Lab 4
Robot Navigation

MECH2110
Semester 1 2019

Nicholas O'Dell, Johannes Hendriks, Craig Wheeler.

1. Introduction

Saddened by the sale of the Bar Near The Hill and Godfrey's Spanner Bar, mechanical engineering academics Dr Lamb Sevens and Dr Villain Spaghetti are in search of a new Friday lunch. Fortunately the transportation company Noober has made impressive headway with their autonomous vehicle initiative and has decided to trial the prototype vehicle on their food delivery service, Noober-Eats. Dr Sevens and Dr Spaghetti have ordered a generous serving of CFK to be delivered by the new vehicle. You have been tasked with developing a navigation algorithm to deliver lunch to the hungry academics.

In this lab assignment you will:

- Translate navigation objectives into a software design using design tools including UML diagrams and pseudo-code.
- Implement your software design on an Arduino microcontroller and control a simulated vehicle model.

You will be marked on your software design and your implementation.

The lab is worth 10% of your course grade and is graded from 0-10 marks.

2. Control Objective

The vehicle must traverse the track from its starting zone, depicted in Figure 1, to the finish zone. It must:

- Not collide with any obstacles or leave the track.
- Pass through zone 1 and zone 2.
- Finish entirely within the finish zone.
- Take no longer than 60 seconds.
- Be repeatable.
- Your control allocation function will be called every 50 milliseconds.

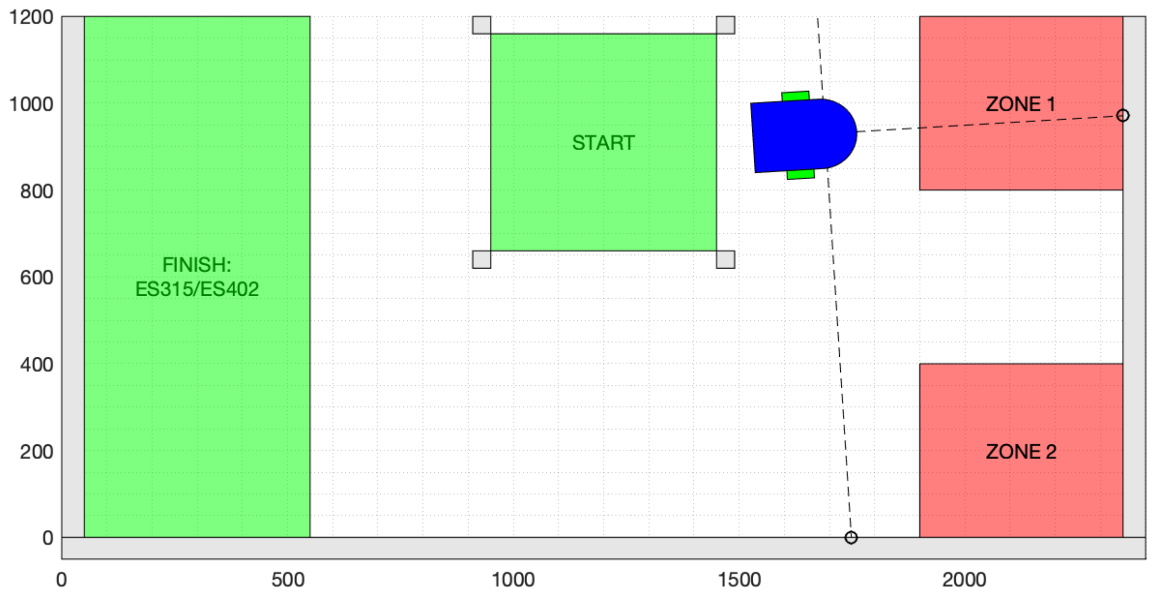

Figure 1: Delivery vehicle depicted in on its route.

3. Simulation Model

The delivery vehicle is depicted in Figure 2, and has the following hardware:

- **Sensors:**
  - Left mounted IR distance sensor.
  - Right mounted IR distance sensor.
  - Forward facing IR distance sensor.

  These sensors return a distance in millimetres, if there is no *hit* within the range of the sensor it returns a value of exactly 1000, its maximum range. The grey walls on the track are 50 mm thick, these are the objects that the sensors can detect.

  The sensors obtain a high quality measurement with very little error. The left and right sensors are mounted 30 mm offset from the central axis of the vehicle, the forward facing sensor is mounted 70 mm back from the front of the vehicle.

- **Actuators:**
  - 2 DC motors mounted in a differential drive configuration.

  In order to control these motors 2 PWM signals whose magnitude is in the range $[0; 255]$ and whose sign represents the direction are used. I.e., a PWM value of 255 represents full voltage in the forward direction as opposed to -255 which represents full voltage in the opposite direction.

Note

A modulo operation is performed on the PWM variables allocated to the motors to ensure its magnitude is between 0 and 255. This is indeed what happens on the Arduino when using `analogWrite(pin,val)`, as we did in Lab 2. Modular arithmetic[a] is similar to counting around a clock face, i.e., $0, 1, 2, \ldots, 10, 11, 0, 1, 2, \ldots$; in our case we have: $0, 1, 2, \ldots, 254, 255, 0, 1, \ldots$. You can avoid any undesired effects by ensuring your PWM variables remain within the correct range by using a simple if statement, or using one of Arduino's built in functions: `constrain()`[b].

```
pwmL=constrain(pwmL,-255,255)
```

[a]https://www.khanacademy.org/computing/computer-science/cryptography/modarithmetic/a/what-is-modular-arithmetic

[b]https://www.arduino.cc/reference/en/language/functions/math/constrain/

To represent real DC motors, the simulated motors have slightly varying parameters which will prevent the vehicle from driving straight if both motors are sent the same PWM signal. The parameters change for each simulation, as is the case in reality; real motors perform differently as the battery discharges and change as each motor wears in and wears out.

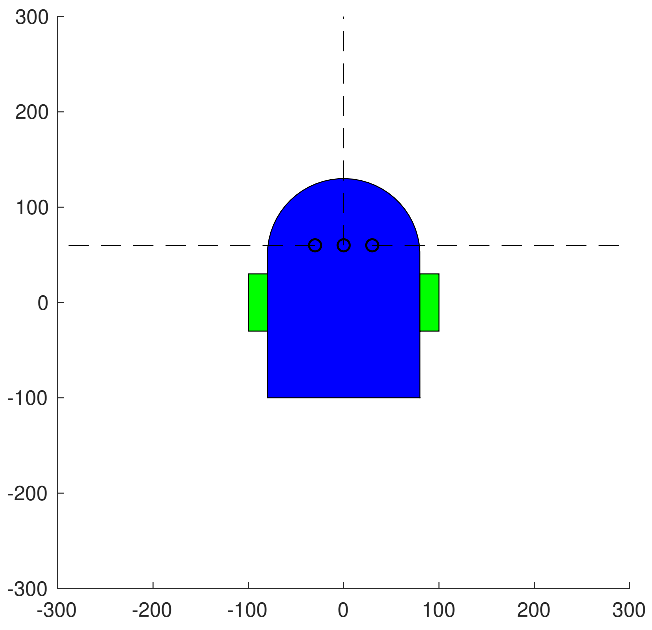

Figure 2: Noober-Eats delivery vehicle; equipped with left, right and forward facing distance sensors, and 2 DC motors in a differential drive configuration.

Tip

The varying motor paramaters for each simulation are partly based on the name of the user profile logged into the computer, therefore your simulations will be consistent among any university computers[a]. If you develop your lab solution on your personal laptop, it is recommended that you bring this to your lab for assessment, otherwise arrive early to adjust your implimented solution based on your new robot's behaviour.

[a]Your username on a university PC is simply your student number.

4. Mandatory Pre-lab (5 marks)

Tip

The following task must be completed before your allocated lab and a hard-copy handed in at the beginning of the lab[a]. You must attach an *Assessment Item Cover Sheet* to the front of your pre-lab work, or you will receive a mark of zero for this part of the assignment.

[a]Or you will not be permitted to complete the lab and will receive zero for the assessment.

Task: Software design

Develop a navigation algorithm that will complete the navigation objectives. You should create a UML diagram that illustrates how your software design will accomplish the navigation objectives. Translate your UML diagram into pseudo-code.

You will be marked on your:

- Sensible selection of states and state transitions.
- Pseudo-code should reflect and expand upon the UML diagram.
- Neatness and clarity of both the UML diagram(s) and pseudo-code.

5. Lab Task

Warning

Extract `lab4_files.zip` before attempting to run the Matlab script or compile the Arduino code.

Use Matlab 2016b or later.

In this lab you will use an Arduino to control a simulated robot, this is called *Processor In the Loop* (PIL). The computer simulates the real robot and environment, this includes the motors and sensors, while all control actions, in our case PWM duty cycle, are calculated on the Arduino and handed back to the computer.

An optional, but recommended, intermediate step if you do not own an Arduino[1] is to implement your algorithm in Matlab at home first before implementing the solution in Arduino code. Instructions for the Matlab only simulation and the full PIL simulation are below in Sections 5.1 and 5.2 respectively.

If you choose to first implement your solution in Matlab before Arduino, you should endeavour to write the code as close to C code as possible. For example:

- In a Matlab `if` statement, brackets enclosing the conditional statement are unnecessary, however will make your code translate over to C easier (and faster!).

[1]Any Arduino, it doesn't need to be the same as the lab, we are only using the serial communication which is common among all Arduino models.

- Ensure you append each line with a semi-colon.
- Remember that array indexing starts at 1 in Matlab but 0 in C, i.e., the first element of an array in Matlab is `array_name(1)`; but in C would be `array_name[0]`.

When writing a line of code in Matlab make sure you are aware of the differences before you start porting it to C. Consider making comments in your Matlab script documenting these differences, and when you do translate your Matlab code to C, do it in a logical order, e.g., one state at a time.

5.1. Matlab Simulation Only

You are provided with a script `runRobotMatlab.m` to simulate the robot with your control law. You should **only** edit the file: `ctrlAlloc.m`.

1. Open Matlab and run the script `runRobotMatlab.m`. Verify that the simulation runs and the robot crashes.
2. Implement your control law in a state machine in `ctrlAlloc.m`. An example of how to implement a state machine in Matlab is already in the file. The inputs to this function are the sensor measurements and the outputs are the PWM values; recall the convention from section 3: a PWM value of 255 is full speed ahead, -255 is full speed reverse.

5.2. PIL Simulation (5 marks)

You are provided with a script `runRobotArduino.m` to simulate the robot and its environment. This script communicates via a serial connection to your Arduino. At each control step, i.e., every 50 ms, the script sends the current distance measurements to the Arduino and requests the PWM duty cycles.

Hint
Recall that Matlab simulates all aspects of the environment including time, therefore, **any** use of an Arduino function such as `delay(ms)` is not needed. E.g., you wish to create a 1 second delay; knowing that your function gets executed every 50 ms you should create a counter variable that counts to 20 by adding 1 every time the function is called, see Appendix A for an example.

Within `ArduinoPIL.ino` edit only:

- The function `void ctrlAlloc(void)` located at the bottom of the file.
- The variable[2] `enum states STATE1,STATE2,ETC` located above `void ctrlAlloc(void)`.

Your function interfaces with the simulation in the following way:

- The left, right and forward distance measurements are stored in the variables `yLeft`, `yRight` and `yFront` respectively, these are global variables and can be accessed by your control allocation function. These variables are determined by the simulation and are updated prior to your control allocation function being called. You should not change these variables!

[2] An `enum` data-type, as the name suggests, can be used to create an enumerated list; as opposed to defining the states with `#define` at the beginning of your program (`https://www.programiz.com/c-programming/c-enumeration`).

Example: If you wanted to see if the right distance sensor was returning its maximum range:

```
if(yRight==1000){
/*do the things*/
}
```

- The desired PWM duty cycles should be stored in `pwmL` and `pwmR` for the left and right motors respectively; recall the convention from section 3: a PWM value of 255 is full speed ahead, -255 is full speed reverse.

  Example: To make the vehicle spin anti-clockwise on the spot:

```
pwmR = 255;
pwmL = -255;
```

Follow these steps carefully:

1. Plug the Arduino into the computer, no other hardware is needed.
2. Open the Arduino IDE and upload the template sketch: `ArduinoPIL.ino`.
3. Make sure the Arduino Serial Monitor is not open, ensuring that Matlab will be free to communicate with the Arduino.
4. Replace `'COM3'` port in line 7: `sobj = 'COM3';`, with the Arduino's serial port; which can be determined from the Arduino IDE.
5. Run `runRobotArduino.m` which will connect to the Arduino's COM port and begin the simulation. Verify that the simulation runs and the robot crashes.
6. Every time you finish making a change to `void ctrlAlloc(void)` and would like to run the simulation:
   a) Upload the sketch to the Arduino.

   > **Info**
   > Although every effort has been made to ensure the `COM` port is handled correctly by Matlab; if you get the following error:
   > `ser_open(): can't open device"\\.\COM3",`
   > run the Matlab script `killSerial.m` to kill any active serial connections before trying to upload again.

   b) Ensure it compiled and uploaded correctly.

   c) Ensure the serial monitor is closed.

   d) Run `runRobotArduino.m` to run the simulation.

   e) The simulation can be aborted at any time by hitting cancel on the loading bar.

   > **Warning**
   > Clicking cancel on the loading bar should be the only way you abort a simulation.

To achieve full marks your implementation must:

- Use the sensor data to make sensible control decisions (feedback).
- Guide the robot to the finish zone, such that it be completely enclosed in the finish zone and finish in a stationary condition.
- Demonstrate consistency with the UML diagram(s) and pseudo-code.

Submission

Before you leave the lab submit your `arduinoPIL.ino` file to Blackboard using the Lab 4 submission link. Students that do not submit an electronic copy will not receive a mark.

6. Warman Recommendations

Follow the software design procedure outlined in this lab to aid with your software design for the Warman Project.

1. Separate the track into a logical set of states.
2. Write a graphical representation of how the program will flow conceptually (UML diagram).
3. Convert that to pseudo-code.
4. Write the first implementation of real code.
5. Iterate.

A. Example 1 - delay

Suppose you require a delay; in your state machine you could create a state named `DELAY`. With the knowledge that your function is called every 50 ms in simulation, you could create a counting variable and a state transition condition to simulate a delay. A pseudo-code example is shown below in Listing 1.

Listing 1: Delay Example

```
INITIALISE DELAY VARIABLE, COUNTER = 0

...

CASE DELAY
  COUNTER++
  IF COUNTER >= (WAITING_TIME_SECONDS*20)
    STATE = NEXT STATE
  ENDIF
BREAK

...
```