Group Fab Lab

Final Group Report

Authors:

Ícaro Almeida (14004456)

Leonardo Jurado (10023227)

Sanyog Chhetri (13021702)

Túlio Dapper e Silva (14004454)

Bristol, 24 May 2015

Contents

Section 1

Introduction

The micromouse project consists of designing, building and testing a mobile robot. The system is a self-contained robot able to negotiate a maze in the shortest possible time. In the micromouse project rules, the maze is composed by 16x16 squares; the goal (2x2 square) is placed in the centre of the maze. Starting from a corner of the maze, the micromouse needs to complete the maze without any external help. For completing the maze, firstly, the micromouse needs to find the centre. After that, the robot needs to return to the start position and then execute the fast route to the centre.

The work involves a huge number of skills that is included in a mobile robot. The group needs to work with programming for embedded systems, design and build of PCB boards, deal with sensor components and solve mechanical issues. This document includes a brief explanation about how the software and hardware were built and how the group has been distributed tasks; in addition, it also includes the problems encountered and most suitable approaches.

Conceptual Design

The micromouse was designed in a modular approach. As a consequence, boards could be more easily tested, modified and hardware issues could be solved separately. In addition, a sensor board for testing could be built before designing and building the other boards, which enabled the use of the Firecracker as the initial testing platform. At the end, as the system was modular and the group faced an unexpected problem with the final sensor board, the testing sensor board was still being used.

Four different boards were designed and built: power supply board, main board (containing the microcontrollers), sensor board and motor board. In order to separate the boards inside the micromouse, mechanical spacers were used. In total, the micromouse ended up with 3 floors. In the first floor, the micromouse has two boards: the sensor board and the motor board. The sensor board was placed on front of the motors for taking readings. The motor board was place in the back of the motors. The main board is located in the second floor; the last floor is used for the supply board.

On the supply board, a linear low dropout regulator was used with an appropriate heat sink; reverse-polarity protection diodes and appropriate capacitors (as on the datasheet) were also used. Two buttons and four LEDs were also placed on this board.

In order to store the programs into the microcontrollers, the group opted to design an external board that can be attached to the main board via an Ethernet cable. On the external board, there are switches to toggle between PIC 1 and 2. In both PICs, the PGC and PGD pins were exclusively used for the burning process, avoiding switches; therefore it was necessary to use the alternative UART ports.

On the motor driver board, the IC L293DNE was used as it supports the current drained by the motors and also has diodes built-in. The only protective components attached outside were the decoupling capacitors recommended on the datasheet and a unidirectional TVS as it is faster than the bidirectional ones. A TVS is basically a diode with a lower breakdown voltage, so it was used in addition to the internal diodes to protect the circuit from over voltages generated by the motor coils as a consequence of the back-EMF.

On the sensor board, transistors were used to switch each IR LED due to the high current they drain, which is nearly a hundred milliamps each.

Project Planning

The group is composed of four members. In order to execute the micromouse work, different tasks were assigned for each member. As a consequence, each member was responsible for searching, developing, presenting and documenting their own part of the project. During these stages, the other members were completely willing to help. The division of the tasks were made as shown in Figure 1.

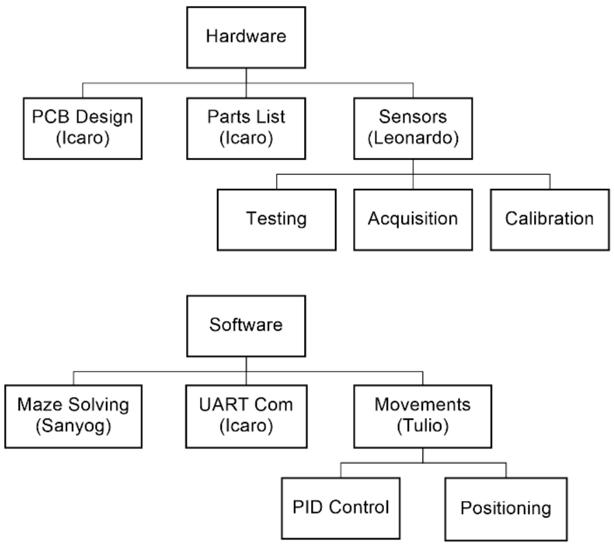
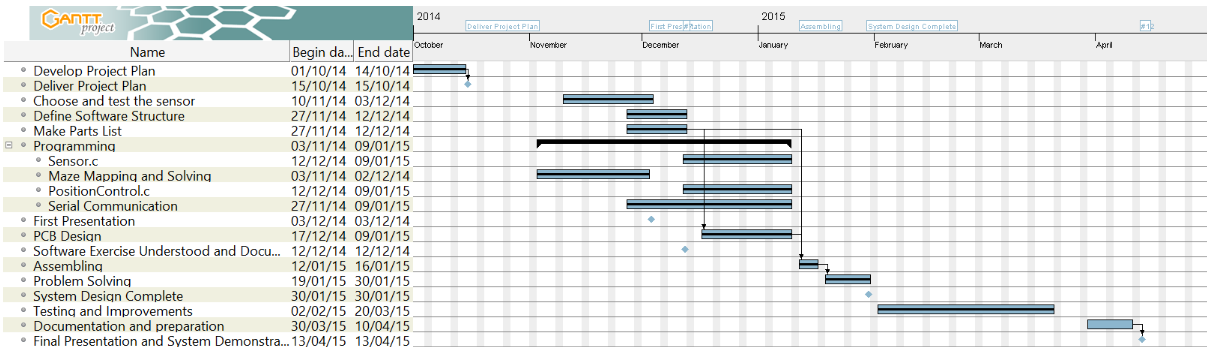


Figure 1: Division of the tasks between members.

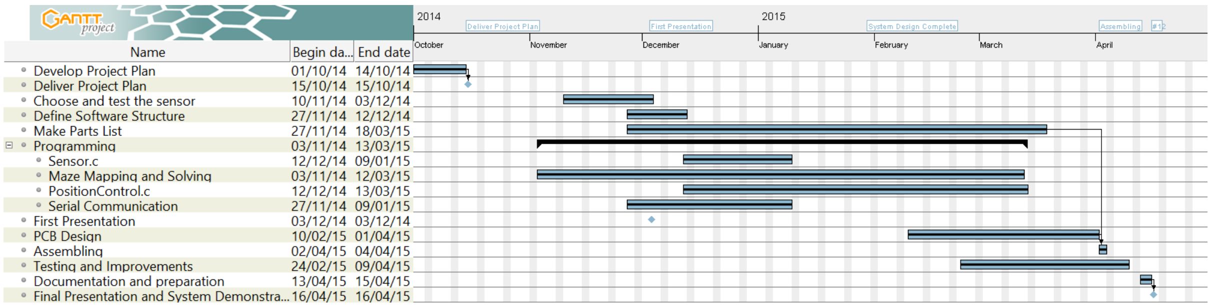
As the project had deadlines to be respected, tasks should be executed within certain period of time. In order to guarantee enough time for all tasks, a Gantt chart were built. The first chart built is depicted in Figure 2. As can be seen in the figure, software structure (creating headers with main constants and functions) and parts list should have be done by the end of December. In addition, all the embedded programming should be finished by the middle of January. The decision was based on having a long time for “Testing and Improvements”, which should have two months of work. However, the work was not executed as planned.



Name	Begin da...	End date
Develop Project Plan	01/10/14	14/10/14
Deliver Project Plan	15/10/14	15/10/14
Choose and test the sensor	10/11/14	03/12/14
Define Software Structure	27/11/14	12/12/14
Make Parts List	27/11/14	12/12/14
Programming	03/11/14	09/01/15
Sensor.c	12/12/14	09/01/15
Maze Mapping and Solving	03/11/14	02/12/14
PositionControl.c	12/12/14	09/01/15
Serial Communication	27/11/14	09/01/15
First Presentation	03/12/14	03/12/14
PCB Design	17/12/14	09/01/15
Software Exercise Understood and Docu...	12/12/14	12/12/14
Assembling	12/01/15	16/01/15
Problem Solving	19/01/15	30/01/15
System Design Complete	30/01/15	30/01/15
Testing and Improvements	02/02/15	20/03/15
Documentation and preparation	30/03/15	10/04/15
Final Presentation and System Demonstra...	13/04/15	13/04/15

Figure 2: Gantt chart before executing the project.

At the end, the amount of time spent for each task is depicted in Figure 3. As can be seen, some tasks took longer than expected. The main reason for those delays is that the assigned duration for some tasks were not enough for the work load. The group were not expecting to have problems that came from the lack of experience of the members. In addition, most of the components were being frequently changed during the process. As a consequence, the parts list was not completed in time; without the components, assembling and testing were also being procrastinated. As a micromouse prototype was being used for testing, the group was waiting as much as possible in order to design and build the final version.

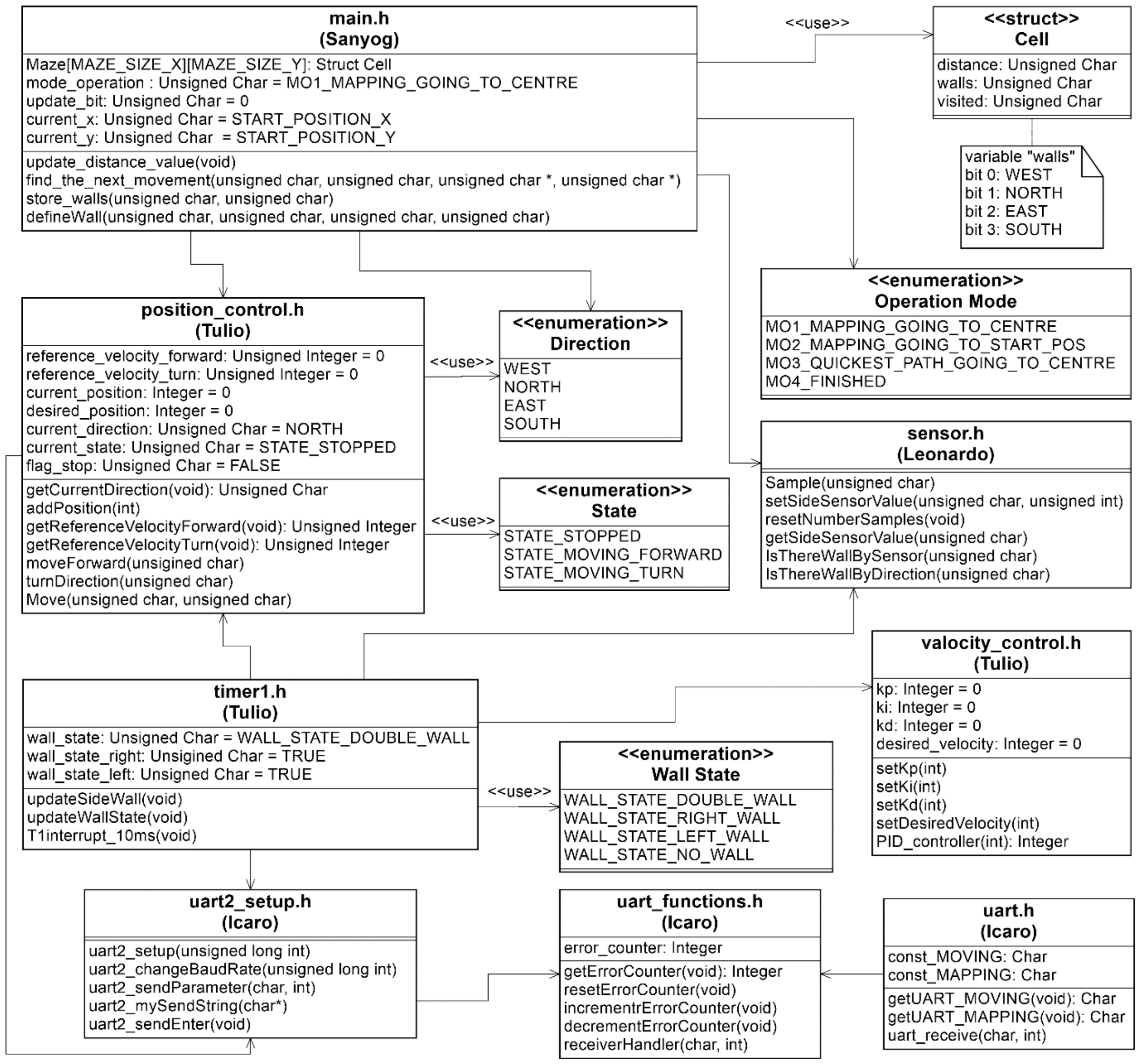


Name	Begin da...	End date
Develop Project Plan	01/10/14	14/10/14
Deliver Project Plan	15/10/14	15/10/14
Choose and test the sensor	10/11/14	03/12/14
Define Software Structure	27/11/14	12/12/14
Make Parts List	27/11/14	18/03/15
Programming	03/11/14	13/03/15
Sensor.c	12/12/14	09/01/15
Maze Mapping and Solving	03/11/14	12/03/15
PositionControl.c	12/12/14	13/03/15
Serial Communication	27/11/14	09/01/15
First Presentation	03/12/14	03/12/14
PCB Design	10/02/15	01/04/15
Assembling	02/04/15	04/04/15
Testing and Improvements	24/02/15	09/04/15
Documentation and preparation	13/04/15	15/04/15
Final Presentation and System Demonstra...	16/04/15	16/04/15

Figure 3: Gantt chart after executing the project.

Controller Design

The software were designed in order to facilitate that each member could develop a part separately. For that reason, meetings were needed in order to create headers and set up how methods were interacting each other. The Figure 4 depicts, using UML class diagrams, how main headers were designed and related each other.



main.h
(Sanyog)
Maze[MAZE_SIZE_X][MAZE_SIZE_Y]: Struct Cell
mode_operation : Unsigned Char = MO1_MAPPING_GOING_TO_CENTRE
update_bit: Unsigned Char = 0
current_x: Unsigned Char = START_POSITION_X
current_y: Unsigned Char = START_POSITION_Y
update_distance_value(void)
find_the_next_movement(unsigned char, unsigned char, unsigned char *, unsigned char *)
store_walls(unsigned char, unsigned char)
defineWall(unsigned char, unsigned char, unsigned char, unsigned char)
<<use>>
<<struct>>
Cell
distance: Unsigned Char
walls: Unsigned Char
visited: Unsigned Char
variable "walls"
bit 0: WEST
bit 1: NORTH
bit 2: EAST
bit 3: SOUTH
<<enumeration>>
Operation Mode
MO1_MAPPING_GOING_TO_CENTRE
MO2_MAPPING_GOING_TO_START_POS
MO3_QUICKEST_PATH_GOING_TO_CENTRE
MO4_FINISHED
position_control.h
(Tulio)
reference_velocity_forward: Unsigned Integer = 0
reference_velocity_turn: Unsigned Integer = 0
current_position: Integer = 0
desired_position: Integer = 0
current_direction: Unsigned Char = NORTH
current_state: Unsigned Char = STATE_STOPPED
flag_stop: Unsigned Char = FALSE
getCurrentDirection(void): Unsigned Char
addPosition(int)
getReferenceVelocityForward(void): Unsigned Integer
getReferenceVelocityTurn(void): Unsigned Integer
moveForward(unsigned char)
turnDirection(unsigned char)
Move(unsigned char, unsigned char)
<<use>>
<<enumeration>>
Direction
WEST
NORTH
EAST
SOUTH
<<use>>
<<enumeration>>
State
STATE_STOPPED
STATE_MOVING_FORWARD
STATE_MOVING_TURN
sensor.h
(Leonardo)
Sample(unsigned char)
setSideSensorValue(unsigned char, unsigned int)
resetNumberSamples(void)
getSideSensorValue(unsigned char)
IsThereWallBySensor(unsigned char)
IsThereWallByDirection(unsigned char)
valocity_control.h
(Tulio)
kp: Integer = 0
ki: Integer = 0
kd: Integer = 0
desired_velocity: Integer = 0
setKp(int)
setKi(int)
setKd(int)
setDesiredVelocity(int)
PID_controller(int): Integer
timer1.h
(Tulio)
wall_state: Unsigned Char = WALL_STATE_DOUBLE_WALL
wall_state_right: Unsigined Char = TRUE
wall_state_left: Unsigned Char = TRUE
updateSideWall(void)
updateWallState(void)
T1interrupt_10ms(void)
<<use>>
<<enumeration>>
Wall State
WALL_STATE_DOUBLE_WALL
WALL_STATE_RIGHT_WALL
WALL_STATE_LEFT_WALL
WALL_STATE_NO_WALL
uart2_setup.h
(Icaro)
uart2_setup(unsigned long int)
uart2_changeBaudRate(unsigned long int)
uart2_sendParameter(char, int)
uart2_mySendString(char*)
uart2_sendEnter(void)
uart_functions.h
(Icaro)
error_counter: Integer
getErrorCounter(void): Integer
resetErrorCounter(void)
incrementrErrorCounter(void)
decrementErrorCounter(void)
receiverHandler(char, int)
uart.h
(Icaro)
const_MOVING: Char
const_MAPPING: Char
getUART_MOVING(void): Char
getUART_MAPPING(void): Char
uart_receive(char, int)

Figure 4: Software Design

The microcontroller dsPIC30F4011 supplied has only one quadrature encoder interface (QEI). As the micromouse has two motors to be controlled, the system requires two microcontrollers. One of them is the dsPIC30F4011, responsible for reading all sensors values, controlling one motor, storing maze information and solving it. The second microcontroller (dsPIC30F4012) is responsible for just controlling one of the motors; the desired velocity is sent by the first microcontroller using the UART communication. A different PIC was used in order to avoid overpricing and oversizing of the system, as it has fewer pins.

The design of the software allowed both PICs to share the same libraries, such as PWM, Timer, UART and PID Control. Setups and some other methods work for both PICs; they do not have any change. If any improvement are made in a shared library, it will automatically have effect on both PICs.

The Figure 5 shows the pin diagram of the microcontroller dsPIC30F4011 used as the main PIC. The diagram was obtained from [4]. The Table 1 describes how the pins are being used.

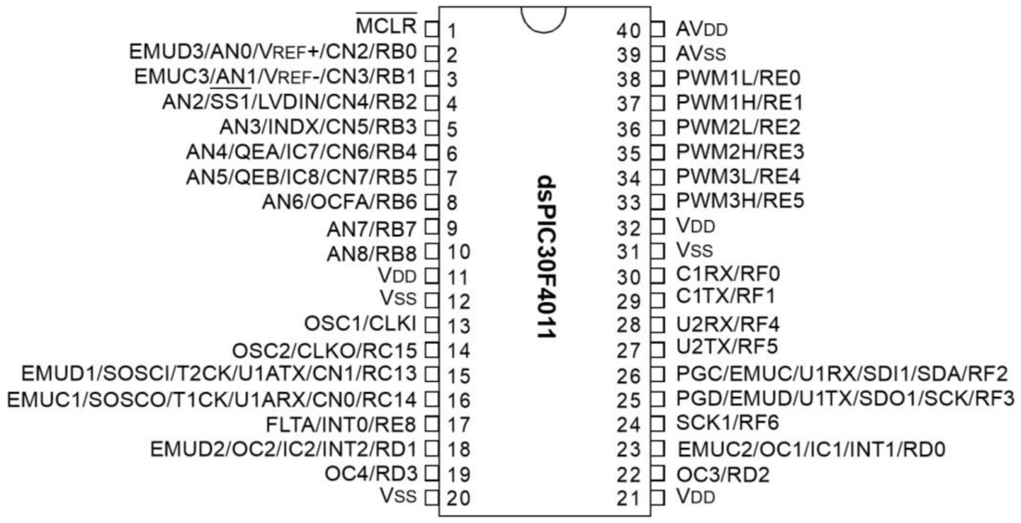


Figure 5: dsPIC30F4011 Pin Diagram

Table 1: Pin allocation for dsPIC30D4011

Port		Pin	Type	Function
PORT B	bit 0	2	Digital Input	Button
	bit 1	3	Analog Input	Sensor (Front Right)
	bit 2	4	Analog Input	Sensor (Diagonal Right)
	bit 3	5	Analog Input	Sensor (Right)
	bit 4	6	Analog Input	Encoder A
	bit 5	7	Analog Input	Encoder B
	bit 6	8	Analog Input	Sensor (Left)
	bit 7	9	Analog Input	Sensor (Diagonal Left)
	bit 8	10	Analog Input	Sensor (Front Left)
PORT C	bit 13	15	Digital Output	Bluetooth (Transmitter)
	bit 14	16	Digital Input	Bluetooth (Receiver)
PORT D	bit 0	23	Digital Output	Sensor Transistor (Right)
	bit 1	18	Digital Output	Sensor Transistor (Left)
	bit 2	22	Digital Output	Sensor Transistor (Front Right)
	bit 3	19	Digital Output	Sensor Transistor (Diagonal Left)
PORT E	bit 0	38	Digital Output	PWM
	bit 1	37	Digital Output	Motor Directional A
	bit 2	36	Digital Output	Motor Directional B
	bit 3	35	Digital Output	LED 1
	bit 4	34	Digital Output	LED 3
	bit 8	17	Digital Output	Sensor Transistor (Front Left)
PORT F	bit 2	26	Digital Input	PGC
	bit 3	25	Digital Output	PGD
	bit 4	28	Digital Input	UART2 Receiver
	bit 5	27	Digital Output	UART2 Transmitter
	bit 6	24	Digital Output	Sensor Transistor (Diagonal Right)

The Figure 6 shows the pin diagram of the microcontroller dsPIC30F4012. Basically, this microcontroller was used only for receiving velocity information and make sure that the second motor is spinning at the desired speed. The Table 2 describes how the pins are being used by the program.

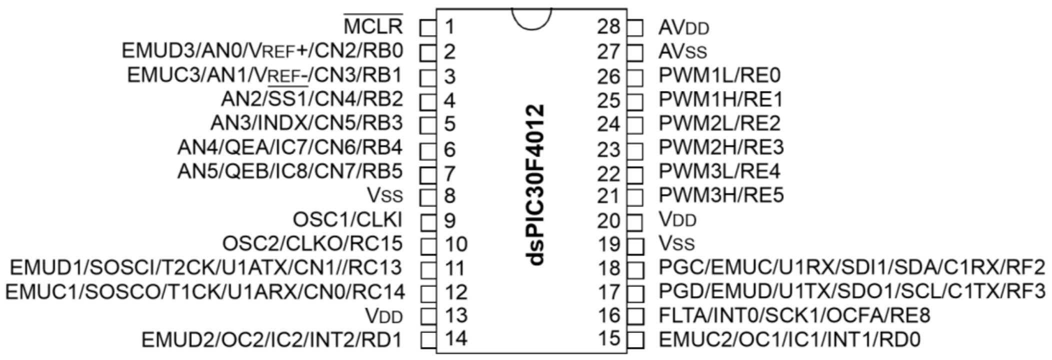


Figure 6: dsPIC30F4012 Pin Diagram

Table 2: Pin allocation for dsPIC30D4012

Port		Pin	Type	Function
PORT B	bit 4	6	Analog Input	Encoder A
	bit 5	7	Analog Input	Encoder B
PORT C	bit 13	11	Digital Output	Alt. UART1 Transmitter
	bit 14	12	Digital Input	Alt. UART1 Receiver
PORT D	bit 0	15	Digital Output	LED 3
	bit 1	18	Digital Output	Button
PORT E	bit 0	26	Digital Output	PWM
	bit 1	25	Digital Output	Motor Directional A
	bit 2	24	Digital Output	Motor Directional B
	bit 8	16	Digital Output	LED 4
PORT F	bit 2	18	Digital Input	PGC
	bit 3	17	Digital Output	PGD

One of the main issues of the project is how to solve the maze. Some researches were done and different methods were found such as A* Search, Using Stacks and Flood Fill Algorithm. A* Search did not seem appropriated to be implemented, as it was apparently difficult to be implemented. Using Stacks could not always find the shortest way to the centre. Using that algorithm, there were a great amount of exceptions and if-else statements in order to achieve the desired results. The best alternative found was the Flood Fill Algorithm.

Briefly, in the Flood Fill Algorithm, each cell contains a number which is the distance value to the goal position; the distance value means how many cells are needed to reach the goal position. For every cell visited, the walls are stored and the distance values are updated. In addition, the next cell to go is always the cell which contains the lowest distance value. Using this method, after storing the wall information, the micromouse can pre-process the map to attain better performance.

Using [2] as reference, the algorithm was written and debugged in MATLAB code. After research, a library that allows a maze to be generated and drawn in MATLAB was found [3]. After modifying that library, the maze could be drawn informing the walls that the micromouse already knows, the distance values and the current position of the micromouse. These visual features obtained made it easier to simulate. The Figure 7Figure 8 shows the maze being drawn during a simulation. The lines highlighted in black are the walls stored. The Hashtag represents the robot.

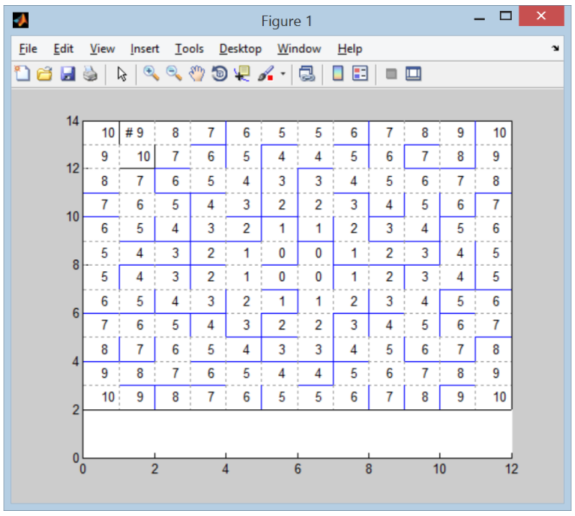


Figure 7: Flood Fill Algorithm being simulated using MATLAB.

The flow chart shown in Figure 8 was developed right after the algorithm properly worked in simulation. In that way, the flow chart is an optimized version of the same code. After that, the C code for the embedded programming was written based on the flow chart.

As can be seen in Figure 8, the logic starts with an initialization. At that stage, the current position is updated to the starting position, the current direction is set to “NORTH” and mode of operation is defined as 1. In addition, the cells, which contains the knowledge about walls, distance to the goal and if the cell has been visited or not, are initialised. After that, the outer walls are stored.

Defining the multi-dimensional array makes it easier to update the distance between the mouse and the goal. Basically, this is achieved by giving the goal a value of zero and incrementing the open neighbouring cell until it reaches the edges of the array.

Operation mode sets the state of the micromouse; the robot performs different task depending on which operation it is on. The system has four mode operations. They are described in Table 3.

Table 3: Operation Mode description

Operation Mode	Description
1	Mapping the maze
2	Going back to the starting position
3	Quick run to the centre without searching
4	Completed!

In the beginning of the while loop, the program checks whether the current operation mode is 4; it terminates by being in an infinite loop which contains no task. Thus coming to a hold.

During modes 1 and 2, the micromouse checks for walls, updates the distance values and moves to the lowest neighbouring cell; this is done until the mouse reaches the goal. Then the operation mode is incremented. The difference between them is that the goal position is not the same. In operation mode 1, the goal position is the centre of the maze. On the other hand, in mode 2, the goal position is the starting point. In addition, during operation mode 1, after reaching the goal, the micromouse declares walls around the goal; due to there is only one entrance.

In operation mode 3, the micromouse updates the distance values only once, before starts moving. In order to determine the next movement, the cells which were not visited are ignored; that makes that the micromouse does not get lost, being in cells that it has never been. This also gives the mouse limited path and doesn't explore all the map. Before moving to the next cell, the micromouse also calculates how many cells it needs to move in the same direction. It allows the micromouse to reach a higher velocity depending on the number of cells.

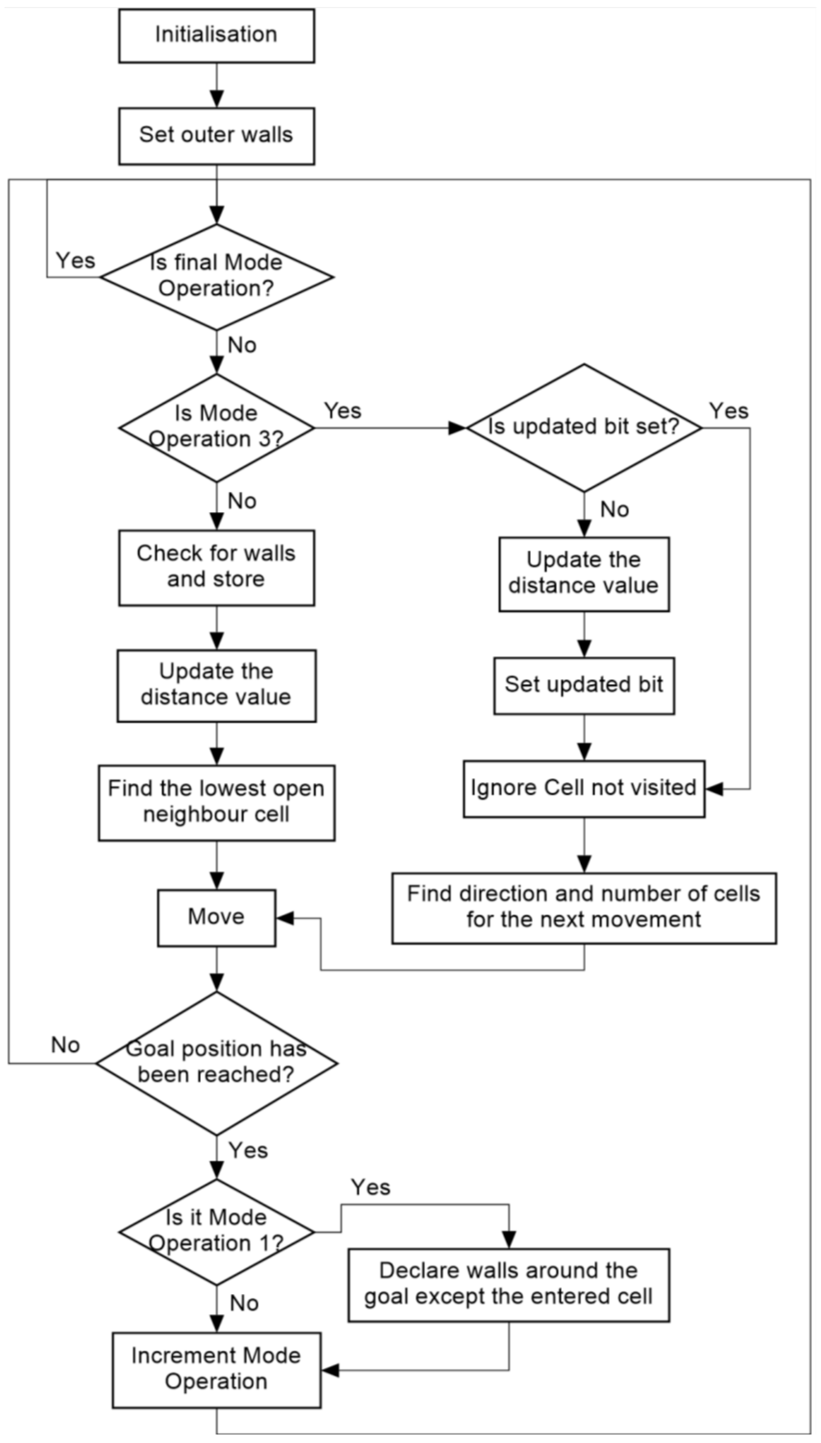

Initialisation
Set outer walls
Is final Mode Operation?
Yes
No
Is Mode Operation 3?
Yes
Is updated bit set?
Yes
No
Check for walls and store
Update the distance value
Update the distance value
Set updated bit
Find the lowest open neighbour cell
Ignore Cell not visited
Find direction and number of cells for the next movement
Move
No
Goal position has been reached?
Yes
Is it Mode Operation 1?
Yes
Declare walls around the goal except the entered cell
No
Increment Mode Operation

Figure 8: Flow Chart Diagram

Section 2

Sensor

The sensor information take an important role for the system. They are basically the eyes of the micromouse. Having these data, the micromouse should be able to maintain in a centre position in the cell, moving in a straight line and detect features of the maze, such as walls and corners.

In a range of possible distance sensors, the group chose to work with infrared sensors. The IR sensors are not accurate and the sunlight might cause interference; however, they are cheaper than other options and, for the micromouse application, their performance is enough.

Some research were realized in order to find how their technical specification should be. For the receiver, viewing angles are always wider than emitter [1]. Using the same reference, angle of an IR receiver should be between 30 and 60 degrees. In addition, an IR transmitter should emit light in a short range, under 10 degrees. Using those information, the IR receiver used for the micromouse project is the “IR T-1 Photodiode 34° SFH229FA”; the infrared transmitter is the “Infrared T-1¾ LED 850nm 6° SFH4550”.

The position of the sensors were decided based on research [1]. Six sensors were placed in the front of the micromouse: one sensor looking to each side, one sensor looking to each diagonal (approximately 45 degrees) and two sensors looking to the front. Initially, the sensors placed in the diagonals did not have any assigned purpose. However, after testing and programming the movements, they became fundamental for matching the virtual to the real position.

After designing and building a PCB board for the sensors, they needed to be aligned. The calibration was performed in two stages. Firstly, targets were drawn onto the walls in order to align all the transmitters. Side sensors should be emitting with a same angle. The same for diagonal sensors and front sensors. By using a camera the infrared could be seen and they could be matched with the targets. After that stage, the IR receivers were adjusted. They should be aligned in order to get the highest ADC values. The Figure 9 shows the infrared lights captured by the camera during the sensors’ alignment.

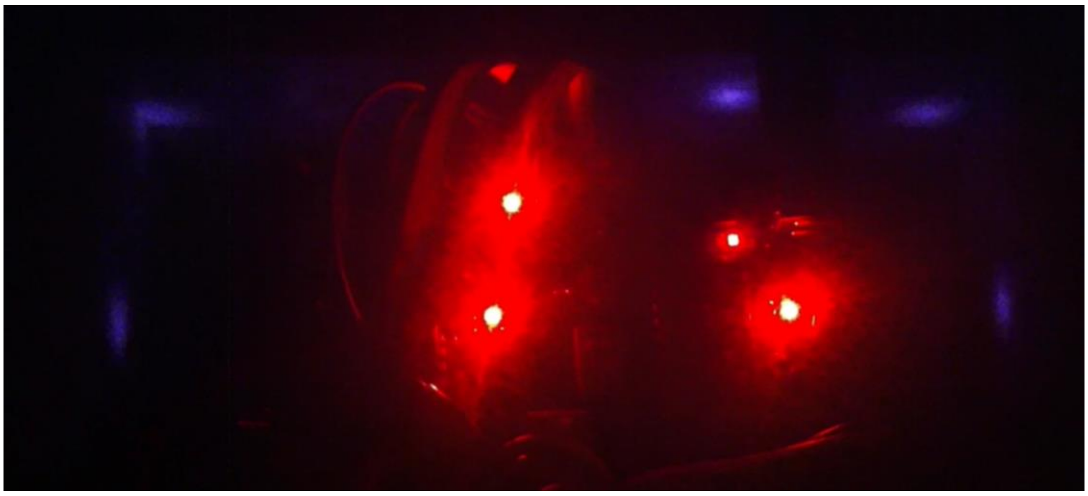


Figure 9: Aligning the sensors.

Finally, the resistors linked to the IR receivers needed to be selected. For that reason, an “ADC Value x Distance” graph for four different resistance values was built. The highest value in the graph is 810kΩ; as can be seen in Figure 10, the ADC Value reaches the saturation when the object is close to the sensor. For that reason, 810kΩ was not used. However, 706kΩ was chosen for the diagonal and front sensors. That value was chosen because it responds in far distances. The 560kΩ was chosen for the side sensors. That decision was made based on the sensibility in close distances. In addition, the side sensors do not need to reach more than 10-12cm.

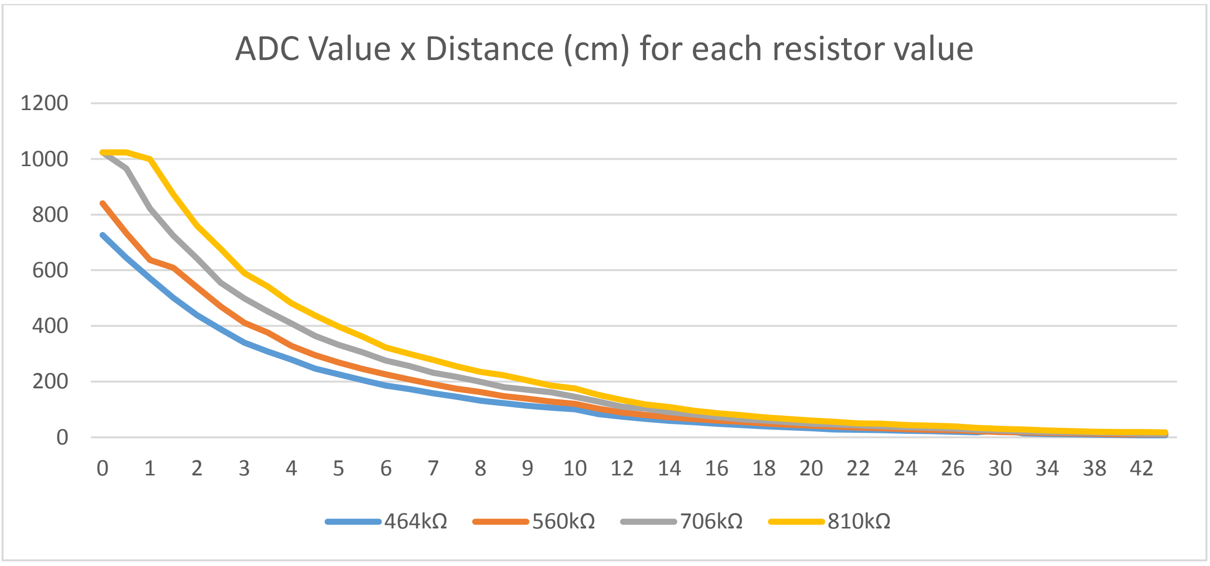


Figure 10: ADC Value x Distance for four different resistance values.

Control and Navigation

During the maze solving algorithm, a Cartesian coordinate system is used. Each combination of x and y integer values represents a different cell. In order to assure the robot is located where it should be, some procedures were taken; the virtual position needs to match with the real position.

In order to perform the movements, the main logic takes the number of encoder pulses into account. Firstly, some assumptions are made; the micromouse is moving in a straight line, in the centre of the cell and both motors are spinning at a same velocity. In that case, a specific number of pulses might indicates whether the robot has performed a turn or has moved a unit of cell. However, after testing, as the robot was not moving in a perfect straight line, the number of pulses could not perform the task for itself. Other logics were needed.

The first logic implemented is depicted in Figure 11. In this case, the micromouse identifies a corner in the maze and then redefine the amount of pulses needed to be counted in order to get to the centre of the next cell. Corners in both sides can be used to help in positioning. In addition, the front sensors can also be used as reference. As shown in Figure 12, when the micromouse faces a wall in front, the ideal distance to the front walls can be already known. The front walls should be also used to adjust the inclination of the micromouse, but that technique was not implemented; it did not seem necessary in order to perform properly.

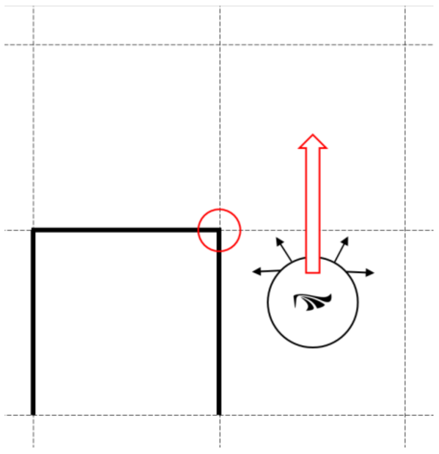


Figure 11: Position adjustment using corners.

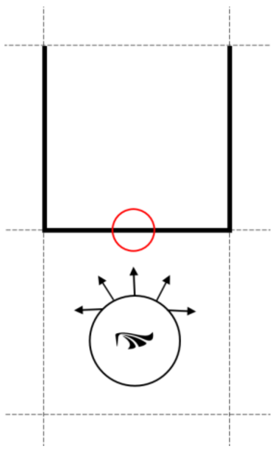


Figure 12: Position adjustment using front sensors.

In order to move in a straight line, the distances from the robot to the walls in the sides are used. Basically, the robot can face four different states: double walls (one wall each side), only a wall at the right side, only a wall at the left side, or no walls in both sides.

The micromouse always starts in the state which there are two walls (one each side). As can be seen in Figure 13, the sensors of both sides can be used in order to maintain the robot centralized. The difference between both sensors can be add to the desired velocity of both motors; then the robot will automatically correct your movement.

In the state which the there is only a wall at one side, the sensor data of the sensor related to that side is used in order to maintain the robot in the centre of the cell. The difference between the sensor data and the ideal distance is applied in the motors. As a consequence, the micromouse will start using only one wall as reference.

In the last case, when the micromouse does not have any wall to use as reference, the robot will basically move blinded. The system supposes that the robot is in central position; then, the micromouse will not apply any correction speed.

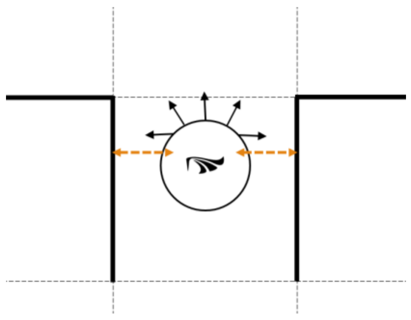


Figure 13: Maintain centralized in Double Wall state.

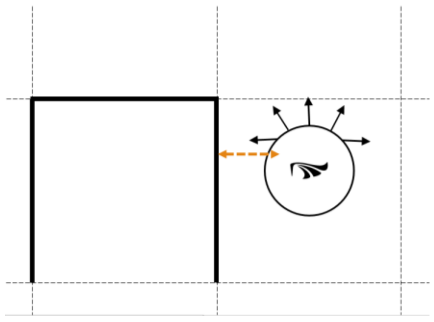


Figure 14: Maintain centralized in Left Side Wall state.

After defining how the micromouse would move, all the constants need to be determined. In order to find these values, a Bluetooth communication with the micromouse was stabilised. Using that tool, the constants could be modified on real-time. In addition, an interface could also show information about the system; as such as sensor values, speed and duty cycle value.

In Figure 16 and Figure 17, the Bluetooth ICARUS Interface is connected to the micromouse. The table shows the main constants. The column “Chr” indicates the character used to send that value; as a consequence, the micromouse will be able to identify where that value should be placed. The values are automatically sent to the micromouse when the user modifies in the table. The button “Save” saves all those values in the computer. In order to send all the current values to the micromouse, the button “Send all” was included. The button “Reset” loads all saved values.

In the panel “Move”, the user can decide when the micromouse should move, pause or stop. The “Stop” button resets all the maze and moving configuration. The “Pause” button just freezes the micromouse movement. Marking the “Mapping” option, the user can view the walls that the micromouse is storing, the position that the micromouse is virtually placed and the distance value of each cell. In Figure 15, the mapping was performed in a real maze. The micromouse is in operation mode 1; it is mapping the maze and searching for the centre.

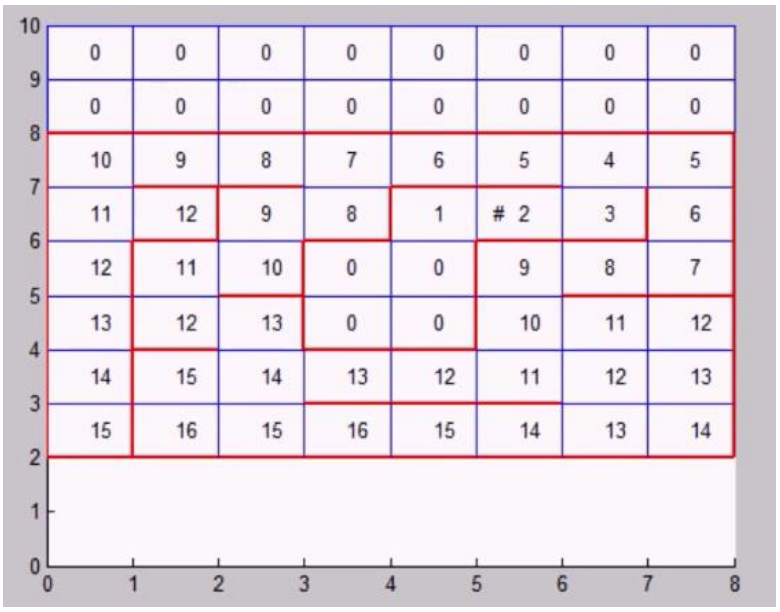
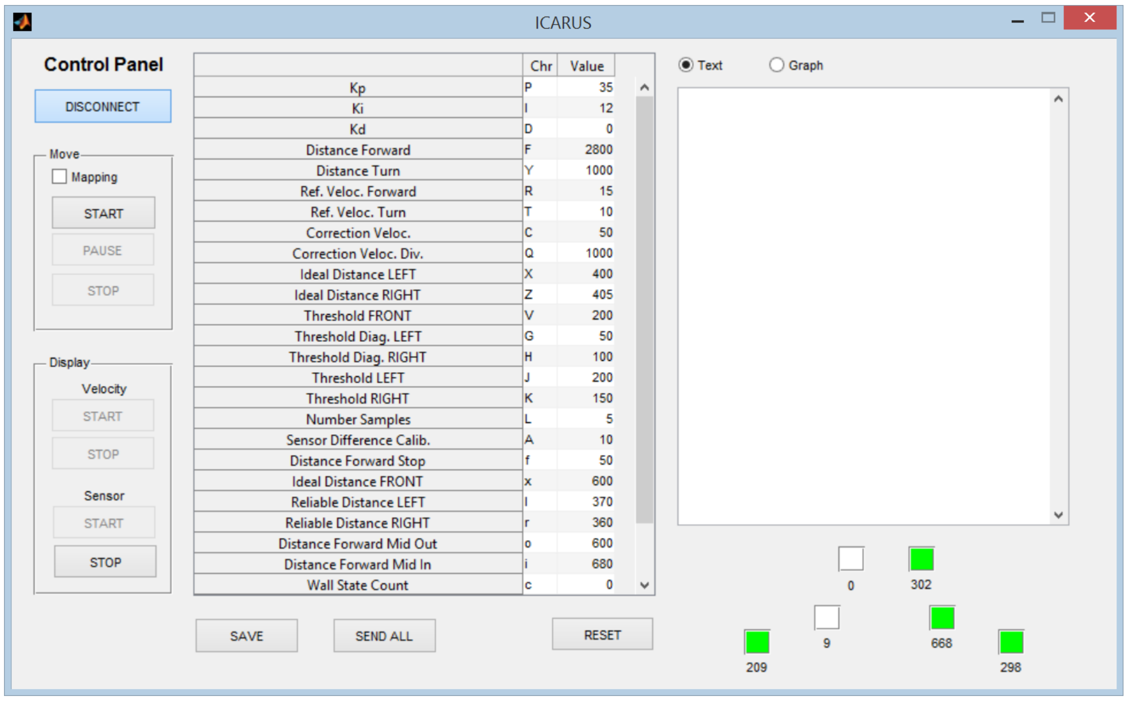


Figure 15: Visualizing the micromouse information on real-time.

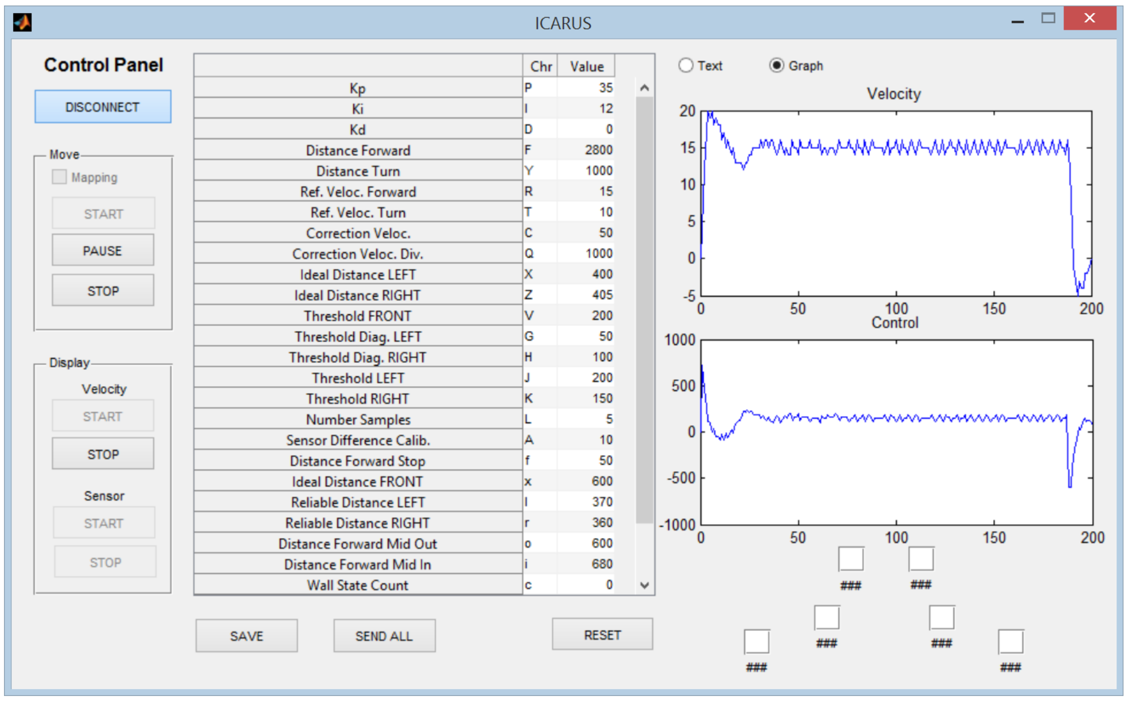
In Figure 16, the Icarus Interface displays all the current sensor values. Using that tool, ideal distances and thresholds values could be determined. In addition, the alignment of the IR receiver should be performed. As the interface shows the sensor values in real-time, sensor positions could be adjusted.



	Chr	Value
Kp	P	35
Ki	I	12
Kd	D	0
Distance Forward	F	2800
Distance Turn	Y	1000
Ref. Veloc. Forward	R	15
Ref. Veloc. Turn	T	10
Correction Veloc.	C	50
Correction Veloc. Div.	Q	1000
Ideal Distance LEFT	X	400
Ideal Distance RIGHT	Z	405
Threshold FRONT	V	200
Threshold Diag. LEFT	G	50
Threshold Diag. RIGHT	H	100
Threshold LEFT	J	200
Threshold RIGHT	K	150
Number Samples	L	5
Sensor Difference Calib.	A	10
Distance Forward Stop	f	50
Ideal Distance FRONT	x	600
Reliable Distance LEFT	l	370
Reliable Distance RIGHT	r	360
Distance Forward Mid Out	o	600
Distance Forward Mid In	i	680
Wall State Count	c	0

Figure 16: Icarus Interface displaying current sensor values.

In Figure 17, the program shows Velocity and PWM duty cycle (Control) in graph format. Using those graphs, the influence of each parameter in PID control could be analysed. When needed, the information could be seen in a text format. In this case, all the values are written in Text field.



	Chr	Value
Kp	P	35
Ki	I	12
Kd	D	0
Distance Forward	F	2800
Distance Turn	Y	1000
Ref. Veloc. Forward	R	15
Ref. Veloc. Turn	T	10
Correction Veloc.	C	50
Correction Veloc. Div.	Q	1000
Ideal Distance LEFT	X	400
Ideal Distance RIGHT	Z	405
Threshold FRONT	V	200
Threshold Diag. LEFT	G	50
Threshold Diag. RIGHT	H	100
Threshold LEFT	J	200
Threshold RIGHT	K	150
Number Samples	L	5
Sensor Difference Calib.	A	10
Distance Forward Stop	f	50
Ideal Distance FRONT	x	600
Reliable Distance LEFT	l	370
Reliable Distance RIGHT	r	360
Distance Forward Mid Out	o	600
Distance Forward Mid In	i	680
Wall State Count	c	0

Figure 17: Icarus Interface displaying Velocity and PWM duty cycle (Control) in graph format.

In order to tune the PID control, in the beginning, only the proportional parameter was taken into account. As much the value was increased, the velocity started to be close to the desired velocity. However, it never could reach the desired velocity. When the response started having overshoot, the integrational parameter was then used. The integrational parameter is able to nil the error in permanent state. The graph above shows the response of the system when Kp = 36 and Ki = 12.

Problems encountered

Since the beginning of the project, problems were faced. Initially, as the group did not have motors and boards, the maze solving algorithm was a good start. A problem came up; how to test and debug? The solution encountered was to use MATLAB in order to process the algorithm, display the maze and the robot. Then, the group wanted to start working in a prototype for testing. The idea was to use the Firecracker for testing. The prototype created is depicted in Figure 18.

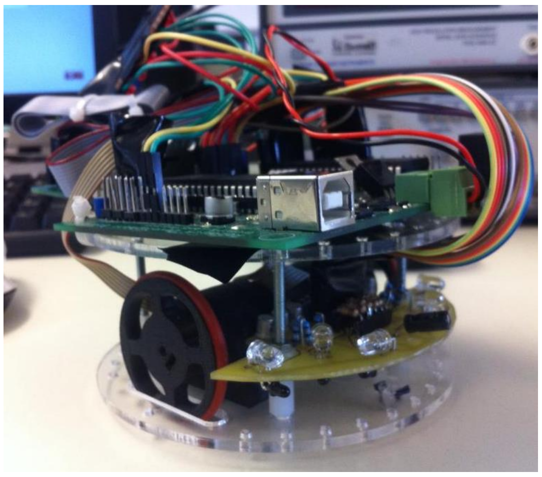


Figure 18: Prototype using Firecracker for testing.

During sensor testing, the group faced another unexpected problem: interferences between different sensors and IR light was being transmitted through the board. In order to solve the first problem, the code was changed in order to sample each sensor individually. In addition, an isolation tape was placed between the transmitter and the receiver. In that way, the IR light could not pass through the board.

After ordering the components, the group started mounting the final robot version (Icarus). During the assembling, many problems were faced. During the PCB design, the size of some connectors were not taken in account; as a consequence, during the mounting process, some of them needed to be cut. Others connectors needed to be removed therefore wires were soldered directly onto the board.

In the prototype showed in Figure 18, a sensor board was placed in front of the motors just below the Firecracker. Before building that prototype sensor board, two other boards were built using strip-boards, however, they did not work properly. A PCB board was built. Initially, this first PCB board was going to be used only for testing. However, as the group faced problems with the tracks in the final sensor board, the testing board was used in the final version (Icarus).

Conclusion

The micromouse project involved working with different skills in the robot sector. The group needed to work with sensor acquisition data, controlling motors, designing and building PCBs. As usual, the group had to deal with problems, such as sensor inaccuracy, sensor interferences, mechanical constrains and wrong components choices. All these facts were important for learning and avoid same problems in further projects.

In terms of working in group, the project is an opportunity to improve that skill. As the group had a great number of tasks to perform and deadlines were given, time management needed to be taken into account since the beginning. For that, Gantt chart was designed and tasks were divided between members. In order to check how each member was performing individual tasks, internal presentations and individual deadlines were extremely essential.

An important learning is about documenting every step or decision taken. The group has not written them down as desired. As a consequence, some thoughts and decisions were lost; there were no way to remember the reason why some decisions had been made.

Appendix

Technical drawings

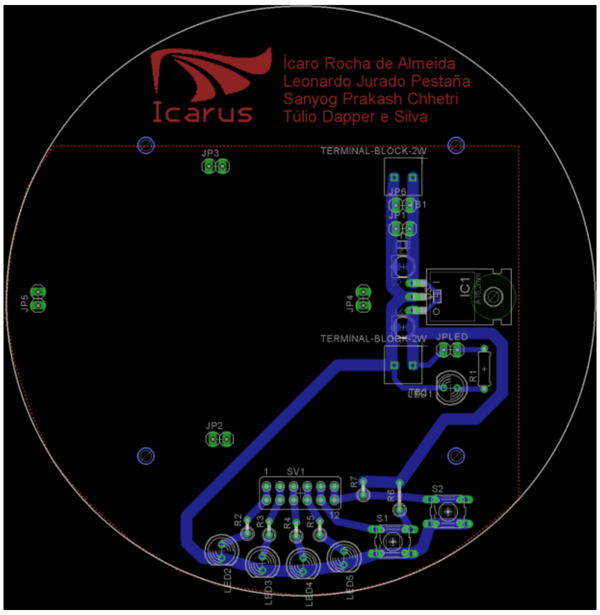


Figure 19: PCB Design – Battery Board.

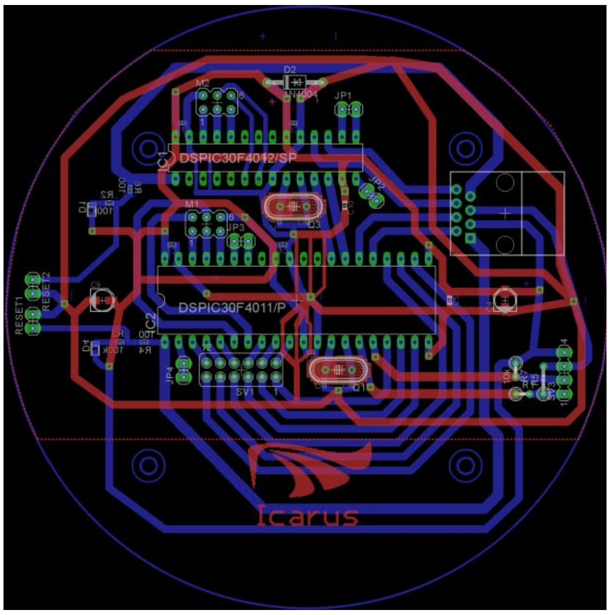


Figure 20: PCB Design – Main Board (PICs).

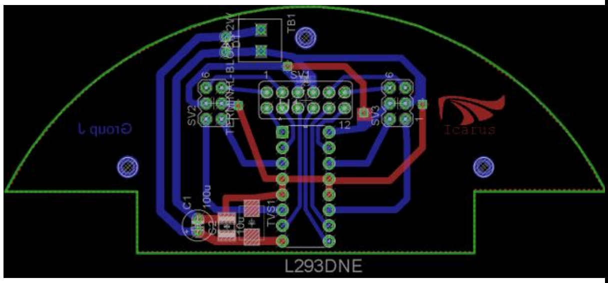


Figure 21: PCB Design – Motor Drive Board.

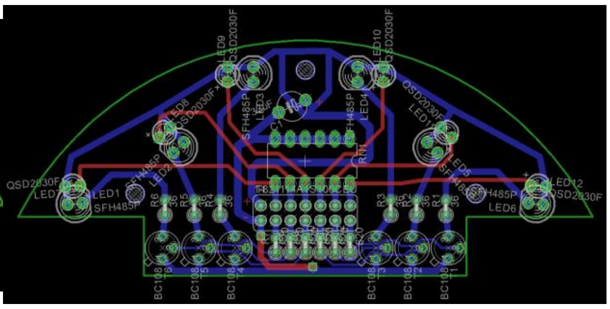


Figure 22: PCB Design – Sensor Board.

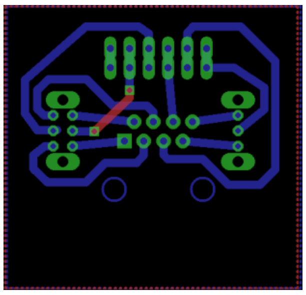


Figure 23: PCB Design – Board for burning the PICs.

Circuit Diagrams

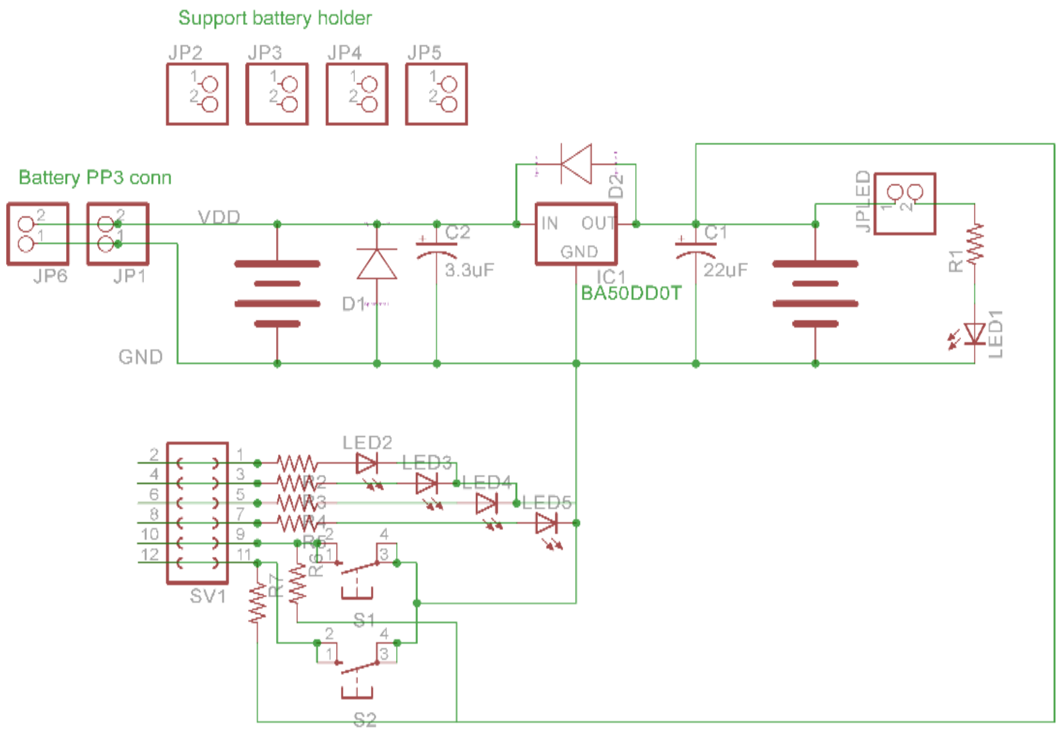


Figure 24: Schematic - Battery Board.

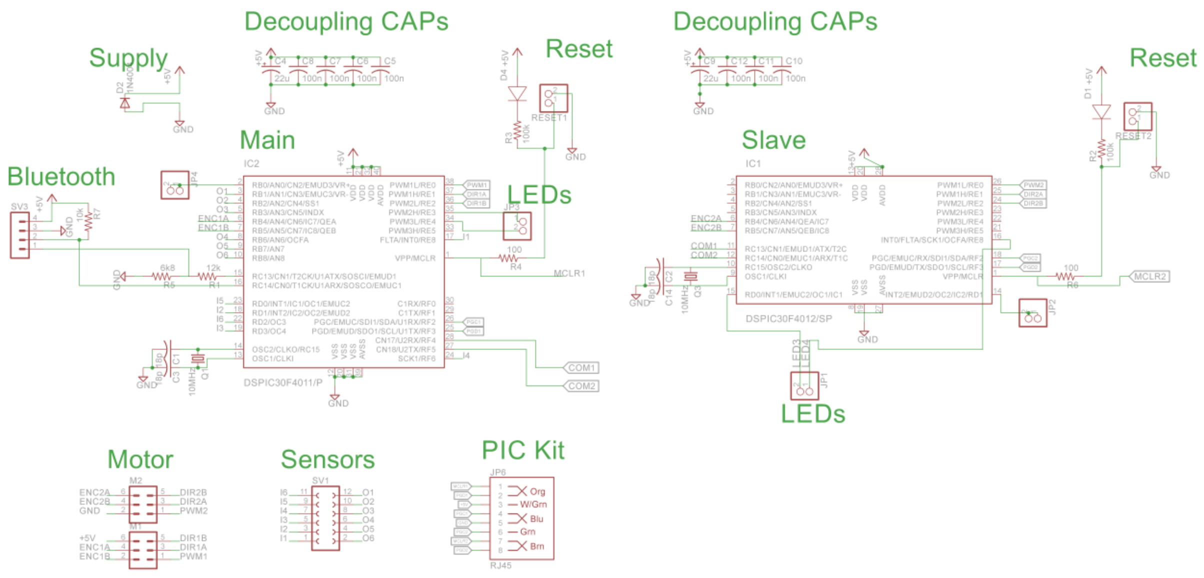


Figure 25: Schematic - Main Board (PICs).

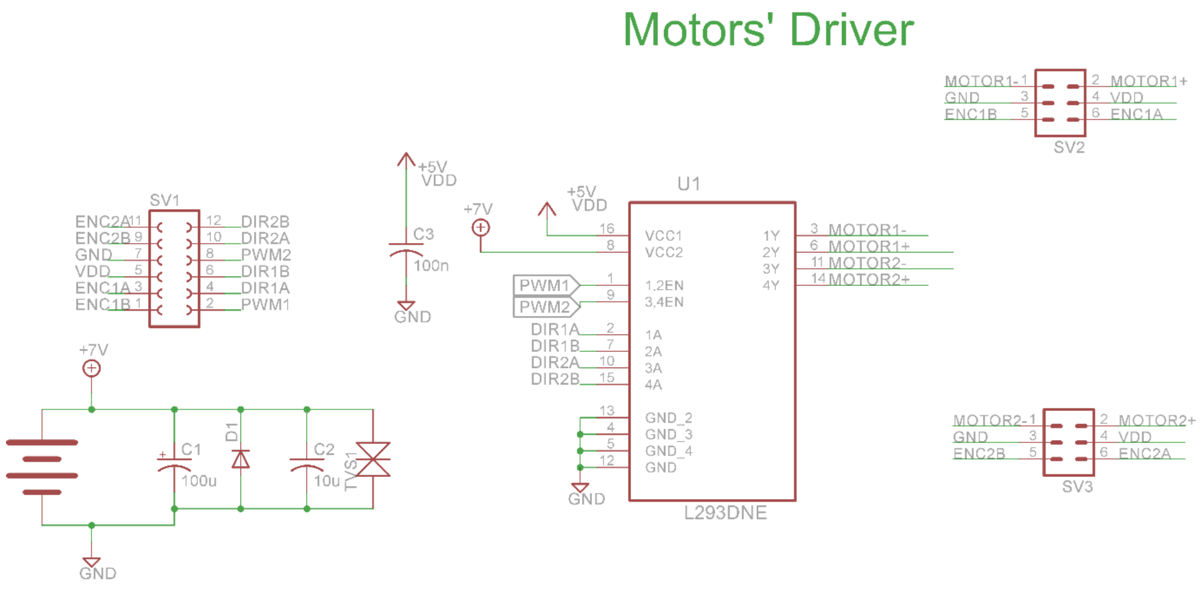


Figure 26: Schematic - Motor Drive Board.

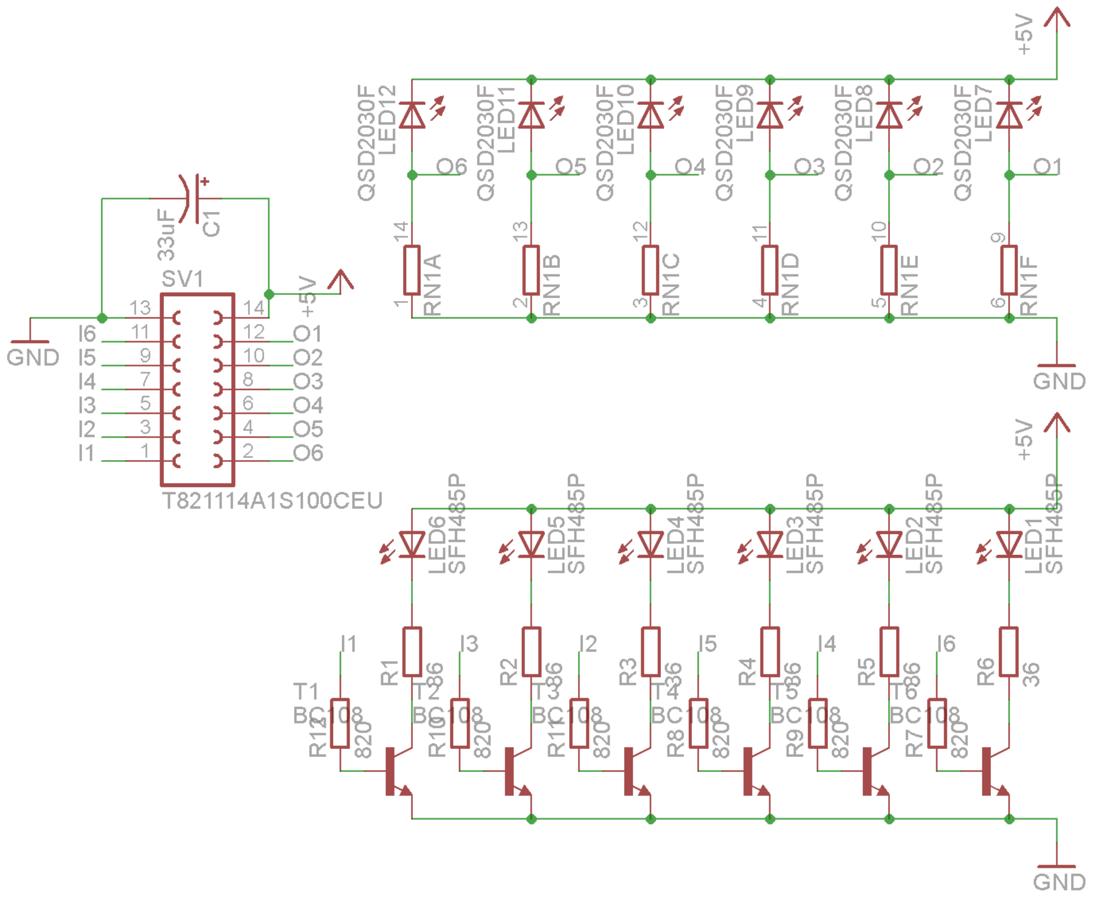


Figure 27: Schematic - Sensor Board.

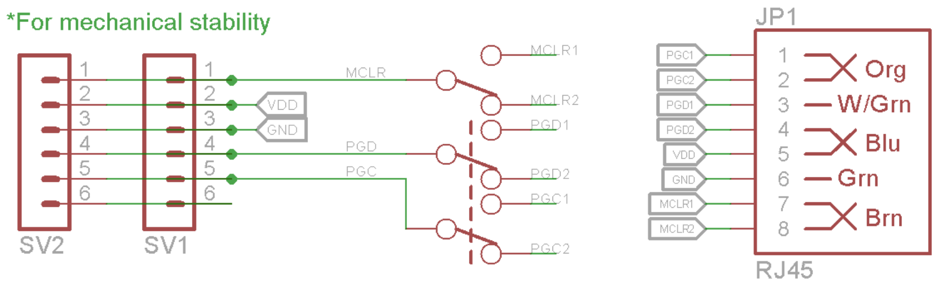


Figure 28: Schematic - Board for burning the PICs.

Parts List and Costing

Qty	Order Code	Description of Goods	Unit Cost	Net Cost
3	1841939	XTAL, 10.000MHZ, 18PF	0.697	£2.09
10	1539447	ALU ELEC, 22UF, 6.3V, SMD	0.16	£1.60
3	1973360	ALU ELEC, 3.3UF, 50V, SMD	0.073	£0.22
1	3829571	BATTERY HOLDER, 6XAA	3.74	£3.74
4	1165527	CAP, ALU ELEC, 22UF, 35V, RAD	0.3	£1.20
10	1759056	CAP, MLCC, C0G/NP0, 18PF, 50V, 0603	0.0083	£0.08
16	1759122	CAP, MLCC, X7R, 100nF, 50V	0.0083	£0.13
2	1327750	ceramic smd cap 10uF	0.603	£1.21
3	1611856	CRYSTAL, 10MHZ, 18PF, HC49/S	0.252	£0.76
4	2335181	diode 1n4004	0.0947	£0.38
5	2341897	DIODE, SCHOTTKY, 0.1A, 40V, 0603	0.0645	£0.32
3	2341897	DIODE, SCHOTTKY, 0.1A, 40V, 0603	0.0645	£0.19
5	1017843	DIODE, TVS, 9V, 600W, SMB	0.409	£2.05
10	1593452	HEADER, 2 ROW, R/ANGLE, 12WAY	0.149	£1.49
10	1022238	HEADER, 2ROW, 12WAY	0.225	£2.25
2	1513592	HEADER, C-GRID III, VERT, 14WAY, 2ROW	0.54	£1.08
2	2215306	HEADER, VERTICAL, 2.54MM, 12WAY	0.513	£1.03
3	2215307	HEADER, VERTICAL, 2.54MM, 14WAY	0.472	£1.42
2	2215302	HEADER, VERTICAL, 2.54MM, 6WAY	0.368	£0.74
1	4620902	HEAT SINK, DIP, GLUE-ON, 46°C/W	0.428	£0.43
2	4302163	HEAT SINK, FOR SMD, 123°C/W	0.343	£0.69
3	2215249	IDC, S/RELIEF, 2.54MM, 14WAY	0.519	£1.56
1	1331561	JACK, SMT, R/A, RJ45, 8P8C	0.787	£0.79
1	1524544	JACK, THT, R/A, RJ45, 8P8C	0.667	£0.67
7	1573495	LED, IR, 5MM, 850NM	0.329	£2.30
1	1524544	MODULAR, JACK, THT, R/A, RJ45, 8P8C	0.667	£0.67
1	1668351	RECEPTACLE, 2.54MM, DUAL, 12WAY	1.33	£1.33
2	1668358	RECEPTACLE, 2.54MM, DUAL, 14WAY	1.56	£3.12
4	2335725	red led	0.0643	£0.26
10	2330119	RESISTOR, METAL , 36R, 0.6W, 1%	0.0279	£0.28
10	2331713	RESISTOR, POWER, 100R, 0.2W, 5%, 0603	0.0062	£0.06
5	2309107	RESISTOR, THICK FILM, 100KOHM, 100MW, 1%	0.002	£0.01
2	2215248	SOCKET, IDC, S/RELIEF, 2.54MM, 12WAY	0.525	£1.05
8	1466845	SPACER, M3X12-NI	0.32	£2.56
6	1466796	SPACER, M3X25-NI	0.355	£2.13
1	2435095	SWITCH SLIDE DPDT	0.42	£0.42
1	2435104	SWITCH SLIDE SPDT	0.33	£0.33
2	2396053	SWITCH, TACTILE, 50MA, 12VDC, SMD	0.173	£0.35
2	3882615	TERMINAL BLOCK, WIRE TO BRD, 2POS	0.281	£0.56
			Net Total:	£41.52
			VAT:	£8.30
			Gross Total:	£49.82

References

[1] Green, Y. (2015). Lecture 5: Parts/design Choices [online]. Available from: http://micromouseusa.com/?page_id=1342 [Accessed 12 February 2015].

[2] Harsh, V. Micromouse Project [online]. PostDoctoral, University of Southern California. Available from: robotics.usc.edu/~harsh/docs/micromouse.pdf [Accessed 03 February 2015].

[3] Kubica, J. (2003). Maze Toolbox [online]. Available from: http://www.cs.cmu.edu/~jkubica/code/matlabMaze.html [Accessed 05 November 2014].

[4] Microchip (2005) dsPIC30F Family Overview [online]. Available from: http://ww1.microchip.com/downloads/en/devicedoc/70043F.pdf [Accessed 28 October 2014].