Software Documentation

James Lecomte jccl4 + Tom Hill th621

CODE

Permalink to Github repo:

https://github.com/Hamezii/m101-IDP/blob/80b2492b2193943c890ced5abd1d3d18f783d01f/Software/IDPmain/IDPmain.ino

STRATEGY

ALGORITHM AND STATE

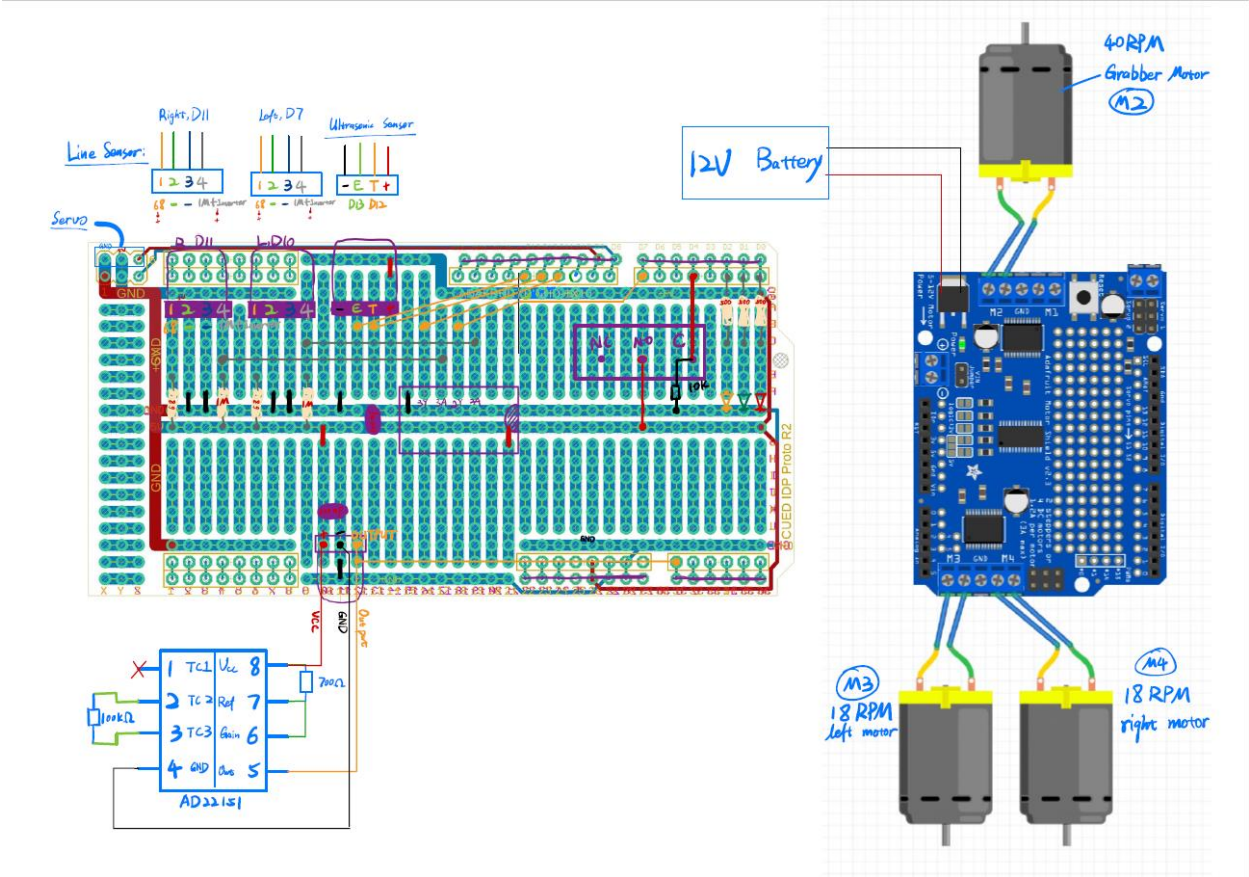
For the robot to keep track of its location and its current place in its behaviour algorithm, we implemented a Finite State Machine into the program. This allowed us to explicitly separate the robot behaviour during the different stages of execution into their own state. This also allowed easy translation from initial algorithm ideation to software implementation.

PATHING

For pathing, we wanted to allow the robot to be aware of its position on the table, and to do so we want line sensors to detect lines. We want the robot to be able to follow lines and detect perpendicular lines it comes across, so we designed a system to satisfy both criteria. We put two line-sensors on the front of the robot since it is the minimum amount needed to gather enough information.

EXECUTION

SYSTEM DIAGRAM

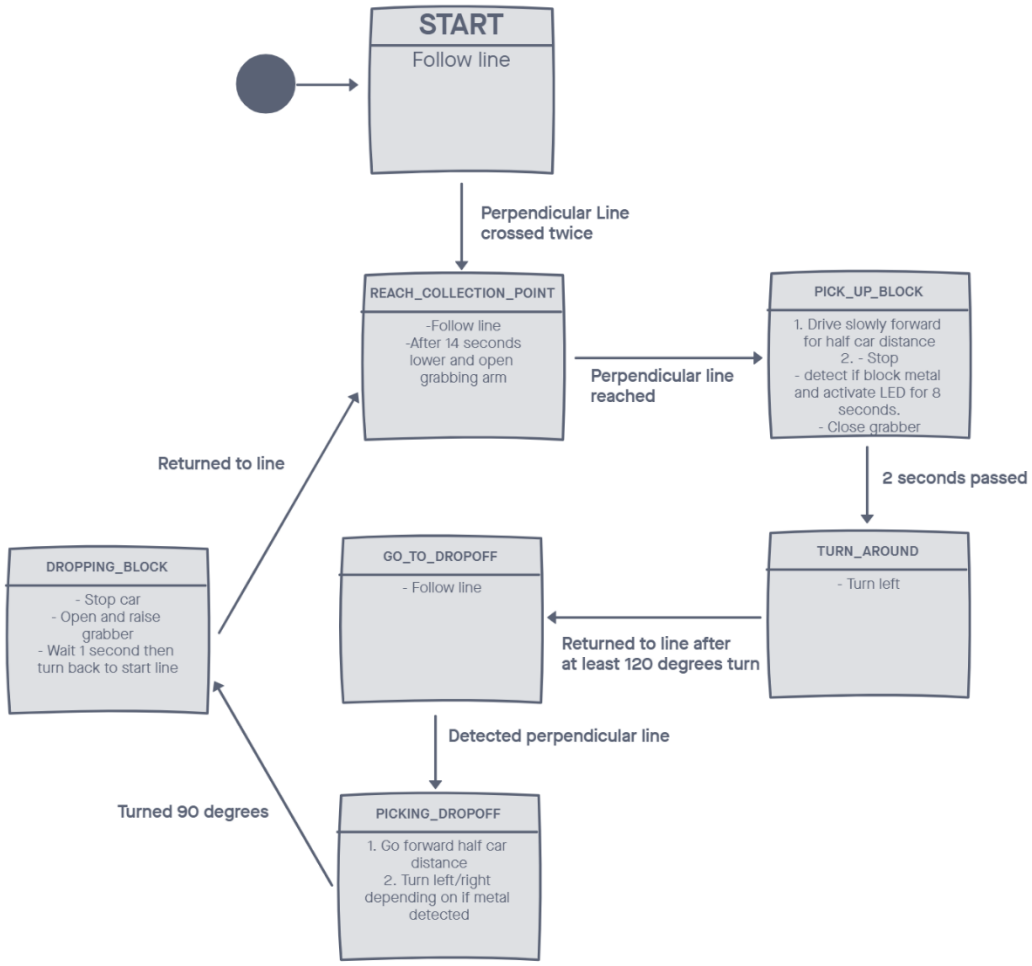


MAIN STRUCTURE

The program consists of subroutines organised using a state machine that enable navigation with line following, object detection with the ultrasonic sensor, metal detection, object grabbing and lifting, placing objects into the appropriate boxes, etc.

The behaviour of the robot at any given time is driven by the FSM; at any arbitrary time-step, using a combination of current state and inputs into the Arduino, the robot will modify its state and behaviour, calling subroutines to do so.

STATE DIAGRAM



PARALLEL PROCESSES

We wanted to allow continuous robot output behaviour and state management in parallel with continuous sensor readings (for running averages, see below). To do this we made sure subroutines don't block the execution of the program by running for a non-trivial period, for example by using delays in a for loop for a continuous motion, as this essentially blinds the robot during its execution. The FSM-centric design allowed for continuous behaviour over multiple program cycles.

LINE FOLLOWING

The line follower circuit includes two OPB704 line sensors which uses an LED, emitting IR light, and a phototransistor. When on the line a low voltage is output and when off the line a high voltage is output. This signal enters an inverter before being input to the Arduino for use in the line following program.

The line-following logic follows a simple negative feedback loop behaviour. When both line sensor inputs are low, the robot moves forwards. Then if either line sensor read value is high, the robot would turn in the direction of the high-reading sensor until it was low.

Using this technique, when there is no reading on either line sensor, it can be assumed that the front of the robot is cleanly on top of the line. This works under the assumption that the robot is already on a line before it begins line-following.

Detecting perpendicular lines

When following a line, if both line sensors read high it can be assumed that the robot has crossed onto a line perpendicular to its direction of movement. In practice, this can be used to detect an edge of the different boxed zones on the table, as well as the point at which two lines intersect.

Sensing

To ensure accurate readings from the different sensors on the robot, we made sure to eliminate noise and false readings by implementing a running average for each sensor. We read the value of each sensor every program loop and pushed it into a Queue-like RunningAverage abstract data type, taking the average of the values in the queue when polling sensors for conditional logic.

Using trial-and-error, we found that a good value for size of the running average Queues was 4, as too small would allow a single noisy reading to disrupt the mean reading and too large would cause the robot to respond too slowly to large discontinuous changes in read value i.e. when line sensors pass over a line.