West Virginia University
Lane Department of Computer Science and Electrical Engineering

CPE 313 – Microcomputer Structures and Interfacing
Spring 2019

Micromouse Project

Instructor:
Dr. Powsiri Klinkhachorn
Benjamin Robert Smith
Team Members:
Luis Ernesto Fernández

Morgantown, VW, USA

Table of Contents

Introduction

The Micromouse is a robot whose purpose is to navigate through a random maze, is comprised of 18x18 cm cells, in the shortest amount of time. The basic function of the Micromouse is to travel from the start square (located at one of the corners) to the final square at the center of the maze. As stated by the rules of the IEEE competition, each group is allocated a total of 10 minutes to access the maze from the moment the judge, in the case of the class is the TA, allows the group to start. Time may be used for adjustments between runs and making as many runs as possible. Each run in which a mouse successfully reaches the destination square will be given a run time and the minimum shall be the official time.
For the robot to navigate the maze autonomously it requires a reliable way to measure the distance traversed, keep track of its position and detect walls as it moves. The Micromice used for the laboratory project have fixed specifications that accomplish all three tasks. The Mouse traverses the maze with stepper motors and IR analog distance sensors to detect the walls and provide feedback to center the robot. Thus, the focus is to make the code necessary for the peripheral interface and maze solving.

Problem Statement

The problem was to program the required code to interface with the Micromouse built in peripherals, since the specifications are fixed to the platform provided, along with the algorithm to navigate autonomously a random maze of the specified size.

Project Objectives

- Configure the necessary beans on the HCS12 microprocessor for the internal peripherals required, Timers, ADC, I/O, etc.
- Program the code to interface the microprocessor and the external devices (LCD screen, Stepper motors and IR distance sensors).
- Program and implement a maze solving algorithm (all algorithms are allowed even though the one taught in the course is the modified flood fill algorithm) for a 6x13 maze starting from the bottom left corner to its center in the shortest amount of time.
- Navigate the a random 6x13 maze.
- Achieve the maze in the shortest amount of time as possible

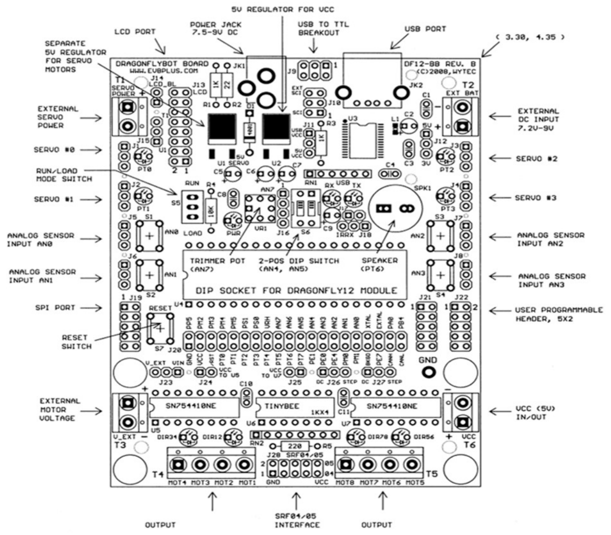
Mechanical Platform and Electrical Platform

Pinout Diagram

The Dragonfly board that the Micromouse uses is broken down into two boards, a controller board containing the minimum circuit for the HCS12 microcontroller (MC9S12C32MFA25) and a carrier board with the power supply, USB interface, power, amplifier stage (required for the motors) and pinout for easier interfacing with analog devices and other peripherals. The board has the following pinout for the microcontroller and the boards peripherals:

Pin Name	Pin #	I/O Usage
PA0	22	EEPROM / LCD RS
PB4	21	EEPROM
PE0	13	-----
PE1	12	SPI Port
PE4	14	SPI Port / 2nd H-Bridge / SRF04
PE7	18	LCD EN
PM0	15	1st H-Bridge
PM1	16	1st H-Bridge
PM2	39	SPI Port / 2nd H-Bridge
PM3	38	SPI Port / 2nd H-Bridge
PM4	37	2nd H-Bridge
PM5	36	2nd H-Bridge
PP5	40	RUN-LOAD Mode Selector
PS0	34	SCI for USB Port
PS1	35	SCI for USB Port
PT0	4	Servo 1 / 1st H-Bridge
PT1	5	Servo 2 / 1st H-Bridge
PT2	6	Servo 3
PT3	7	Servo 4
PT4	8	1st H-Bridge
PT5	9	1st H-Bridge
PT6	10	Speaker
PT7	11	SRF04
PAD0	25	Pushbutton 1
PAD1	26	Pushbutton 2
PAD2	27	Pushbutton 3
PAD3	28	Pushbutton 4
PAD4	29	Accelerometer / DIP Switch S6
PAD5	30	Accelerometer / DIP Switch S6
PAD6	31	Accelerometer
PAD7	32	Trimmer for VR1

A more detailed view of the peripherals can be seen on the Board and schematic views provided.



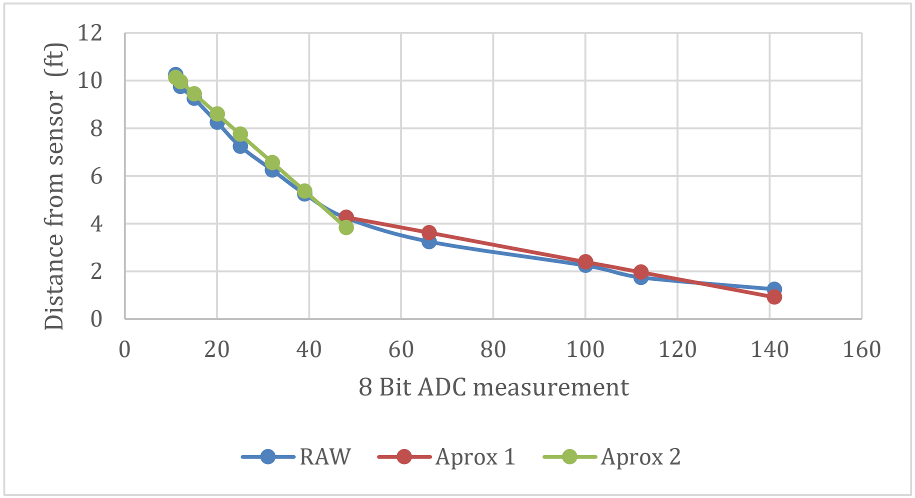
The peripherals used for the Micromouse to navigate the maze are the LCD (4 Data Lines, 1 Read Select and 1 Enable) for debugging purposes like printing the current position, orientation, distance values, etc. , the stepper motors (using the SN754410NE H-Bridge drivers) and 3 ADC Channels for the Sharp IR distance sensors for walls detection.

The Micromouse is designed with modularity first in mind, as can be seen in the two-board design as it is first an educational tool and second a maze solving robot. For this purpose, all its peripherals can be swapped or replaced (good for repairs), ports and external peripherals not required for maze solving, like a buzzer, SPI port, PWM ports, push buttons, etc. and the full pinout of the HCS12 is provided for the user to interact with the hardware.

The fixed hardware platform allows all teams to stand on equal ground, even though not all Micromice work the same, making the focus of the project the software design and not designing and debugging the electronics of the robot. As previously stated, the robot requires a reliable way to measure the distance traversed, keep track of its position and detect walls as it moves and the hardware for maze solving is designed around these three statements.

For a wheeled robot to keep track of the distance traveled a DC motor and an incremental encoder would be an ideal combination to have a fluid and reliable maneuverability with the only caveat being the complicated control loop required to interface the encoder and the wall detection sensors. On the other hand, a stepper motor has a slower movement and requires a driver circuit (two H bridges rather than one) but with the advantage that movement is so reliable that an encoder can be omitted from the system, for this reason the stepper motors on the robot are used. It should be noted that none of the options previously stated solved the problem of the robot sleeping on the surface of the maze, this would depend on the grip of the wheels and the track.

For the detection of walls different sensors can be used both mechanical or electronic, the Sharp IR sensor is of the later. The Sharp analog IR sensor used for both wall detection and distance measurement is an easy device to interface required the microcontroller used has an internal ADC. The main problems are the sensors logarithmic response to distance, it's not linear at all, and the dependency on the lighting, even if the sensors are properly calibrated, they are only warrantied to work on the same lighting conditions.

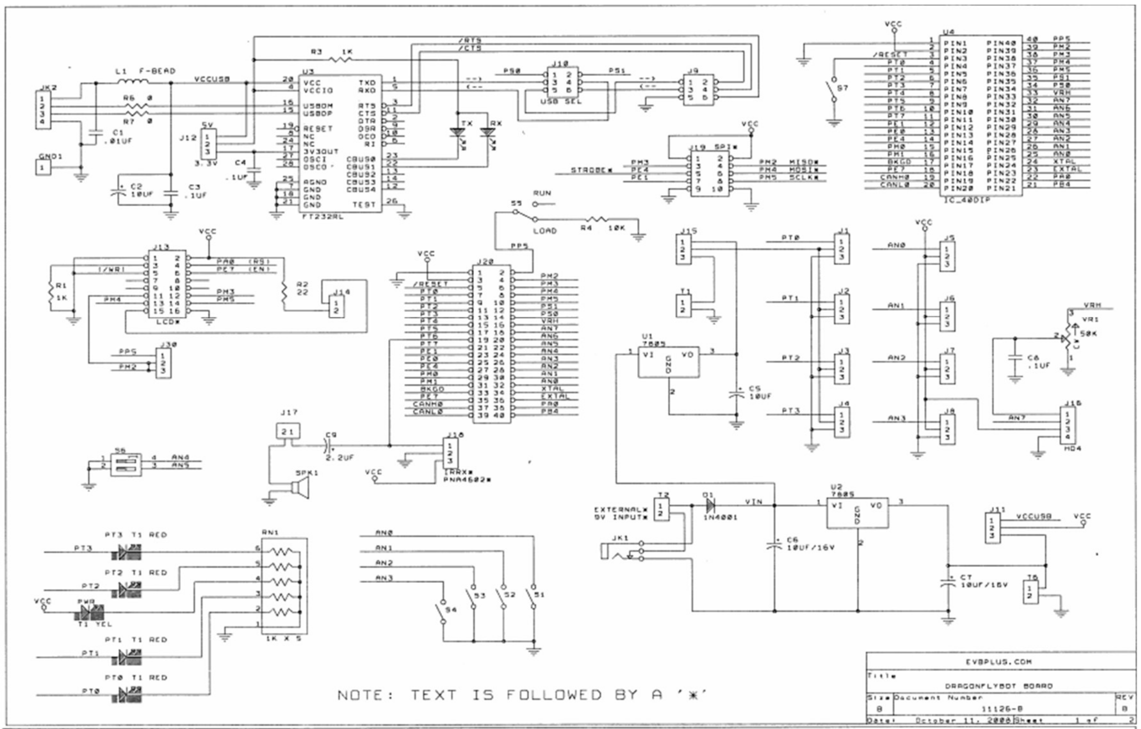


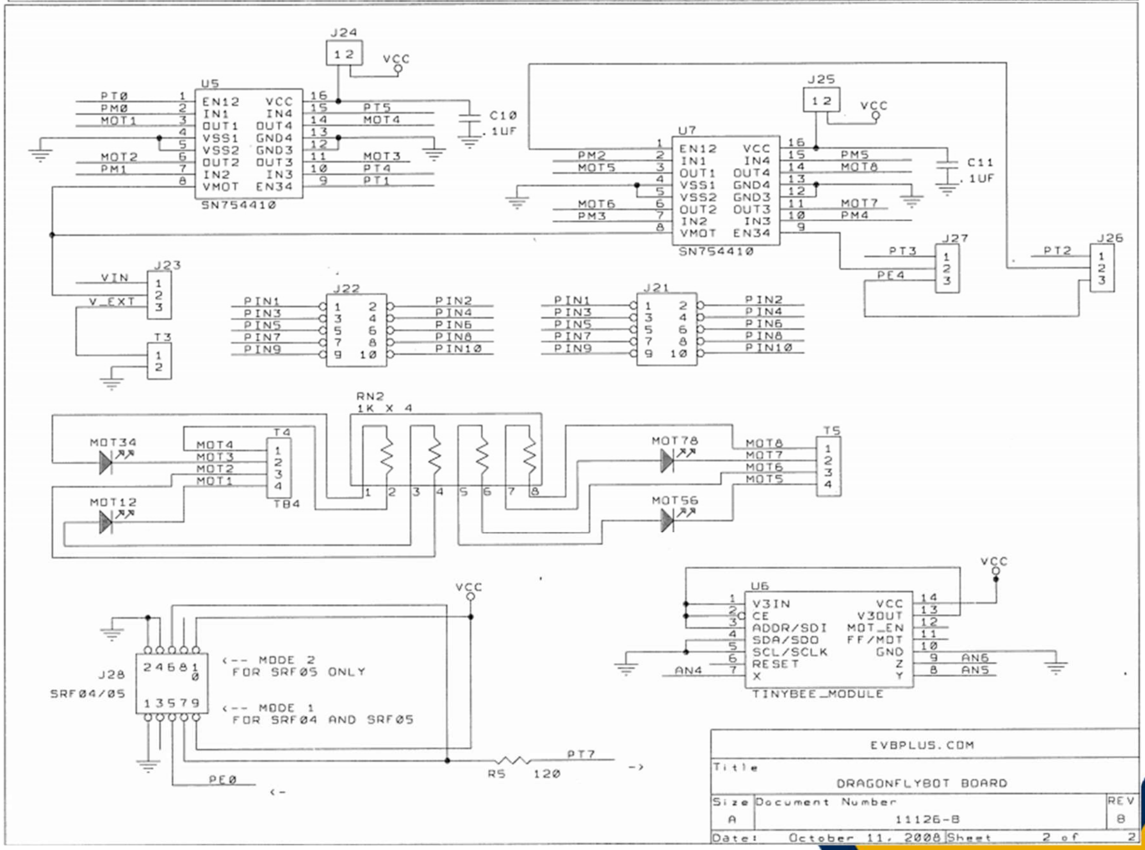
A better solution is employing ultrasonic sensors so external noise from the environment wouldn't affect the performance of the robot with the draw back of interfacing with the Echo and Trigger signals with the timing specified on the datasheet.

The last point is keeping track of the robot's position of the maze which is done on software but can be better implemented with additional hardware. Currently when the robot turns or moves one cell its position is recorded on software without any kind of feedback or correction. A SPI or I^{2}C magnetometer and gyroscope can be used to better keep track of the Micromouse's position and correct any turning error.

Overall the Micromouse fixed hardware provides a reliable and easy platform to prototype and develop a maze solving robot. It can be greatly improved with the drawback of the coding and control required being greater than the scope of the laboratory.

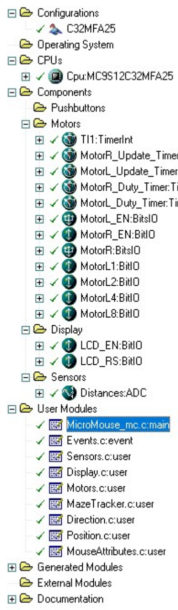
Micromouse Schematic



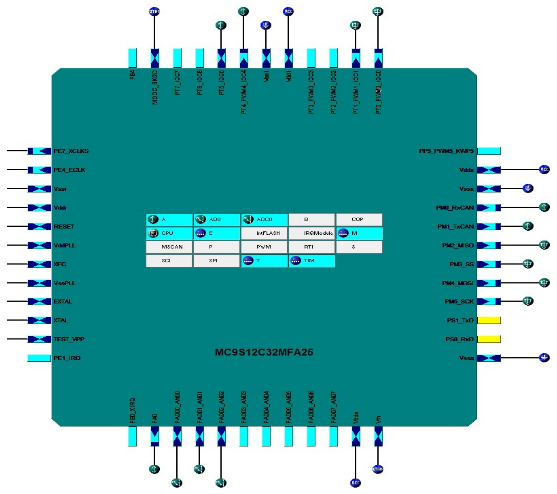


Software Platform

Beans and Methods



```
Configurations
  C32MFA25
Operating System
CPUs
  Cpu:MC9S12C32MFA25
Components
  Pushbuttons
  Motors
    TI1:TimerInt
    MotorR_Update_Timer:TimerInt
    MotorL_Update_Timer:TimerInt
    MotorR_Duty_Timer:TimerInt
    MotorL_Duty_Timer:TimerInt
    MotorL_EN:BitsIO
    MotorR_EN:BitIO
    MotorR:BitsIO
    MotorL1:BitIO
    MotorL2:BitIO
    MotorL4:BitIO
    MotorL8:BitIO
  Display
    LCD_EN:BitIO
    LCD_RS:BitIO
  Sensors
    Distances:ADC
User Modules
  MicroMouse_mc.c:main
  Events.c:event
  Sensors.c:user
  Display.c:user
  Motors.c:user
  MazeTracker.c:user
  Direction.c:user
  Position.c:user
  MouseAttributes.c:user
Generated Modules
External Modules
Documentation
```



- Internal HCS12 Components required:
- TI1: TC4 on the HCS12. 55ms Timer dedicated to a PID interrupt for realignment of the Micromouse while traversing the maze.
- Distances: PAD00_AN00 though PAD02_AN02 on the HCS12. 3 Channel 8-Bit analog to digital converter required to interface with the Sharp analog IR distance sensors.
- MotorR_Update_Timer: TC2 on the HCS12. 4 ms low priority interrupt required to commute the stepper motor sequence (period time is modified according to the PID controller to accelerate or deaccelerate to maintain the robot centered on the maze).
- MotorL_Update_Timer: TC3 on the HCS12. 4 ms low priority interrupt required to commute the stepper motor sequence (period time is modified according to the PID controller to accelerate or deaccelerate to maintain the robot centered on the maze).
- MotorR_Duty_Timer: TC1 on the HCS12. 200 us low priority interrupt required to reduce the current sent to the motor drivers.
- MotorL_Duty_Timer: TC0 on the HCS12. 200 us low priority interrupt required to reduce the current sent to the motor drivers.

Stepper Motor control signals:

- MotorR_EN: PE4 on the HCS12. Right stepper motor H-bridge driver enable signal.
- MotorL_EN: PT0 and PT1 on the HCS12. Left stepper motor H-bridge driver enable signal.
- MotorR: PM2 through PM5 on the HCS12. Right stepper motor 4 commutation control signals required to activate the steps in the commutation sequence. For the bipolar stepper motors used with the SN754410NE H-Bridge the commutation sequence is 0x01,0x09,0x08,0x0C,0x04,0x06,0x02,0x03
- MotorL1:PM0 on the HCS12. Left stepper motor commutation control signal, bit 0.
- MotorL2: PM1 on the HCS12. Left stepper motor commutation control signal, bit 1.
- MotorL4: PT4 on the HCS12. Left stepper motor commutation control signal, bit 2.
- MotorL8: PT5 on the HCS12. Left stepper motor commutation control signal, bit 3.

LCD control and data signals:
The LCD R/W control signal is grounded as the HCS12 is used only to send data to the LCD registers and not to read data from them.

- LCD_EN: PE7 on the HCS12. LCD Enable control signal
- LCD_RS: PA0 on the HCS12. LCD register select control signal to change between instruction and data registers.
- MotorR (LCD Data): PM2 through PM5 on the HCS12. The LCD is operated in 4-Bit mode with the data lines multiplexed with the right stepper motor requiring the program to stop the Micromouse and disable the stepper motor timer interrupts to communicate with the LCD.

Maze Solving Algorithms

Navigating and solving the maze are accomplished using two separate algorithms. Both use intersections as the primary point of reference, where an intersection is defined as any cell that does not exactly contain a wall to the left of the mouse, a wall to the right, and no wall ahead. So, no calculation is done while the bot is moving through a hallway. An Intersection struct is used to store information about each intersection encountered (discussed in “MazeTracker.c” in the Functions section.)

Navigating is done using Tremaux’s algorithm, which works as follows:

- Upon encountering an intersection, mark the direction the bot just came from (i.e. the opposite direction it is facing)

- Give each direction containing a wall two marks
- Mark the direction that is chosen next
- Always prefer paths with no marks
- Do not take any paths with two marks
- If all possible paths are marked, reverse unless the way the bot came from has two marks. In this case, choose a path with one mark
- An intersection containing only paths with two marks will never be encountered

Solving the maze is done by building a tree of intersections. Each intersection encountered is given a pointer to the previous visited, called that intersection's "parent." No modifications are made to old intersections (those already storing a parent). When the end is found, a tree will have been constructed where the start cell is the root, and the end cell is a leaf. Returning to the start is as simple as travelling straight up the tree.

To find the shortest path, small modifications are included. The mouse's current depth is stored in each intersection visited. When an old intersection is found, compare the current depth with the one stored. If the stored is larger, then we just found a new path to this intersection that is more efficient. Reinitialize the intersection by overriding the current parent, "from" direction, and depth. Otherwise, set the current depth to the value stored.

Functions

The code was kept organized using separate .h and .c files for each hardware component, as well as for any high-level operations (e.g. solving the maze).

Sensors.c: handle any operations related to distance sensor data

- poll: poll the sensors, convert values, and update error for PID

Display.c: handle any operations with the LCD display

- WriteCom: write a command to the display
- WriteData: write data to the display
- InitLCD: initialize the LCD for operation
- ClearLCD: clear any text from the display
- Displays: display a specified string
- Displayi: display a specified integer
- Displayp: display a specified Position (custom struct)
- Displayd: display a specified Direction (custom enum)

Motors.c: handle any operations with the motors

- MotorsOn: activate power to the motors
- MotorsOff: deactivate power to the motors
- Forward: go forward one maze cell
- Turn: turn to face the specified direction

Events.c: process updates to the motors through timer interrupts

- MotorX_Duty_Timer_OnInterrupt (where X is R or L): send the current step configuration to the relevant motor every 200 microseconds
- MotorX_Update_Timer_OnInterrupt (where X is R or L): increment or decrement the current step configuration every 4 ms. Controlled by a "tick" variable, which is decremented each time the step changes. So, motors will only turn if tick > 0.
- TI1_OnInterrupt: update the actual triggering period of the previous timers. Controlled by PID

Position.c: provide a Position struct, which includes two integers for tracking X and Y positions. Also provide a global "depth" integer, which tracks the number of cells the mouse would visit to return to the start.

Direction.c: provide a Direction enum, including Up, Down, Left, and Right. Also provide a global "local" Direction, which tracks the current direction the mouse is oriented (its "local" orientation).

MouseAttributes.c: provide various customizable values. Useful for quickly adapting the code to a new mouse

- StepsPerCell: the number of motor steps necessary to advance one cell
- StepsPerRotation: the number of motor steps necessary to rotate 90 degrees
- DistanceToSide: the maximum value read by a distance sensor that would still cause the mouse to consider a (left or right) wall there
- ActualDistanceToSide: (poorly named) the sensor value used to "fake" a wall so the PID does not adjust too harshly while passing an open wall
- DistanceToFront: the maximum value read by a distance sensor that would still cause the mouse to consider a (forward) wall there

MazeTracker.c: handle algorithms related to navigating and solving the maze. Uses an Intersection struct to store all known information about each intersection encountered. This includes its depth (number of cells away from the start), the direction from which the mouse first entered, marked paths, and parent intersection.

- solve: solve the maze. Assumes the mouse is in the official starting position
- back: return to the beginning of the maze by blindly following the sequence of directions pulled from the tree. Assumes the mouse is in the end cell
- forth: travel to the end of the maze in a similar fashion as “back,” but use the inverse of the direction sequence. For example, if back followed “Up, Right, Up, Left”, forth will follow “Right, Down, Left, Down”

Testing

The following tests were performed each time a new Micromouse was used and as necessary to readjust the PID controller and Motor constants.

Sensor Reading Testing

Description:
Mouse was tested for consistent response between both sensors.
Measurements for maximum error within one cell and when rotating 90° without a wall on one side.

Resources Required:
Micromouse robot
Measuring tape.
6x13 Maze, just 3 cells required.

Justification:
Ideally both should provide the same response to allow the PID controller to work properly and adhere to the results taken experimentally for the conversion to inches. The sensors don’t provide a linear response so proper tuning is required to obtain the correct distance measurements.

Results:
The sensors deliver similar response around the 2 in mark when properly centered on the maze cell. The maximum error the sensors can read is around 3.25 in depending on the robot in use and the ambient lighting conditions. In the case of some Micromice the sensors yielded inconsistent results between each other which caused the PID controller to be fruitless.

Displacement Constants Tuning

Description:
Mouse was tested for 7 in forward movement, the equivalent to 1 cell inside the maze, 90° rotation and 180° rotation.

Resources Required:
Micromouse robot
6x13 Maze, just 2 cells required.

Justification:
Tuning the displacement constants, steps required for the previously mentioned movements, used by the code is required as the Micromouse has no feedback system for the motors in terms of displacement even though steppers are very reliable some of the robots require more of fewer steps.

Results:
stepsPerCell = 330
int stepsPerRotation = 180

PID Controller Testing

Description:
Mouse was tested while moving trough the maze for alignment using the sensor readings. The proportional gain P is adjusted accordingly to have the required amount of speed compensation on the stepper motor timer interrupts MotorR_Update_Timer and MotorL_Update_Timer.

Resources Required:
Micromouse robot
6x13 Maze
Error sensor measurements, maximum and while rotating.

Justification:
The PID controller implemented allows the mouse to realign itself due to the different speed response of both motors. The controller used is a proportional one due to its simplicity, PID and PD controller were not implemented due to the erratic response derivative gain has. The controller parameter need to be tuned according to the response of the robot currently in use.

Results:
P = 50

Maze Solving Testing

Description:
Mouse was placed on one of the starting points with the maze in its current state to test its capabilities on random mazes, the randomness factor is provided by the changes made by other teams testing and modifying the maze.

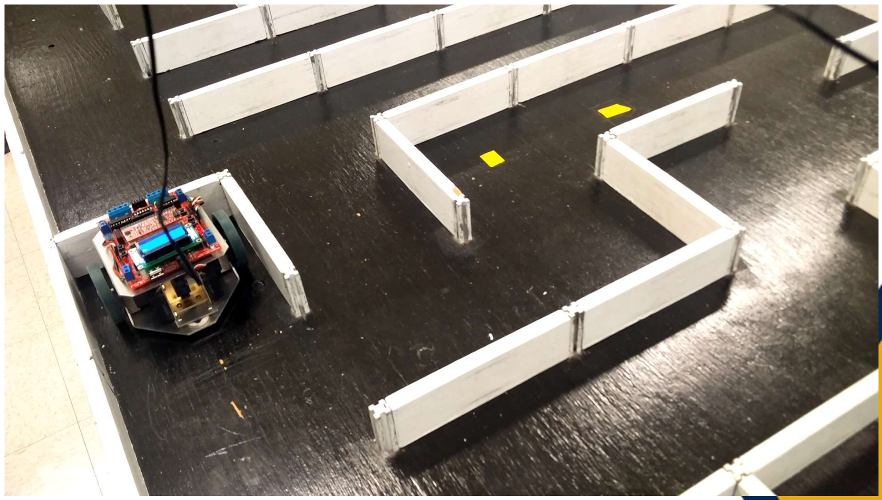
Resources Required:
Micromouse robot
16x3 Maze

Justification:
The objective of the project is for the robot to reach the center of the maze in the least amount of time, it needs to find the shortest route based on the recorded information.

Results:
As expected, the implementation of the Trémaux algorithm on the Micromouse allows it to reach the center of the maze and return to the starting point eliminating the closed paths. The Mouse requires "assistance" to compensate for the PID controller overcorrecting at times as it happened with other teams.

Results

The Micromouse was capable to steer and traverse the maze from the starting point to the center with both the Trémaux algorithm and Flood Fill with better results on the first one. Due to the inconsistencies on sensor response the PID controller wasn't completely capable of maintaining the mouse perfectly centered causing it to steer in to the walls at times. Although the mobility of the robot required further



refinement and adjustment tailored specially to the Micromouse currently used, the maze solving logic tested proved to be accurate and consistent even if the maze was modified. As shown in the presentation, the Micromouse successfully solved the 6x13 maze while "assisted" to correct the steering and prevent crashes and returned to the starting position given the shortest route found and eliminating the closed paths and backtracking of the first run.

Conclusion

Development of the Micromouse robot came with various obstacles we had to overcome, such as the fixed hardware, inconsistent response from motors and sensors between Micromice and the coding required to solve any random maze. Since the hardware platform was already provided the focus of the project shifted to software development with the necessary functions to interface with the peripherals developed throughout the semester and the final maze solving functions developed on the last couple weeks.

The Micromouse competitions use a much bigger maze 16 x16, more than 3 times bigger, which would require a more efficient implementation than the current one. The Micromouse is capable of solving any random maze and determine the shortest return path back to the start from the whole path the robot traveled while avoiding crashing into the walls (the performance in this regard is dependent on the Micromouse used as some sensors have a greater drift than others making additional manual tuning and correction required).

The project meets the objective of the laboratory as it integrates the work of the entire semester interfacing with the internal and external hardware peripherals (ADC, Timers, LCD, etc.).