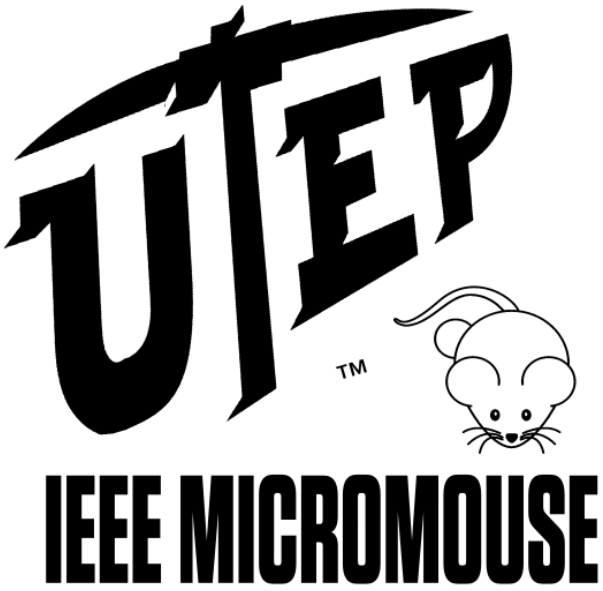


Workshop Week 4:

Serial console and code refactoring

Objective:

Use the serial console for wireless and wired communication with an external device. Refactor code using separate compilation units and header files.

Background information:

Writing and testing code can be hard, especially when you cannot tell what is happening inside of the computer. On previous weeks, you might have noticed that when writing code, sometimes you want to know what the value of a variable is or if a branch is being taken or not. With Arduino, there is no way to know this directly since there are no displays (or debuggers) involved. Most microcontroller evaluation boards ship with an onboard debugger to start, stop and step through your code, Unfortunaly, the Arduino Nano is not one of those boards. So, what can we do about it? Well, we can use the serial port on the Arduino Nano, more commonly known as the Universal Asynchronous Receiver and Transmitter (UART). This peripheral device is built inside the Arduino chip, and it will help us transmit data to an external device with a few lines of code. This will allow us send data to test and debug our code on the micromouse. Furthermore, we will be able to print out data for the user to see the maze and the mouse position within the maze.

UART is a type of serial communication that can be used to communicate with devices that support it. As the name implies, serial communication works by sending data bit by bit in series, or one right after another. The Arduino handles all internal functionality of the UART through a set of functions. We are interested in using the UART to communicate with a computer through the USB cable and to the same computer (or another device) using the HC-05 Bluetooth as a serial bridge. For this workshop we will use the HC-05 to send useful data to a computer or device that has Bluetooth support.

Coming back to the complexity of writing code, you might be starting to realize that your code is becoming a bit unwieldy. There is just too much to keep track on one single file, and it is difficult to see which lines of code are doing what. To remediate this issue, we will start refactoring some of our code and add a Hardware Abstraction Layer (HAL). The HAL will help us abstract low level hardware functionality from some other parts of our code like the maze, path finding algorithm and mouse pose.

Serial communications with Arduino Nano

To get the serial communication started we need to write this line of code on the setup function of the Arduino sketch

```
1.  #include "micromouse.h"
2.
3.  void setup()
4.  {
5.    Serial.begin(9600);
6.  }
7.
8.  void loop()
9.  {
10. }
11.
```

The function

```
1.  Serial.begin(9600)
```

Starts the UART with a baud rate (or bits per second) of 9600. The number entered in the argument determines the speed in which the data is sent to the device. This number has a theorical limit, but for now we are just going to use a low speed of 9600 bits per second, which is standard for a great deal of devices.

After we setup the UART, we can then start sending data in the form of text, integers, fractions, and other types. To test this let's send some values to the serial terminal.

```
1.  #include "micromouse.h"
2.
3.  void setup()
4.  {
5.    Serial.begin(9600);
6.  }
7.
8.  void loop()
9.  {
10.   int integer = 10;
11.   float pi = 3.1415
12.   char text[] = "hello from micromouse";
13.   Serial.println(integer);
14.   Serial.prinln(pi);
15.   Serial.prinln(text);
16. }
```

The function

```
1.  Serial.println();
```

Will print the value inside its argument with a new line after the value has been printed. This is in contrast with the function

```
1.  Serial.print()
```

Which will only print the value WITHOUT a new line added at the end

To visualize the data being print out, open the serial console on the Arduino and select your Arduino's COM (check the setup Arduino for micromouse file) port and baud rate, which for this example it will be 9600.

Figure 1 The Arduino serial console will be on the top right corner of the window

Once you open this window and you have selected the right parameters, you should the text:

10
3.1415
Hello from micromouse

Being print out constantly in this console.

Ok, so let's use this new idea to send useful data back to a computer. On last week workshop we worked with moving the mouse 180mm forward. Let's use the serial console to see how accurate the mouse is in reaching this position and while we are it, let's print a greeting message and a button pressed notification message.

```
1.  #include "micromouse.h"
2.  #include <Encoder.h>
3.
4.  Encoder right_encoder(ENCODER_0_CHA, ENCODER_0_CHB);
5.  Encoder left_encoder(ENCODER_1_CHA, ENCODER_1_CHB);
6.
7.  const int speed = 64;
8.
9.  void setup()
10. {
11.   /*Setup motor pins*/
12.   pinMode(ENABLE, OUTPUT);
13.   pinMode(MOTOR_0_1A, OUTPUT);
14.   pinMode(MOTOR_0_2A, OUTPUT);
15.   pinMode(MOTOR_1_1A, OUTPUT);
16.   pinMode(MOTOR_1_2A, OUTPUT);
17.
18.   pinMode(BUTTON, INPUT);
19.   Serial.begin(9600);
20.   Serial.println("Micromouse Workshop Week 4. Hello Student!")
21.   Serial.println("-----------------------------------------")
22. }
23.
24. void loop()
25. {
26.   while(digitalRead(BUTTON) != HIGH);
27.   Serial.println("Button has been pressed");
28.   delay(1000);
29.
30.
31.   /*Move motor forward at given speed*/
32.   analogWrite(MOTOR_0_1A, 0);
33.   analogWrite(MOTOR_0_2A, speed);
34.
35.   analogWrite(MOTOR_1_1A, speed);
36.   analogWrite(MOTOR_1_2A, 0);
37.
38.   long int right_encoder_count;
39.   long int left_encoder_count;
40.   long int average_encoder_count = 0;
41.
42.   while(average_encoder_count < 1090)
43.   {
44.     right_encoder_count = right_encoder.read();
45.     left_encoder_count = -left_encoder.read();
46.
47.     average_encoder_count = (right_encoder_count + left_encoder_count)/2;
48.     delay(20);
49.   }
50.
51.   Serial.print(right_encoder_count);
52.   Serial.print(", ");
53.   Serial.print(left_encoder_count);
54.   Serial.print(", ");
55.   Serial.println(average_encoder_count);
56.
57.   /*Stop motors*/
```

```
58.   analogWrite(MOTOR_0_1A, 0);
59.   analogWrite(MOTOR_0_2A, 0);
60.
61.   analogWrite(MOTOR_1_1A, 0);
62.   analogWrite(MOTOR_1_2A, 0);
63.
64.   right_encoder.write(0);
65.   left_encoder.write(0);
66. }
67.
```

While you can test the previous code using the USB cable, this will become inconvenient when the mouse is moving. To communicate with the mouse wirelessly, connect your HC-05 module to the micromouse and flip the switch on the right side to the ON position.

CAREFUL: You will not be able to program the Arduino Nano while you are using the HC-05 module, you must flip the right-side switch back to the OFF position when you want to load a new program.

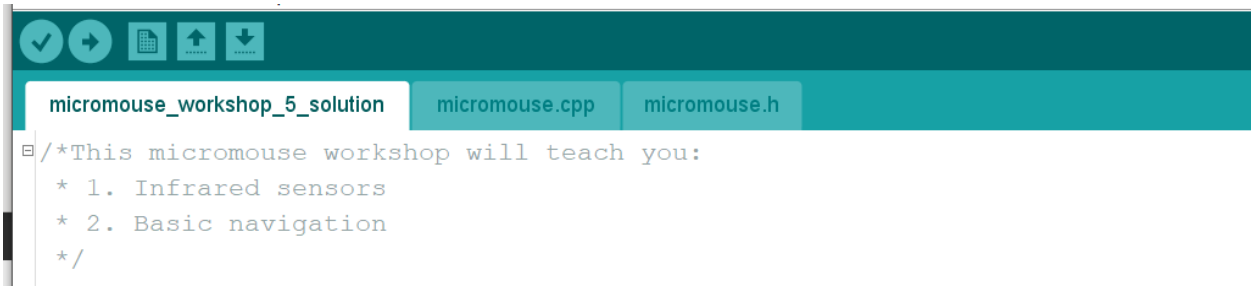
Figure 2 HC-05 Bluetooth module

Code refactoring

From the previous example, you can notice that the code is becoming messy and unwieldy. We must do something about this before we tackle more complex concepts like maze, path planning and mouse pose.

Like it was mentioned on the background information section, we are going to develop a Hardware Abstraction Layer (HAL) that will handle the low-level hardware (Motors, gyroscope, button, infrared sensors). To start this HAL, add a new file into your sketch folder with the name *micromouse.cpp*

If you do not see this file on your sketch, restart the Arduino IDE and open the same sketch again. After you do this, you should see this on your sketch.



On the new *micromouse.c* file write the following function

```
1.  #include "micromouse.h"
2.
3.  void motor_0_speed(int speed, bool direction)
4.  {
5.    if(direction == true)
6.    {
7.      /*Move motor forward at given speed*/
8.      analogWrite(MOTOR_0_1A, 0);
9.      analogWrite(MOTOR_0_2A, speed);
10.   }
11.   else
12.   {
13.     analogWrite(MOTOR_0_1A, speed);
14.     analogWrite(MOTOR_0_2A, 0);
15.   }
16. }
17.
```

Then on the *micromouse.h* add a prototype function for this newly created *motor_0_speed* function.

Your *micromouse.h* file should look like this

```
1.  #ifndef MICROMOUSE_H_
2.  #define MICROMOUSE_H_
3.
4.  #include <stdbool.h>
5.  #include <Arduino.h>
6.
7.  /*Motor definitions*/
8.  #define ENABLE            A6
9.  #define MOTOR_0_1A        11
10. #define MOTOR_0_2A        5
11. #define MOTOR_1_1A        6
12. #define MOTOR_1_2A        9
13. #define ENCODER_0_CHA     2
14. #define ENCODER_0_CHB     13
15. #define ENCODER_1_CHA     3
16. #define ENCODER_1_CHB     8
17.
18. /*Infrared sensors definitions*/
19. #define IR0_TX       A1
20. #define IR1_TX       12
21. #define IR2_TX       7
22. #define IR3_TX       4
23. #define IR0_RX       A0
24. #define IR1_RX       A7
25. #define IR2_RX       A2
26. #define IR3_RX       A3
27.
28. /*Extras*/
29. #define BUTTON      10
30.
31.
32. void motor_0_speed(int speed, bool direction);
33.
34. #endif
35.
```

Using this code you can now set the speed and direction of the motor on your sketch by simply calling this function like this

```
1.  const int speed = 64;
2.  void setup()
3.  {
4.    /*Setup motor pins*/
5.    pinMode(ENABLE, OUTPUT);
6.    pinMode(MOTOR_0_1A, OUTPUT);
7.    pinMode(MOTOR_0_2A, OUTPUT);
8.    pinMode(MOTOR_1_1A, OUTPUT);
9.    pinMode(MOTOR_1_2A, OUTPUT);
10.
11.   pinMode(BUTTON, INPUT);
12. }
13.
14. void loop()
15. {
16. /*Move motor forward at given speed. A true value will move the motor forward, false
    backwards*/
17.   motor_0_speed(speed, true);
18. }
19.
```

Exercise:

Refactor the rest of the code to initialize the motor pins, set the speed of motor 1 and wait for the button to be pressed. You prototype functions could look like this:

```
1.  void motors_init(void);
2.  void wait_button(void);
3.  void motor_1_speed(int speed, bool direction);
4.
```