Lab 1 — An Introduction to 3DoT & C++

This lab is designed to introduce you to the 3DoT Board (and its equivalent, the Sparkfun Pro Micro board), the Arduino Integrated Development Environment (IDE) and C++ programming language. Plus, you will learn about the power of library files. Library files are simply files that you instruct the Arduino IDE to include in your program. In this lab, you are going to create a library file called 3DoTConfig that handles all of the initialization and configuration for your robot.

You can find this and future lab updates at http://www.csulb.edu/~hill

Table of Contents

What is New?

New terminology and concepts are listed below in black. If you have any questions on this information, refer back to the lectures on the introduction to C++ and configuring the GPIO registers.

Data Types

```
const        // Define variable as a constant
static       // Define variable that will persist after a function return
uint8_t      // Defines the data type as an unsigned 8 bit integer (0-255)
uint16_t     // Defines the data type as an unsigned 16 bit integer (0-65,535)
```

C++ Code

```
#define Name Number // Allows the user to define Name to be equivalent to Number
#include <filename> // Used to include any built-in libraries such as avr/io.h
#include “filename” // User defined libraries or files that are in the sketch folder
```

Bit / Byte Operations

```
variable = byteValue << numberOfShifts; // Left shift byte value by the number defined
variable = _BV (bitNumber);             // Turns a bit number into a byte value
variable |= _BV(bitNumber);             // Set a specific bit in a byte value
variable &= ~_BV(bitNumber);            // Clear a specific bit in a byte value
variable = byteValue | byte Value;      //
```

Arudino Built-In Functions

Pin Configuration & Control

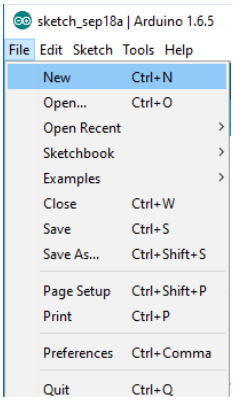
```
pinMode(pin_number, TYPE);        // Define a pin as an input or output
digitalRead(pin_number);          // Read a digital value for an input pin
analogRead(pin_number);           // Read an analog value from an input pin
digitalWrite(pin_number, output); // Output a digital value to an output pin
```

Introduction to the Arduino IDE

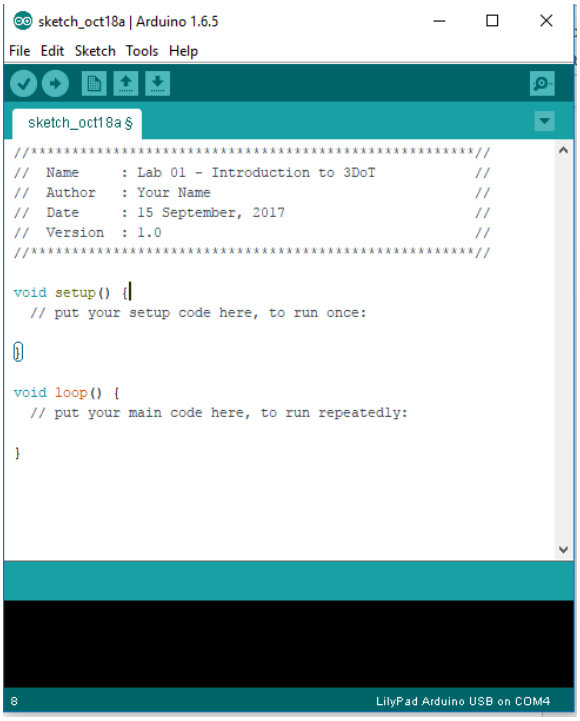
In lab, you will be spending most of your time working within an Integrated Development Environment (IDE). For our labs, we will be working in the Arduino IDE. The IDE lets us write our program in a human readable form (C++) and then translate it into a machine-readable form understood by the ATmega32U4. There are many other types of IDEs for C++ programming such as Eclipse and Atmel Studio 7 that allow us to create very complex projects. The focus will be on the Arduino IDE because of how easy it is to learn and use.

Creating A New Sketch In Arduino IDE

To start our lab, we will be creating a new sketch in the Arduino IDE. This is done by opening the Arduino IDE and it should generate a blank sketch to start with. The other way is to click File -> New as shown in the figure below.



The first thing we will be adding into our code is a title block to describe the purpose of the sketch and the author. This will go before the setup function and should look something like this.



Next, you will need to rename the sketch to a more descriptive name. Click on File -> Save and type in the name you want to save it as such as Lab01. It will create a folder for your sketch in the location you have designated for all sketches (Default is in the Arduino folder located in My Documents). Now we will take a look at the hardware before developing the code to control it.

3DoT Board Schematic & Block Diagram

The end goal of these labs is to program a robot utilizing the 3DoT board or the equivalent setup using the Sparkfun Pro Micro, TB6612FNG Motor Driver, and other components. Shown below are the major features of the 3DoT board and the block diagram of the latest version.

3DoT is a micro-footprint 3.5 x 7 cm all-in-one Arduino compatible microcontroller board designed for robot projects.

3DoT Board

- ATmega32U4 Microcontroller Unit (MCU)
- CSULB IR Sensor Shield
- Power from a single CR123A 650mAh rechargeable Li-ion battery
- Integrated 3.7v Li-ion battery charger
- All digital logic powered from Low Dropout (LDO) 3.3v regulator with power and ground output header pins provided.
- Battery Level Sensor
- TB6612FNG Dual DC Motor Driver
- 3.7v, 5.0v (default), and 6.0v Turbo Boost for driving DC motors
- Reverse voltage and overvoltage protection circuitry
- Current protection provided by a PTC polyfuse.
- Android and Apple iOS application software (HM-11 Bluetooth BLE module required)
- Two I2C compatible connectors
- Connectors for two 3.7v Micro or Ultra-Micro Servos

It would be ideal to spend some time analyzing the block diagram to get a better understanding of the capabilities of the 3DoT board but we will focus on the connections between the motor driver and IR sensors for Lab 1. There is a specific way that the pins are mapped to the motor driver and sensors, which is what you will want to follow if you are using the Sparkfun Pro Micro. Figure 2 provides an isolated view of those connections.

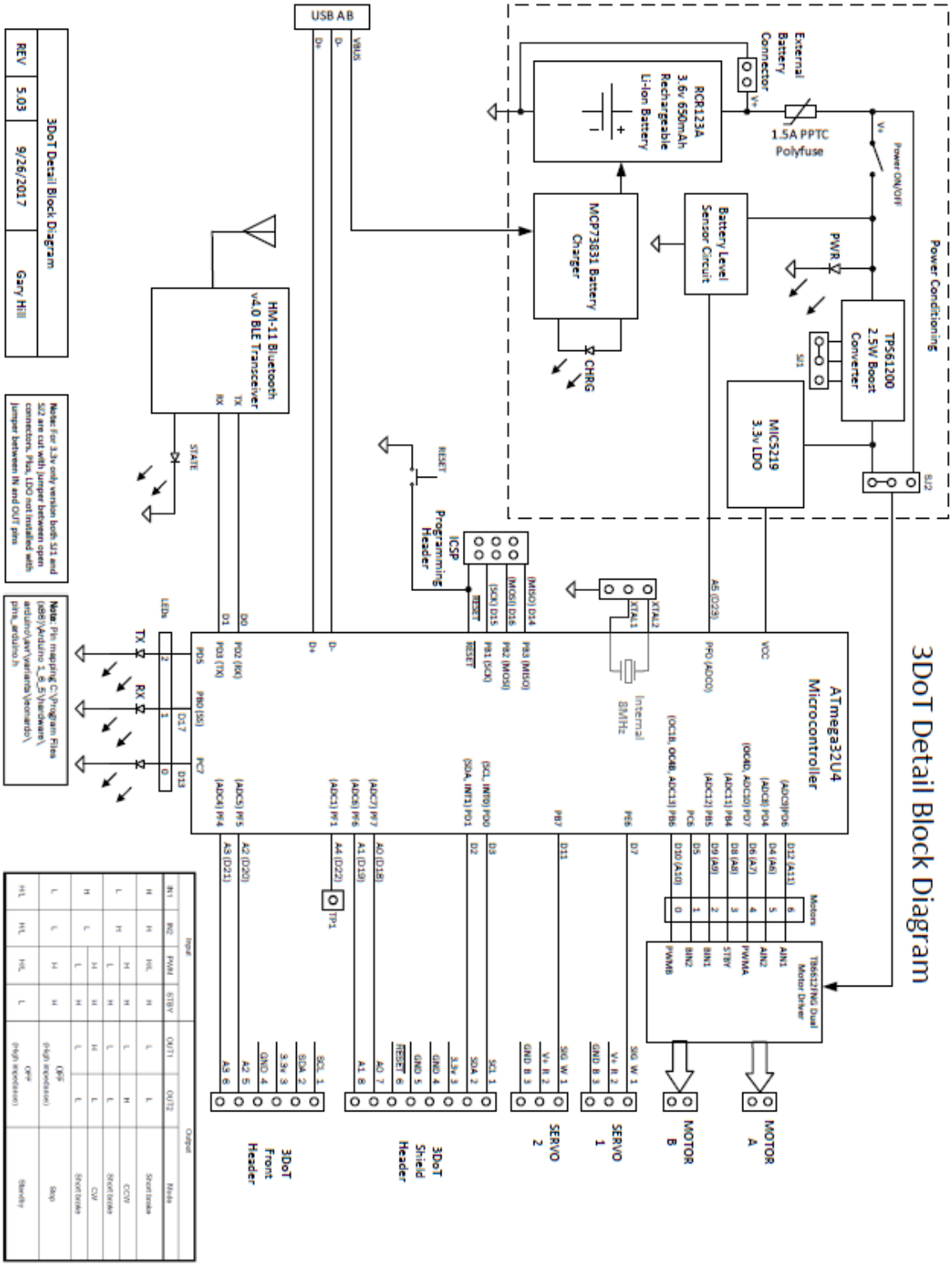


Figure 1 – 3DoT Detailed Block Diagram

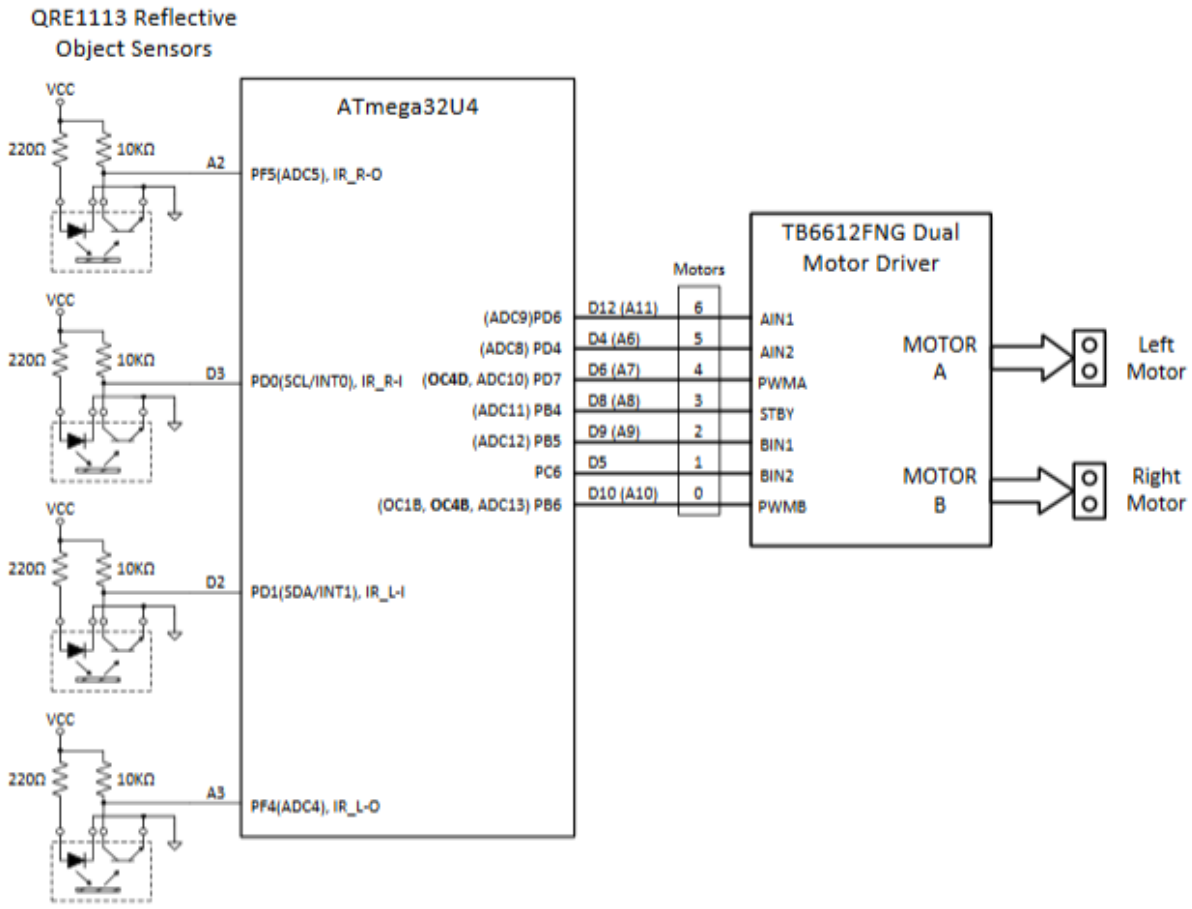


Figure 2 – Optical Sensor to Motor Driver Interface

The main objective of Lab 1 is to develop the line following logic for your robot to move through the maze that was shown in the Mission PDF. We will handle what to do at intersections in later labs. For the line following algorithm, it will involve taking data in from the four IR sensors and deciding how the control signals being sent to the motor driver needs to change based on that information. The inner two IR sensors provide the most relevant data about your robot’s position as it follows the line and is what we will be focused on. You can utilize all four sensors if you want to. To start developing this code, you will need to configure the input and output pins.

Configuring the Input and Output Pins

The first thing that needs to be done in your program is to define what pins are going to be used for the Atmega32U4. Just because components have been hardwired to specific pins does not mean the microcontroller knows what they are being used for. You will need to configure the input and output pins. As you learned in lecture, this involves modifying the GPIO registers (DDRX and PORTX).

Working with the GPIO Registers

In order to configure a pin as an input, the corresponding bit position in the DDRX register needs to be cleared to 0. If that input pin needs to use a pull-up resistor, the corresponding bit position in the PORTX register needs to be set to 1. Obtaining the value that is being detected on that input is done by reading

the corresponding bit position in the PINX register. For an output pin, the value in the DDRX register needs to be set to 1. You have to write the value that is to be outputted in the PORTX register. This information is also provided in figure 3.

SIMPLIFIED VERSION (PUD = 0)

DDRxn	PORTxn	I/O	Pull-up	Comment
0	0	Input	No	
0	1	Input	Yes	
1	0	Output		Output Low (Sink)
1	1	Output		Output High (Source)

Figure 3 – GPIO Pin Configuration

From figure , you will notice that our input pins for the IR sensors are PD0, PD1, PF4, and PF5. The output pins that go to the TB6612FNG motor driver are PB4, PB5, PB6, PC6, PD4, PD6, and PD7. It is quickly apparent that some of the GPIO registers such as DDRD will have both inputs and outputs defined. It is possible to define each pin individually but it would be more efficient to develop an expression to get the final value that needs to be written to the register. It is also important to preserve the values of the other bits in the GPIO register because that would change the functionality of those pins. This can be done with a combination of the bit/byte operations that were introduce earlier.

We will use the output pins PB4, PB5, and PB6 as our first example. Because they are outputs, bit positions 4, 5, and 6 of DDRB will need to be set to 1. If you were to configure them individually, you would the following instruction.

```
DDRB |= (1<<PB4);
```

That instruction takes the original value of the DDRB register and performs the OR operation with the value provided. This will set only bit position 4 and define that pin as an output. That value can be a single expression or multiple expressions combined together, which means the instruction could be expanded to this.

```
DDRB |= ((1<<PB4) | (1<<PB5) | (1<<PB6));
```

The expression (1<<PB4) translates into a binary value of 0b00010000 because of how the << operator works and is equivalent to _BV(PB4). You can use either when defining the values that need to be put into the GPIO registers.

For clearing values, the following instruction would be used. It is performing the AND operation on the original value of DDRD and the complement of the expression.

```
DDRD &= ~(1<<PD0);
```

On Your Own

Apply what you have learned to write the code needed to configure the input and output pins for your robot.

Using the Arduino Built-In Functions

Now that you have a better understanding of the configuration of the pins, we can utilize the built-in functions that the Arduino IDE provides to simplify the code. These functions are the `pinMode()`, `digitalRead()`, `analogRead()`, and `digitalWrite()` functions. They essentially operate the same way as manually working on the GPIO registers that we covered earlier. The main difference is that it uses the pin numbering that is specific to the board being used. In this case, that would be the Sparkfun Pro Micro. Figure 4 shows the pin numbering and how they correspond to the GPIO ports.

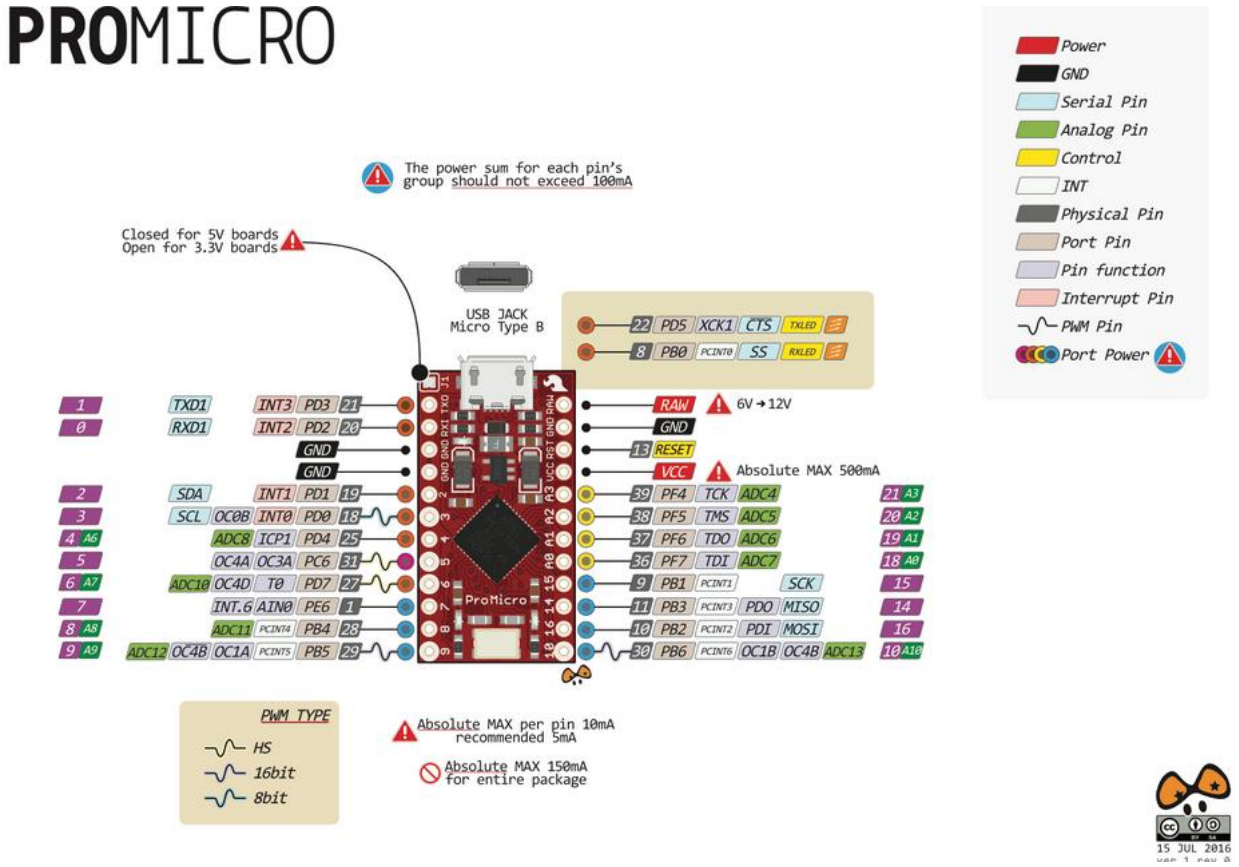


Figure 4 – Sparkfun Pro Micro Pin Number Diagram

You will notice that PD0 corresponds to pin number 3 and PD1 is pin number 2. Find out the specific pin numbers that correspond to the pins used for the IR sensors and the motor driver. **If you are using the 3DoT board, the pin numbering matches with the Pro Micro. If you are using the Pro Micro but want to wire it differently than shown in figure , please use the corresponding pin numbers.**

Now that you have all of the pin numbers, it is very straightforward to configure the pins as inputs or outputs. Using the pinMode() function, you can define if that pin number is an input or output by using something like the following.

```
pinMode(3,INPUT);
```

Once the pins have been configured, you can read the data from an input by using the digitalRead() or analogRead() functions. The main distinction is that analogRead() will utilize the analog to digital

converter (ADC) to produce a value between 0 and 1023. The digitalRead() will return a value of 0 or 1. If you wanted to read a value from one of the IR sensors, you could use digitalRead(2);

For outputting a value on a pin, you can use the digitalWrite() function. If we needed to output a value of 1 on PD7, this can be done with the following function call.

```
digitalWrite(6, HIGH);
```

With all of this information, you can now convert the code that you wrote before and develop the line following algorithm.

Make the code that you created for working with the GPIO registers into comments and keep them in the “ino” file. Add the equivalent code using the built-in functions.

Creating the 3DoTConfig “library” file

Now that you know how to configure the inputs and outputs, it would be optimal to group all of this code into one function and call it once in the setup() function instead of having all of the code there. Doing this will also allow you to quickly modify the configuration if anything changes and help make your code more readable. We will do this by defining the pin numbers with names that are more informative putting it all into a separate file to be included. It will be the first library file that is included in our code.

Making the 3DoTConfig.ino file

We are going to be utilizing a feature of the Arduino IDE that makes it very easy to use but it is not present in other C++ IDEs. The way we are defining this file is also not how it would be done normally. Library files are normally composed of the header file that provide the prototypes of the functions in the library and the .cpp file which has all of the C++ code. The Arduino IDE is unique in the sense that it can accommodate multiple “.ino” files in the same sketch project without needing to use the #include statement. This way, the user can separate their code as if it was a library file and still use those functions and definitions in their main file.

We will be doing this with the configuration of your robot. First, a new “.ino” file will need to be created. The figure below shows how to do this. Name the file 3DoTConfig.ino

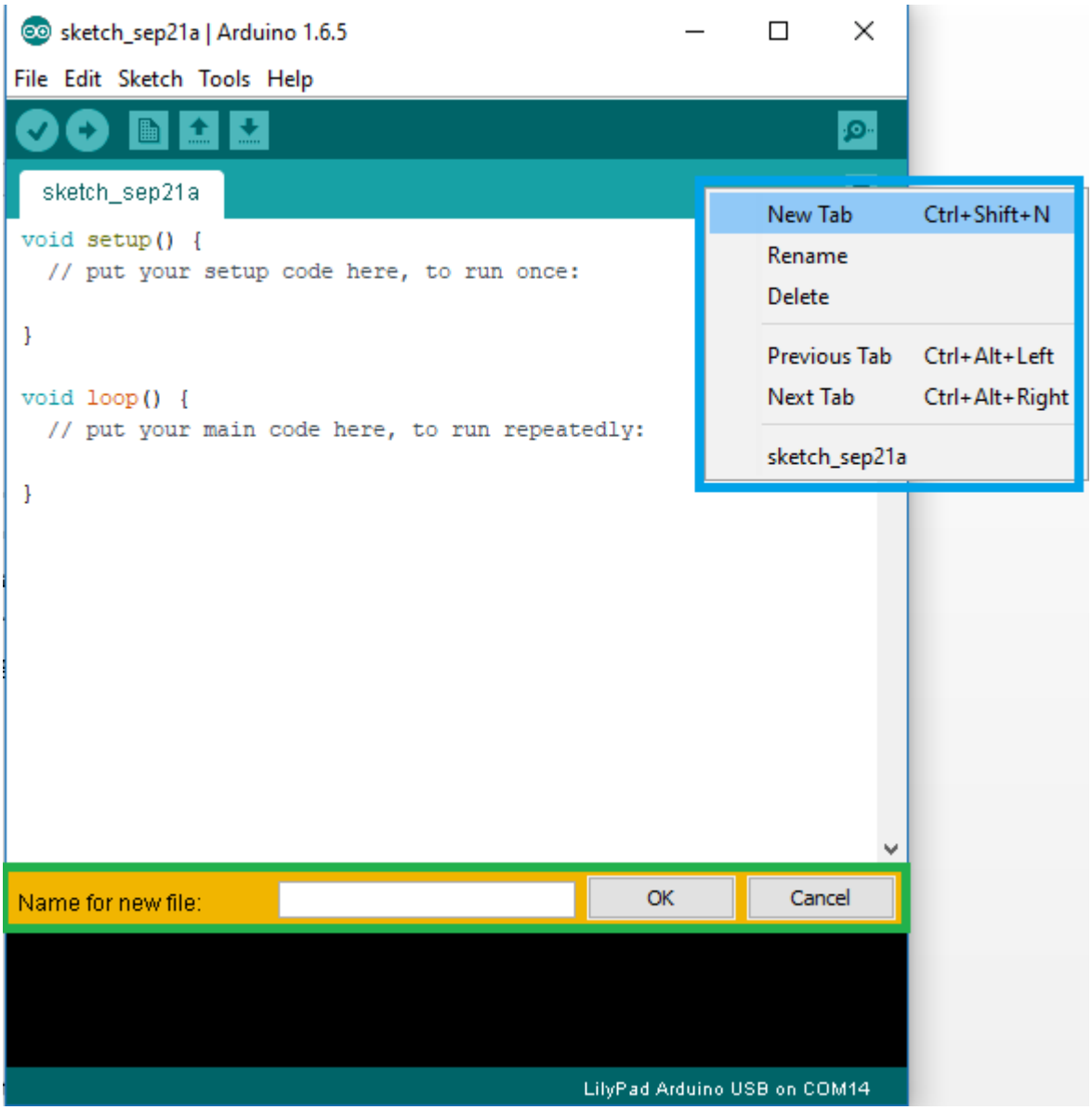


Figure 5 – Make a New Tab

Now that you have created the new file, write the following function definition and copy the code you developed for defining the inputs and outputs to the location indicated.

```
void Init3DoT() {
      // Put your code here
      //
};
```

Using #define to create informative names

We will be using the #define statement in order to associate specific pin numbers with a user defined name. These statements should go after any include files that you have in your main file. They could also be within any library files you have included but because we are making a “pseudo” library file, they will go at the very top of Lab 1 after the title block and before the setup() function.

You can define the pins however you would like. For example, the inner IR sensors could be specified as follows. The same can be done for the control signals to the motor driver, so that each motor can be clearly distinguished from the other.

```
#define IR_IN_L 3     // PD0 = pin 3, IR_IN_L is assigned to PD0
#define IR_IN_R 2     // PD1 = pin 2, IR_IN_R is assigned to PD1
#define BIN1 9        // PB5 = pin 9, BIN1 is assigned to PB5
#define BIN2 5        // PC6 = pin 5, BIN2 is assigned to PC6
```

These definitions can also be done by creating a variable that is set to the pin number value. These values will never change, so the const modifier will be used. The following is equivalent to the example given above. You can use either format for your program.

```
const int IR_IN_L = 3;
const int IR_IN_R = 2;
const int BIN1 = 9;
const int BIN2 = 5;
```

Line Following Algorithm

Now that you have finished the configuration, we can finally work on developing the line following algorithm. We will need to first get the motors to move the robot forward and then adjust them if the robot strays from the line.

Controlling the Motors

In order to get your robot to move forward, each motor will need to be configured to rotate in a specific direction. Depending on the orientation of the motors on your robot, you may need to have the motors going the same direction or opposite from each other. Using the following table from the motor driver datasheet, find the ideal combination that will get the robot moving forward.

Input				Output		
IN1	IN2	PWM	STBY	OUT1	OUT2	Mode
H	H	H/L	H	L	L	Short brake
L	H	H	H	L	H	CCW
		L	H	L	L	Short brake
H	L	H	H	H	L	CW
		L	H	L	L	Short brake
L	L	H	H	OFF (High impedance)		Stop
H/L	H/L	H/L	L	OFF (High impedance)		Standby

Figure 6 – Truth Table for TB6612FNG Motor Driver

You will see that you need to set several of the control pins to be either high or low. For example, the STBY pin (PB4) needs to be HIGH and this can be done using the digitalWrite() function or modifying the PORTB register. Do this for all 7 of the control pins. You will notice that this will put the motors at full speed.

Interpreting the IR sensor Data

Now that you have gotten the robot to move forward, we will now incorporate the IR sensors to provide feedback as your robot tries to follow the line. Depending on how you have configured the inputs, you can treat them as analog or digital inputs. If you are following the 3DoT shield configuration, the outer sensors are digital while the inner sensors are analog. The main distinction between the two is how the data should be interpreted.

If it is a digital input, the input value that comes out from the IR sensor when it is over the black line is HIGH (a value of 1). It will be LOW otherwise. This is determined by the microprocessor’s own threshold for digital logic and cannot be modified. While there is no customization, it is very useful for situations that have only two states and is easy to implement.

If there are more than two states or if a custom threshold needs to be defined, you will want to use an analog input. The voltage on the input pin is processed by the analog to digital convertor (ADC) and it will become an integer within the range of 0 to 1023. From there, you can create an algorithm to determine what that value represents. For example, if the value is less than 50, it could mean that you are detecting the white of the maze. If the value is greater than 50 but less than 700, it could be some range of shades of grey. If it is above 700, it could mean that black is detected. This gives you the flexibility of defining exactly what thresholds to look for.

Regardless of what configuration you have for the inputs, you will need to read the value from that pin and save it to a variable for your program to use. This is done with the following lines of code.

```
int x =  digitalRead(IR_IN_L);          // For digital inputs
int y = analogRead(IR_IN_R);            // For analog inputs
```

You can then create an algorithm that will adjust the motors according to the IR sensor inputs. If the robot starts to veer off the line towards the left, it means that the right motor is moving faster than the left motor. This can be due to the differences in the gearing of the motors or the quality of the motors themselves. The right IR sensor should detect the black line the moment it starts to go too far. In order to adjust and get the robot back on the line, your code should stop the right motor until the right IR sensor is back on the white surface of the maze. The same logic applies to when it veers to the right.

Develop the code to have the robot continuously follow a line and adjust if it veers off course. Place this code in the main loop function.

Lab 1 Deliverable(s)

All labs should represent your own work - DO NOT COPY.

Submit all of the files in your sketch folder. Make sure that the code compiles without any errors.

Checklist

Remember before you turn in your lab...

1. The configuration of the robot is written in the 3DoTConfig.ino file.
2. All of the configuration is done in the Init3DoT() function.
3. The Init3DoT() function should be called in the setup function.
4. The robot should move forward and use the IR sensors to follow a line