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

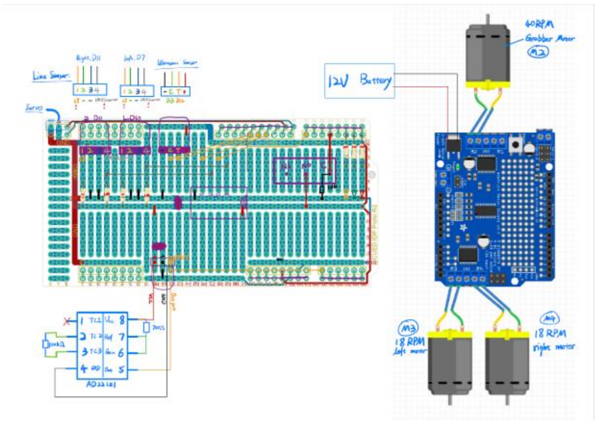
STRATEGY

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

EXECUTION

SYSTEM DIAGRAM

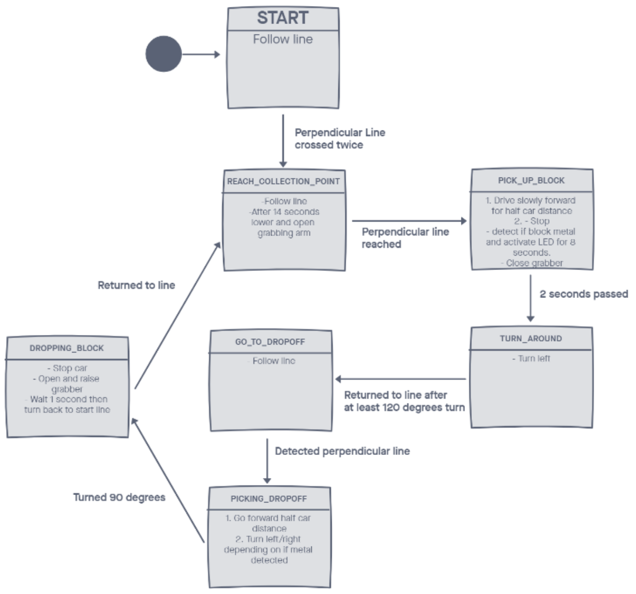


MAIN STRUCTURE

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


START
Follow line
Perpendicular Line crossed twice
REACH_COLLECTION_POINT
-Follow line
-After 14 seconds lower and open grabbing arm
Perpendicular line reached
PICK_UP_BLOCK
1. Drive slowly forward for half car distance
2. - Stop
- detect if block metal and activate LED for 8 seconds.
- Close grabber
2 seconds passed
TURN_AROUND
- Turn left
Returned to line after at least 120 degrees turn
GO_TO_DROPOFF
- Follow line
Detected perpendicular line
PICKING_DROPOFF
1. Go forward half car distance
2. Turn left/right depending on if metal detected
Turned 90 degrees
DROPPING_BLOCK
- Stop car
- Open and raise grabber
- Wait 1 second then turn back to start line
Returned to line

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.

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.