Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Section: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Names: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ and \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
| LABORATORY CHECK OFFS | |
| **Section 1**  Both motors turn using analogWrite() test code.  **Section 2**  Default prescaler value for Timer 0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_ (by printing Timer registers)  Default PWM frequency for Timer 0 \_\_\_\_\_\_\_\_\_\_\_\_\_ Hz (by calculation)  TCCR0A= \_\_\_\_\_\_\_\_\_\_\_\_\_\_ (HEX) TCCR0B= \_\_\_\_\_\_\_\_\_\_\_\_\_\_ (HEX)  Both motors turn using Timer 0 PWM.  **Section 3**  Go\_forward (Green), go\_backward (White), turn\_clockwise (Green Blue), turn\_counterclockwise (Green Red), stop (Red)  Code is neatly formatted with indents, separators, and no extraneous code.  **Section 4**  State machine implementation follows coding standards, has entry/state/exit housekeeping.  **Section 5**  Makes a rough box pattern using left turns with stops in between | \_\_\_\_  \_\_\_\_  \_\_\_\_  \_\_\_\_  \_\_\_\_ |
| Points | \_\_\_\_ |

Prelab:

Complete the separate prelab that involves building the robot base platform.

Learning Outcomes:

By the end of this lab you will be able to:

1. Configure Timer 0 to drive two motors independently
2. Create a state machine to structure the code controlling the robot.

Overview:

At the end of this lab, you will have a basic robot platform that can make turns. Your code will use a state machine structure so that it easier to read and debug. You will also be creating helper functions that also make the code easier to understand and debug. This code will be the basis for future labs so you must follow good coding practices when creating it. See the tips below.

References: Robot Platform: <https://arduinoinfo.mywikis.net/wiki/RobotKitMenu-5>

How to Succeed With This Lab:

Refer back to previous labs on state machines.

**Troubleshooting Tips:**

1) **Keep your code clean** (no extra code that isn’t used, remove it) and properly formated (tabs and indents are essential, as are section dividers //===== ). See the example below.

/\* Lab6\_Robot\_Platform.ino

Written by: Awesome Student, revised: Mar 18, 2019

Description: Shows section separators.

\*/

unsigned long timeStampStartOfLoopMs = 0; // variable name includes units of measurement

//==============================================================================

void setup() {

Serial.begin(115200);

Serial.println(“Robot Platform code version 1”);

} // setup

//==============================================================================

void loop() {

…

Serial.println(yaw);

delay((timeStepS\*1000) - (millis() - timeStampStartOfLoopMs));

} //loop

2) **“When in doubt, print it out.”** Everything that you know to be true, check it by printing it out to be sure.

**SECTION 1 – Dual Motors Running on analogWrite()**

**Procedure:**

1) Demonstrate your prelab work to your instructor for a signoff (showing both motors turning).

**SECTION 2 – Using Timer0 fast PWM to drive motor**

In this section, remove the analogWrite(), pinMode() or digitalWrite() code and insert your own code that uses Timer 0 to generate PWM signals for both motors and DDR and PORT commands for setting the direction signals.

**Procedure:**

1. Start with the sample code in the textbox below.

//Lab6\_section2\_timer0\_starter.ino

void setup() {

Serial.begin(9600);

Serial.println(F("Testing motor A and B using Timer0 in fast PWM mode 3."));

Serial.println(F("Requires external 9V battery pack."));

// add your code here, no digitalWrite() or pinMode() commands

}

void loop() {

// no code at all in main loop

}

1. In the setup() section:
   1. The prescaler for Timer 0 is already set by the Arduino default code (it is used for the millis() timer function.
   2. Before you do anything, print out the timer registers and figure out what the timer prescaler value is. Record it (as a decimal number) on the sign off sheet.
   3. Next, configure Timer 0 (not Timer 1) for fast PWM mode 3. Note that Timer 0 is already configured to provide the millis() timer so do not change the clock prescaler at all from its default value. Refer to the ATMEL datasheet section 15.9 for Timer 0’s register definitions.
   4. Configure the COM bits so that OC0A and OC0B (not Timer 1’s pins, but Timer 0) are cleared on compare. Record the values of TCCR0A and TCCR0B on the signoff sheet.
   5. Set OCR0A and OCR0B to 120.
   6. Use PORT commands to set the pin corresponding to OC0A/OC0B pins as outputs. Look back at the Arduino Pin Map diagram as needed. Note that no PWM signal will appear until the pin is configured as an output.
   7. The motor connected to OUTA and OUTB has its direction set by pins 10 and 9 and the PWM value set by pin 6. The motor connected to OUTC and OUTD has its direction set by pins 8 and 7 and the PWM value set by pin 5.
   8. Configure the pins that control both motors using DDR and PORT commands. For each motor, set one direction pin HIGH and one direction pin LOW.
   9. Print out the values of TCCR0A and TCCR0B after you have configured the timers. Record these values on the sign off sheet.
2. Take all of your new code for configuring Timer 0 and put it into its own function called: void configureTimer0RegisterForPWMtoDriveMotor(). Call this function in setup().
3. Upload the code and the motors should spin exactly as before in section 1.
4. Show the code to your instructor for a signoff. Once you are done with the demo, unplug the battery pack to conserve battery energy.

**SECTION 3 – Creating Helper Functions**

Now that the motors running, you are going to create a set of helper functions that set the direction of the left motor (motor AB) and the right motor (motor CD). The helper functions are named go\_forward, go\_backward, turn\_clockwise, and turn\_counterclockwise.

Each function takes an integer argument called rate that sets how fast the motors turn (rate is used to set the PWM value from 0 to 255). Each function also sets its own unique color on the RGB led so that it is easy to identify what mode the robot is in. The desired led colors are shown after the function names below.

void go\_forward(int rate) // RGB led color is GREEN = DIM\_GREEN\_COLOR

void go\_backward(int rate) // RGB led color is WHITE = DIM\_WHITE\_COLOR

void turn\_clockwise(int rate) // RGB led color is GREEN plus BLUE = DIM\_GREEN\_BLUE\_COLOR

void turn\_counterclockwise(int rate) // RGB led color is RED plus BLUE = DIM\_RED\_BLUE\_COLOR

void stop(int rate) // RGB led color is DIM\_RED = DIM\_RED\_COLOR

**Procedure:**

1. Create # defines at the top of your code for each color e.g.
   1. #define DIM\_GREEN\_BLUE\_COLOR 0x001F1F
2. Create the helper function called go\_forward and configure it to have one input called rate that is an integer. See the same code in the textbox below. Add the necessary code to set the appropriate pins high or low and the appropriate OCR0A/B value to the value of rate.
3. For testing this code, remove the tires from the yellow motors so that the robot does not move—it will make it easier if the robot stays still.

//==============================================================

void go\_forward(int rate) {

// set direction pins so both motors are turning forward,

// e.g. Motor AB has either pin 9 High, pin 10 Low or pin 10 High and pin 9 Low

// set PWM values (OCR0A and OCR0B to value of “rate”) so that motors turn fast or slow

// set RGB LED color to DIM\_GREEN\_COLOR

}

1. Place the definition of your new function below the main loop code and use a separator ( //==== ) to set it apart from the main loop.
2. In the main loop call the go\_forward function with a rate value of 120.

//==============================================================

void loop() {

go\_forward(120);

}

1. Install the RGB led, if it is not already connected. Make sure to set the color of the RGB led to Green in the go\_forward function. You will need to include all the code required to run the RGB led (e.g. defining the pins names, setting pin direction) including the display\_color\_on\_RGB\_led() function.
2. Test your code and modify the motor direction pin settings (always with one direction pin HIGH and one pin LOW) until the robot motors are both turning forward.
3. Next, create the go\_backward helper function and test it. Make sure to set the color of the RGB led to White as given in the list of functions above.
4. Complete the rest of the helper functions. Format your code neatly and remove any unused code.
5. Once all of your functions have been independently tested, modify your main loop to run through each mode as shown in the code box below.

//================================================================

void loop() {

go\_forward(120);

delay(5000);

go\_backward(120);

delay(5000);

turn\_clockwise(120);

delay(5000);

turn\_counterclockwise(120);

delay(5000);

stop(0);

delay(5000);

}

1. Show your code running through mode to your lab instructor and get a signoff.

**SECTION 4 – State Machine Implementation**

In this section, you will create a state machine to control the robots motion and allow you flexibility for future software enhancements.

**Procedure:**

1. Start with your code from the previous section and save it under a new file name.
2. Define a new type of enumerated variable (directionState\_t) to represent the state of the robot. Include the enumerated states of STOPPED, FORWARD, BACKWARD, LEFT\_TURN, RIGHT\_TURN

enum directionState\_t { //list all possible states here }

// declares a new type of data (directionState\_t), with defined values

1. Create two global variables of this new type. Name the first one directionState and set its value to STOPPED. Name the second variable previousDirectionState and set its initial value to -1.
2. Create a global Boolean variable called isNewState and initialize it to true.
3. In the main loop, set isNewState to the appropriate value to detect a change in the directionState. Also in the main loop, set previousDirectionState as needed to get proper operation of the statemachine. Refer to the lab on statemachines.
4. In the main loop, set up a switch case structure with cases for each state.
5. In the entry housekeeping, start the motion of the motor (which will set the RGB led color) by calling the appropriate helper function, print the name of the state “new state is FORWARD” and set the state timer to 0.

switch (directionState) {

case FORWARD:

//entry housekeeping

//state business

//exit housekeeping

break;

1. In the state business, increment the stateTimer, e.g. stateTimer++

**Discussion – Measuring Time Using a stateTimer**

There are several ways to control the rate at which a section of code is executed. In a microcontroller, the best solution is often to setup a timer interrupt, as this will allow very precise control. Each time the interrupt is called, the code is executed once, guaranteeing that the code will run at a constant rate (assuming no other significant delays occur due to other interrupts).

However, a simpler solution can often be created by using a simple delay function at the bottom of the loop. With this method, the main code is loop is executed regularly, every so many milliseconds and a counter keep track of how many times the main loop has run. This counter value can be converted to a time value by multiplying the counter by the execution time for one pass through the main code loop.

This method is not as precise as the interrupt method, but it also doesn’t require use of a dedicated timer (which might be needed for other purposes such as PWM).

1. In the exit housekeeping, check for the stateTimer to see if 5000 milliseconds have passed. If it has, change the state to the next state in the enumerated list of stateDirections, running through all the states.
2. Configure the main loop so that it is executed only once every 10 milliseconds, for example using the code below.

//==============================================================================

void loop() {

timeStampStartOfLoopMs = millis(); // mark the time

…

// Wait until a full 10msec has passed before next reading

delay(10 - (millis() - timeStampStartOfLoopMs));

} //loop

1. Show to the lab instructor for a signoff.

**SECTION 5 – Making The Robot Move In a Square Box**

In this section, you will modify the state machine so that the robot makes a box pattern on the floor. Start by having the robot go forward for 1.5 seconds, stop for 1.5 seconds, turn counterclockwise for 1.5 seconds, go forward for 1.5 seconds, and repeat until a full four turns are completed. This is rough exercise to understand how to program a sequence of moves for the robot. Don’t worry about creating a perfect square box. Adjust the times as needed to improve but don’t perfect the box maneuver. The goal here is to understand the challenge of making precise turns.

**Procedure:**

1. Change the state machines exit conditions to get the box pattern.
2. Get a sign off once you have made a rough four-sided box maneuver.