programming software which can be used to programme external devices such as the flowcode buggy. Instead of writing code like a traditional programming software, it makes use of dragging and dropping functions (and in this case simulation macros which control the buggy) and also loops and decisions as well as various other features. Variables can be declared, and information read from sensors can be stored in those variables.

Equipment

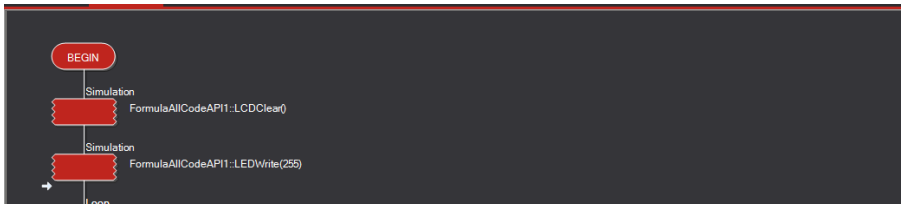
- Formula allcode buggy
- Computer
- Flowcode (V7.3.0.5) software

Procedure

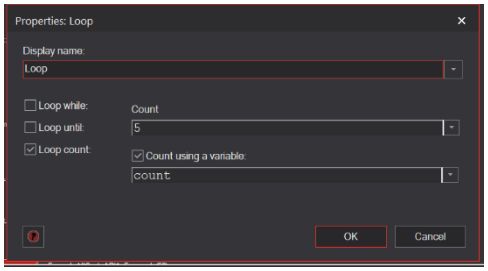
After opening flowcode and starting a new project, I clicked on the mechatronics tab and selected formula allcode API. I then clicked on properties and changed the COM port to COM4 because that is the port used by the buggy. I found the COM port number from the Bluetooth menu on my computer.

The first step was to decide which variables I needed to use and the data type for those variables. The variables I used were IR0 and IR2 which will be used to store the output from the front and left sensors. I also used the variable count which stores the amount of iterations through the programme. The variable type for IR0 and IR2 is byte, this stores a number from 0 to 255. I declared count as an integer, which stores whole numbers from -32768 to 32767. I set all these variables to be initially equal to zero.

To start the simulation I added the simulation macros to clear the screen and write to the LEDs. I wrote the number 255 to the LEDs, this turns all LEDs on. This will be used to indicate the simulation has started.

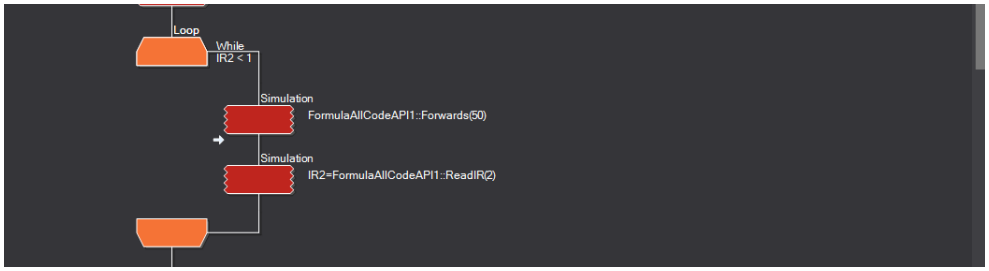


I then added the loop to set the number of iterations the programme does. In the loop settings, I selected loop count and count using the variable “count” that I declared earlier. I selected count to 5 so it runs 5 times, but this can be changed depending on how many obstacles you want the buggy to avoid.

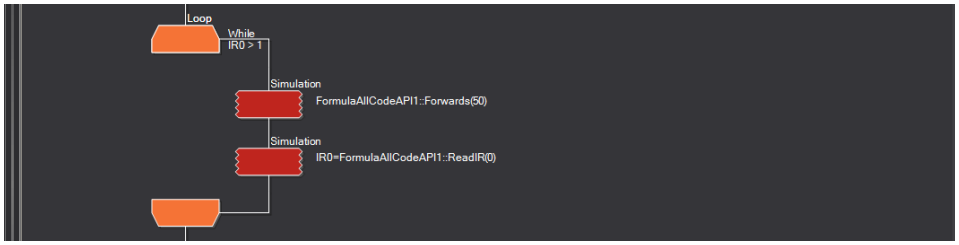


Inside the loop, I used the simulation macro “LCD print number”. Instead of printing a specific number, I made it print the variable count so each time the loop iterates the number goes up to indicate how many obstacles have been avoided. I then used a simulation macro to read the front sensor (sensor 2) and store the output in variable (IR2).

Then I created a while loop. The condition of the loop is while IR2<1, execute the code inside the loop. Inside the loop, I inserted a simulation macro to go forward a distance of 50. Then it reads the sensor again before checking the condition of the loop again. The function of this loop is to check if anything is blocking the front IR sensor, while it is not it moves forward in steps.

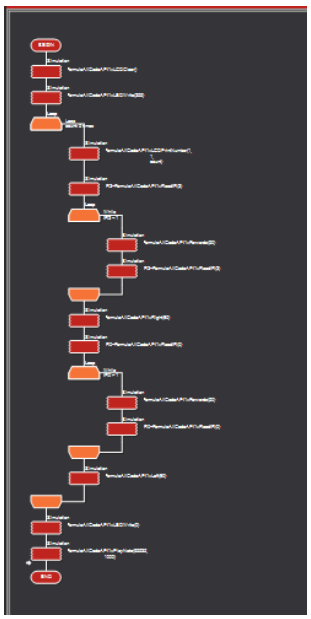


Once the condition of loop above is no longer met, the simulation macro is called to rotate the buggy 90° to the right and then read the sensor on the left (sensor 0). The output from the sensor is stored in the variable “IR0”. Another while loop then starts similar to the last one but the condition is while IR0>1 and sensor zero is read instead of sensor 2. The function of this loop is to keep moving forward until there is no longer and obstacle detected on the left.



When the condition of the loop is no longer met, the simulation macro is called to rotate the buggy 90° to the left back to the buggy’s original position. This is the end of the original loop, which will repeat 4 times.

Outside of the count loop (at the end), a simulation macro is called to write 0 to the LEDs which turns them all off and another one to play a note at 65535Hz for 1000ms. This indicates the simulation has finished.



Testing and results

To test the simulation, I turned the Bluetooth on, on my computer and switched on the buggy. I selected API mode on the buggy and run simulation on flowcode.

The simulation mostly did what I expected, however I did notice some problems with the sensors. Both the front and left sensors would occasionally detect an object that wasn't there or not detect an object that was. To try and fix this issue, I tried changing the conditions of the while loop to calibrate the sensitivity. After a lot of trial and error, trying numbers across the whole range I realised that the sensors were highly inaccurate and no matter the values, the sensors would run into issues. Because of this I stuck with the original loop conditions.

Conclusion

In conclusion, I think the project was a success for the most part. It ran through the simulation completely as expected until it came to reading the IR sensors which I eventually determined to be inaccurate.

If I were to make improvements on this project in the future, I would use another set of external sensors and attach them to the buggy. The new external sensors would have to be more accurate than the current ones so I would recommend ultrasonic sensors instead of IR because ultrasonic sensors detect objects using sound waves rather than light, which could have been a factor in the inconsistent readings from the IR sensor. Although this solution is more expensive, it would be far more reliable and consistent. Another alternative would be to use a different type of buggy with different hardware specification.

Instead of using a for loop to specify the amount of obstacles the buggy needed to avoid, a better solution would have been to have some kind of switch I could press to tell the buggy when to stop avoiding obstacles.

References

https://www.matrixtsl.com/wiki/index.php?title=Component:_Formula_AllCode_API_(Mechatronics) [accessed 19/03/19]

https://www.matrixtsl.com/resources/files/misc/CP5894%20Formula%20AllCode%20Robotics%20Course.pdf [accessed 19/03/19]

https://www.maxbotix.com/articles/ultrasonic-or-infrared-sensors.htm [accessed 19/03/19]