EXERCISE #3 MEMORANDUM

TO: RON BROCKHOFF

FROM: KAVIAN KALANTARI

SUBJECT: EXERCISE #3 DETAILED MEMORANDUM

DATE: 9/21/21

The main purpose of this exercise was to use the SN754410 driver [1] and become accustomed with using an H-Bridge to control a motor [2] via external power supply [3]. This external power supply allows the MCU (the Arduino) [4] to not have to supply the current driving the motor. For most MCUs, the limited output current makes it so having a dedicated external power supply for driving motors and other high current draw devices a must. The flyback diodes [5] also assist in protecting the MCU, as they prevent the current spikes from switching the motor on and off from reaching the MCU through the direction and enable pins. A 10k resistor [6] is used to make the driver normally low, but it is toggled with pin 45's state.

By setting pin 41 to high, and pin 42 to low, (direction), and pin 45 to high (enabling) the motor should spin forwards. By alternating low/high states on pins 41 and 42 the direction can be reversed.

The SN754410 chip can control 2 motors at the same time, but for the purposes of this exercise we are only utilizing half of its functionality for testing. Figure 1 shows all relevant hookups, and figure 2 shows them in a schematic form.

The code written allows for the user to enter either a plus or minus via sending text to serial port (for motor direction), and then make the motor spin at full speed or stop. Overall, the code doesn't have much extra functionality, but it accomplishes the task.

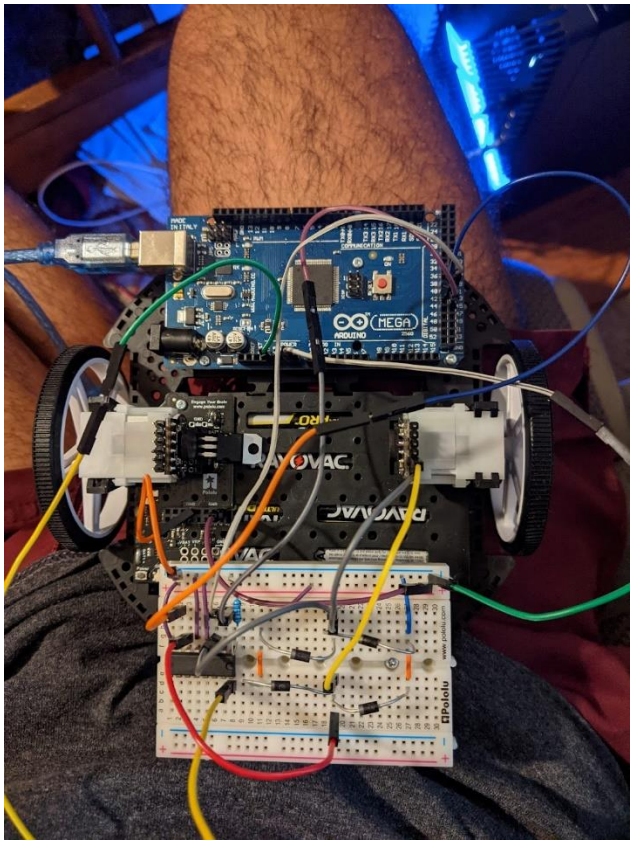


Figure 1, real life hookups

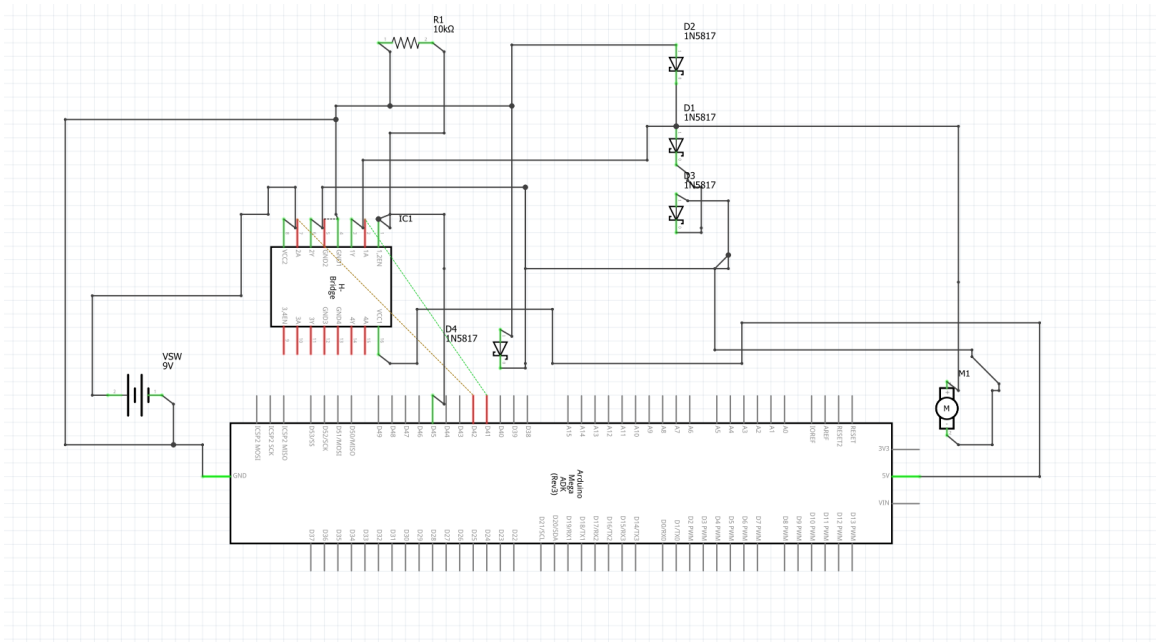


Figure 2, circuit diagram of all hookups

Relevant Arduino Code:

```
const int pwmPin = 45;
const int hbPin1 = 41;
const int hbPin2 = 42;
//creating const ints for pins used in project

void setup()
{
   Serial.begin(115200);
   //set output baud rate
   pinMode(hbPin1, OUTPUT);
   pinMode(hbPin2, OUTPUT);
   pinMode(pwmPin, OUTPUT);
   //setting H-Bridge pins as outputs
}

void SerialChecker()
{
   //creating simple function to check if data is sent to serial port
   while (Serial.available() == 0)
   {
   }
}

void loop()
{
   Serial.println("Enter motor direction ('+' or '-')");
   SerialChecker();
   String direction_s;
   direction_s = Serial.readString();
   //creating direction_s to represent user value as string
   if (direction_s == "+")
   {
      digitalWrite(hbPin1, HIGH);
      digitalWrite(hbPin2, LOW);
      Serial.println("Direction set to '+'");
   }

   else if (direction_s == "-")
   {
      digitalWrite(hbPin1, LOW);
      digitalWrite(hbPin2, HIGH);
      Serial.println("Direction set to '-'");
   }

   else
   {
      Serial.println("You entered something other than '-' or '+'");
      Serial.println("Automatically setting direction to '+'");
      digitalWrite(hbPin1, HIGH);
      digitalWrite(hbPin2, LOW);
```

```
    //if user enters something wacko just set default value
  }

  while (true)
  {
    Serial.println("Start or stop motor ('S' = full speed, 'X' = turn off), 'RS' to restart");
    SerialChecker();

    String motorState;
    motorState = Serial.readString();
    motorState.toUpperCase();
    //setting all inputs to be in upper case for easier check statements

    if (motorState == "S")
    {
      digitalWrite(pwmPin, HIGH);
      Serial.println("Motor Speed: HIGH");
      //setting pwmPin to HIGH if user enters "S" or "s"
    }

    else if (motorState == "X")
    {
      digitalWrite(pwmPin, LOW);
      Serial.println("Motor Speed: OFF");
      //setting pwmPin to LOW if user enters "X" or "x"
    }

    else if (motorState == "RS")
    {
      Serial.println("Restarting");
      break;
      //if user wants to change direction, they must enter "RS" to exit while loop
    }

    else
    {
      Serial.println("You entered something other than 'S' or 'X'");
      Serial.println("Setting speed to 'X' (off)");
      //setting default state to off if garbage is sent to serial port
    }

    delay(250);
    //250ms delay until prompts are shown again
  }

  digitalWrite(pwmPin, LOW);
  digitalWrite(hbPin1, LOW);
  digitalWrite(hbPin2, LOW);
  //setting all pins to low if program is restarted with "RS"
}
```

Relevant Score Sheet:

Exercise #3	**Score: 20**
Code Comments	
CODE REVIEW COMMENTS: -5 while(true) in second half of loop prevents user from every changing motor direction after initial selection. GRADER COMMENT: Interface does not allow directoin to be changed after initial selection.	
Video Comments	
Did you figure out what was going on with your bridge?	

BIBLIOGRAPHY

[1] Texas Instruments, "SN754410 Quadruple Half-H Driver," January 2015. [Online]. Available: ME 615 Canvas Page. [Accessed 21 September 2021].

[2] Polulu, "120:1 Mini Plastic Gearmotor HP, Offset 3mm D-Shaft Output, Extended Motor Shaft," [Online]. Available: https://www.pololu.com/product/1520. [Accessed 21 September 2021].

[3] Polulu, "Power Distribution Board for Romi Chassis," [Online]. Available: https://www.pololu.com/product/3541. [Accessed 22 September 2021].

[4] Arduino, "Arduino MEGA 2560 & Genuino MEGA 2560," [Online]. Available: https://www.arduino.cc/en/pmwiki.php?n=Main/arduinoBoardMega2560. [Accessed 21 September 2021].

[5] ON Semiconductor, "1N5817-D.pdf," July 2006. [Online]. Available: ME 615 Canvas Page. [Accessed 28 September 2021].

[6] Stackpole Electronics, Inc., "sei-cf_cfm.pdf," 22 October 2020. [Online]. Available: ME 615 Canvas Page. [Accessed 28 September 2021].

ME 615: Applications in in Mechatronics

Exercise #3

Instructions:

Complete each of the following steps. When you have completed all steps assigned, upload the files associated with the exercise on Canvas. Make sure the filenames you upload as part of your exercise submission have the exact names specified in each steps of this exercise. If the filenames do not match exactly, then your submission will not be graded.

Background

Build an H-bridge motor driver circuit to control one of the brushed DC motors attached to the wheels of the Romi chassis. You need to implement the circuit on the breadboard you previously attached to the robot.

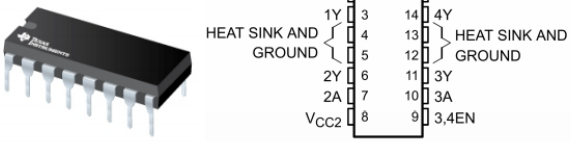
Brushed DC Motor

The brushed DC motor is a type of motor that operates on DC voltage and current. It is bidirectional, meaning if the supply terminals are swapped the motor turns in the opposite direction. An H-bridge helps in achieving this functionality without physically having to swap the terminals.

The DC motor assemblies (with gearbox and encoder already installed) used by the Romi chassis are manufactured by Pololu. Specifications for this motor can be found at: https://www.pololu.com/product/1520. This motor is capable of high speeds, with a rated output speed of 150 rpm.

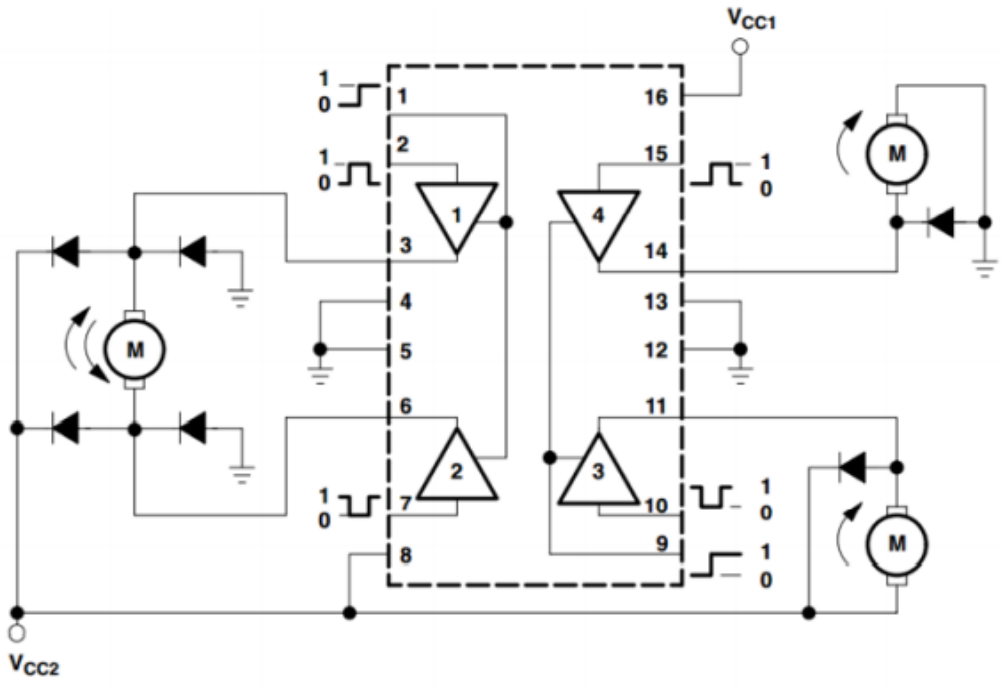
H-Bridge

H-Bridges are commonly used to switch direction of current through a motor. They allow a digital device to activate motor rotation in both directions. The H-bridge IC that we use for this exercise is the TI SN754410, which is a quadruple half H-bridge driver IC. It has four half H-bridges that can be used either independently for motors with unidirectional applications, or in pairs for motors with bidirectional applications. The datasheet for SN754410 can be found at: http://www.ti.com/lit/ds/symlink/sn754410.pdf. The pinout for the IC is shown below:



To control a single motor bidirectionally only two channels of the H-bridge driver IC are needed. Either channels 1 and 2 as a pair to control the motor (MR), or channels 3 and 4 to control the other motor (ML).

The figure below shows the possible wiring configurations that can be used with the SN754410 H-Bridge IC. Note that in the figure, the interfaces at ports 3 and 4 are shown for unidirectional operation of two motors. The left side of the diagram (ports 1 and 2) shows the wiring schematic for bidirectional operation of a motor, which you will configure as part of this exercise.



The pins of the SN754410 H-bridge IC are described in Table 1.

Pin Number	Pin Name	Pin Function
1	1,2EN	Enable signal for driver channels 1 and 2 (active high input)
2	1A	Input control signal for driver channel 1
3	1Y	Output signal to the right motor for driver channel 1
4	Heat Sink/GND	Connect to Ground
5	Heat Sink/GND	Connect to Ground
6	2Y	Output signal to the right motor for driver channel 2
7	2A	Input control signal for driver channel 2
8	VCC2	Power supply for motors (connect to VSW = 7.2V to 9V)
9	3,4EN	Enable signal for driver channels 3 and 4 (active high input)
10	3A	Input control signal for driver channel 3
11	3Y	Output signal to the left motor for driver channel 3
12	Heat Sink/GND	Connect to Ground
13	Heat Sink/GND	Connect to Ground
14	4Y	Output signal to the left motor for driver channel 4
15	4A	Input control signal for driver channel 4
16	VCC1	Power supply for internal logic (connect to 5V)

Note: For this exercise you will be using the bidirectional configuration for one side of SN754410 to implement bidirectional control of one DC motor. The SN754410 IC comes in a PDIP package and has an orientation identifier to help locate Pin 1 on the chip: There is a Half Circle or Notch on the end of the chip, Pin 1 is to the left of the notch. IC pins are numbered in a counterclockwise fashion from Pin 1.

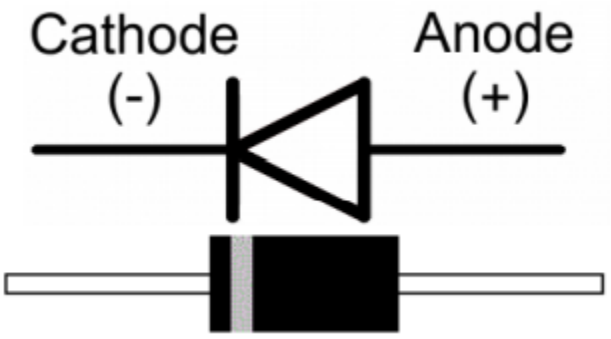
For the configuration shown in the prior schematic, setting the enable pins (pin 1 and pin 9) to high will enable the associated driver channels. As such, pull-down resistors are typically attached to the enable pins to make them low (i.e. disabled) by default.

Setting pin 2 of SN754410 high and simultaneously setting pin 7 of SN754410 low will turn the motor (attached to the output pins 3 and 6) in one direction. To turn the motor in the other direction, pin 2 should be low and pin 7 of SN754410 should be high.

To operate the motor in either direction at full speed, a high-level signal (5V) can be applied to one of these two pins while maintaining the other pin low (GND). However, for this exercise these input pins will be attached to PWM signals so the speed of the motor can be varied.

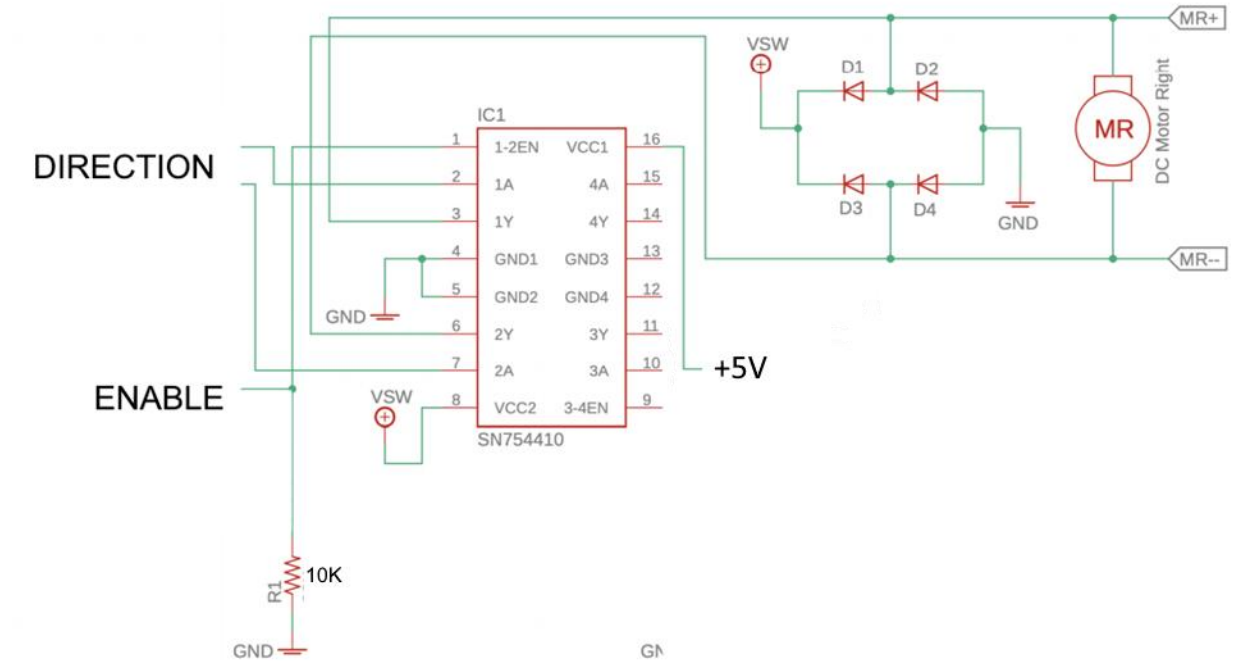
Diodes

Diodes are semiconductor devices used to control the direction of current flow. They are used often in mechatronic devices. The 1N5819 diodes provided in your kit will be used as flyback diodes for the motor driver. Flyback diodes provide alternative path for current in motor coil and mitigates arcing/excessive current through transistors, to avoid damaging the transistors. The silver line on 1N5819 diode indicates the cathode pin, which matches the vertical line in the diode circuit symbol (see below);

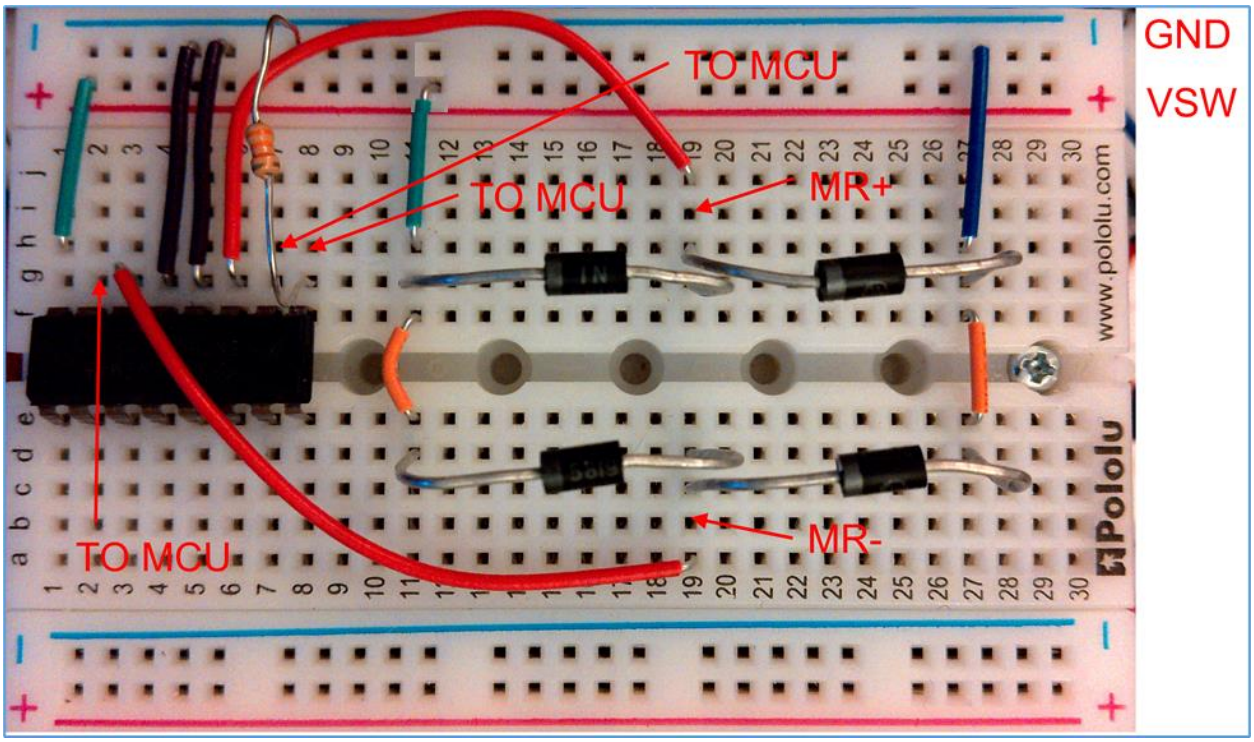


Problem Statement

Use your solderless breadboard to construct the motor driver circuit using the H-Bridge IC included in your lab kit.



One possible configuration of this circuit is shown below:



Note: The flyback diodes provide paths for current to flow when the drivers are switched off and are designed to protect the H-Bridge IC from damage. Pay extra attention to the direction of the diodes when implementing them on the breadboard.

Note: A 10K pull-down resistors is connected to the SN754410 enable to make it low (disabled) by default. This enable signal is active-high and will be toggled between 5V and 0V by connecting it to a digital pin on the MCU.

Once you have constructed the circuit disconnect the right motor from the power distribution board and attach it to your motor driver circuit to test the associated functionality. If you constructed the circuit as shown in the figure above, then connect the M1 encoder terminal to MR+ and the M2 encoder terminal to MR-.

You will power the motors with the the six AA batteries installed in the robot chassis, which provide a voltage in the range of 7.2V to 9V through the VSW pin on the power distribution board. If you constructed the circuit as shown in the figure above, then connect the power distribution board pin labelled VSW to the positive rail of the breadboard, and a GND terminal on the distribution board to the negative rail of the breadboard.

Connect PIN ***41*** of the Arduino to H-Bridge terminal ***1A***, PIN ***42*** of the Arduino to the H-Bridge terminal ***2A***, and PIN ***45*** to the H-Bridge ***1,2EN*** terminal.

The expected operation is as follows: Applying 5V to pin 1A (pin 2) and GND to pin 2A (pin 7) results in the right motor turning in forward direction. To turn the right motor in backward direction, the pin 1A is connected to GND and pin 2A to 5V. You will be using the Arduino Mega (i.e, MCU) to control the voltage levels for these two pins.

Using these components write a program, which meets the requirements listed below:

Requirements

1. Create a sub directory named “ex03a”, and an associated Arduino sketch file named “ex03a.ino”, in your Arduino directory.
2. Create a program that includes a ***setup()*** and ***loop()*** routine. Add code to the ***setup()*** routine that opens a connection to the serial port at a baud rate of 115,200 bits/second.
3. Perform the following calculations within the ***loop()*** routine of your sketch:
    a. Prompt the user via the serial port to enter the motor direction. If the user types in **+**, then the motor should turn one direction. If the user types in **-**, then the motor should turn the opposite direction.
    b. Prompt the user to start and stop the motor. If the user enters ***S*** then the motor should turn on at full speed. If the user enters ***X*** the motor should turn off. This can be accomplished by toggling the digital state of pin ***45*** to either HIGH or LOW.

c. If the user enters **+** in Step 2a, then set the signal on pin ***41*** to HIGH and set the signal on pin ***42*** to LOW. If the user entered **–** in step 2a, then setup the signal on pin ***42*** to HIGH and set the signal on pin ***41*** to LOW.

d. Ensure that for a when S is entered the motor turns at full speed, and that when X is entered that the motor is in the off state.

e. Delay 250 milliseconds.

3. In your project notebook provide pertinent spec sheets and detailed pictures that show your breadboard setup, your connections to the Arduino, connections to the motor, and connections to the Romi chassis. Clearly label connection points in your pictures. Supporting diagrams are also suggested.

4. You will also need to record a short video (1 minute or less) and post the video to canvas. The video should demonstrate the motor running at full speed in both directions. It should also demonstrate the ability to turn off the motor. Verbal narration is highly encouraged (as well was tasteful jokes).

Submission

Upload the files ex03a.ino to canvas once the exercise has been completed. The code will be evaluated as follows:

1. It will be evaluated to make sure the code is implemented using the methods outlined in this exercise.

2. Your program will be evaluated the ensure that the program successfully compiles.

3. Your program will be evaluated to ensure that it includes proper comments that explain the functionality of the source code.

4. Your program will be evaluated to ensure that it runs correctly, collects the input as described, and outputs information as described.

You are also expected to include proper documentation for this exercise in your project notebook.

EXERCISE #4 MEMORANDUM

TO: RON BROCKHOFF

FROM: KAVIAN KALANTARI

SUBJECT: EXERCISE #4 DETAILED MEMORANDUM

DATE: 9/28/21

This exercise's intention is to show functionality of both motors connected to a single DRV8835 dual motor driver carrier [1], as well as making sure all components are properly mounted to the Romi Chassis [2] for movement. On a button [3] depress, the robot will rotate to the right, then to the left, then spin one way, then spin the other way. These operations have various duty cycles. Then the robot stops. This display of changing each motor's [4] direction and speed is due to the ability of the DRV8835 change speed by pulsing the "PHASE" connections with a PWM signal, and change directions by flipping the "ENABLE" pin from low/high states. This motor driver is slightly different to the one used in exercise 5 as it requires one less pin for direction control. It is similar in the fact that it can be used to supply external power to the motors [5], thereby bypassing the relatively low current output of the Arduino [6].

In the code, a small debounce is added for button press, as well as making sure that the movement doesn't begin until the user actually lets go of the button (button depress). After this the pins are then set to the states that correspond to the correct motion detailed in Exercise4. Serial statements were used extensively for debugging, as my brain had a fairly difficult task understanding what clockwise and counterclockwise looked like visually without using a picture of a clock. Figure 1 shows the full assembly mounted on the Romi chassis, and figure 2 shows the circuit diagram.

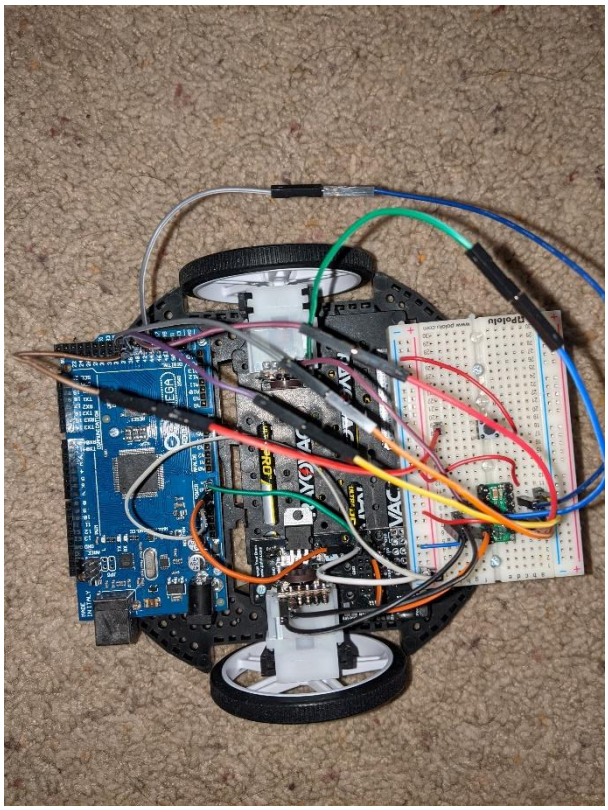


Figure 1, components mounted to Romi chassis

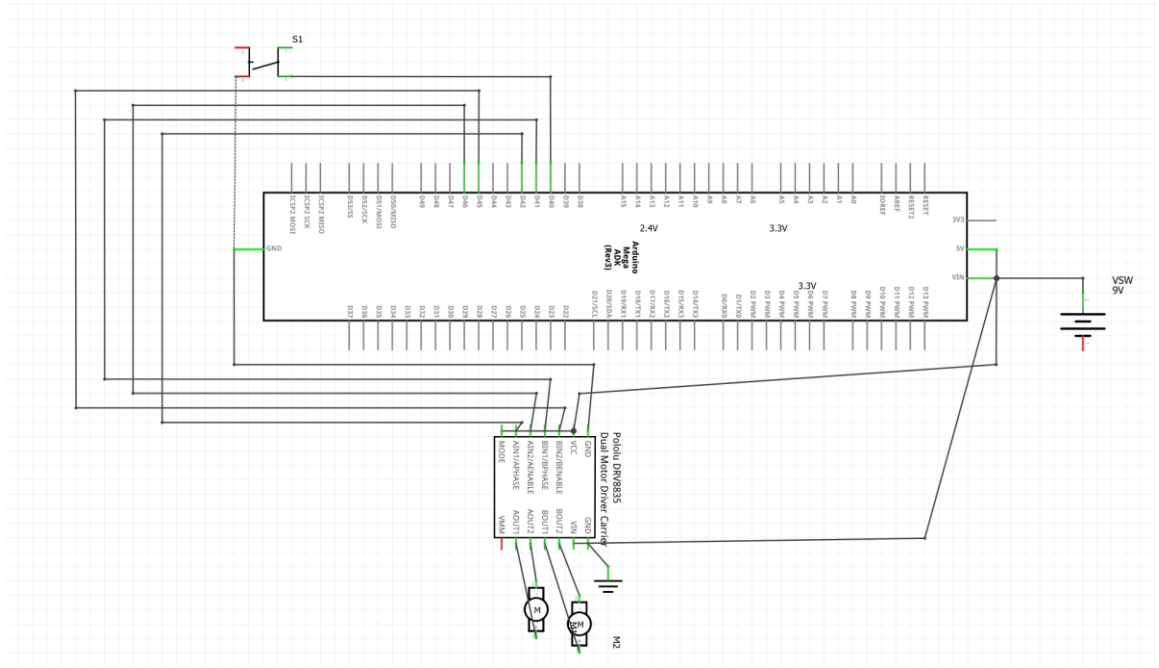


Figure 2, schematic view of components

Relevant Arduino Code:

```
const int pwmPinLeft = 45;
const int pwmPinRight = 46;
const int hbPinRight = 41;
const int hbPinLeft = 42;
const int button = 40;
//creating const ints for pins used in project

float dcPercent;
//used for easily setting duty cycle from percentage

void setup()
{
   Serial.begin(115200);
   //set output baud rate
   pinMode(hbPinRight, OUTPUT);
   pinMode(hbPinLeft, OUTPUT);
   pinMode(pwmPinLeft, OUTPUT);
   pinMode(pwmPinRight, OUTPUT);
   //setting all H-Bridge pins as outputs

   pinMode(button, INPUT_PULLUP);
   //setting button as input with pullup resistor
}

void StopMotion()
{
   //creating function to easily set all pins to low
   //and stop robot from moving
   digitalWrite(hbPinRight, LOW);
   digitalWrite(hbPinLeft, LOW);
   digitalWrite(pwmPinLeft, LOW);
   digitalWrite(pwmPinRight, LOW);
   Serial.println("All motors have been stopped.");
}

void loop()
{
   if (digitalRead(button) == LOW)
   //reading low state as button press, because of using pullup resistor
   {
      delay(10);
      //debouncing delay
      if (digitalRead(button) == HIGH)
      //reading depress of button (user can hold button down, robot wont move until button is let
go)
      {
         delay(150);
         //adding small delay to allow user time to move their hand away after
         //letting go of button
```

```
        //PART A, RIGHT CLOCKWISE @40% DC
        digitalWrite(hbPinRight, HIGH);
        dcPercent = 40.0;
        analogWrite(pwmPinRight, dcPercent * 255 / 100.0);
        //dcPercent * 255 / 100.0 is correct value to pass to pwmPinLeft
        //because pwm range is 0-255 not 0-100
        Serial.println("Direction of rightmotor is set to 'Clockwise'");
        delay(2000);
        //testing with robot mounted stationary so using serial statements

        //StopMotion();
        digitalWrite(pwmPinRight, 0);
        //stopping all motion

        //PART B, LEFT COUNTERCLOCKWISE @40% DC
        digitalWrite(hbPinLeft, LOW);
        analogWrite(pwmPinLeft, dcPercent * 255 / 100.0);
        Serial.println("Direction of leftmotor is set to 'Counterclockwise'");
        delay(2000);

        StopMotion();

        //PART C, LEFT + RIGHT CLOCKWISE @80% DC
        digitalWrite(hbPinRight, LOW);
        digitalWrite(hbPinLeft, LOW);
        dcPercent = 80.0;
        analogWrite(pwmPinLeft, dcPercent * 255 / 100.0);
        analogWrite(pwmPinRight, dcPercent * 255 / 100.0);
        //Serial.println("Direction of robot is set to 'Clockwise'");
        delay(2000);

        StopMotion();

        //PART D, LEFT + RIGHT COUNTERCLOCKWISE @30% DC
        digitalWrite(hbPinRight, HIGH);
        digitalWrite(hbPinLeft, HIGH);
        dcPercent = 30.0;
        analogWrite(pwmPinLeft, dcPercent * 255 / 100.0);
        analogWrite(pwmPinRight, dcPercent * 255 / 100.0);
        //Serial.println("Direction of robot is set to 'Counterclockwise'");
        delay(2000);

        StopMotion();
    }
  }
}
```

Relevant Score Sheet:

Exercise #4	**Score: 25**
Code Comments	
CODE REVIEW COMMENTS: Codde works as expected.	
Video Comments	
Good narration and video.	

BIBLIOGRAPHY

[1] Polulu, "DRV8835 Dual Motor Driver Carrier," [Online]. Available: https://www.pololu.com/product/2135. [Accessed 28 September 2021].

[2] Polulu, "Romi Chassis Kit - Black," [Online]. Available: https://www.pololu.com/product/3500. [Accessed 28 September 2021].

[3] TE Connectivity, "ENG_CD_1825910_C10.pdf," [Online]. Available: ME615 Canvas Page. [Accessed 28 September 2021].

[4] Polulu, "120:1 Mini Plastic Gearmotor HP, Offset 3mm D-Shaft Output, Extended Motor Shaft," [Online]. Available: https://www.pololu.com/product/1520. [Accessed 21 September 2021].

[5] Polulu, "Power Distribution Board for Romi Chassis," [Online]. Available: https://www.pololu.com/product/3541. [Accessed 22 September 2021].

[6] Arduino, "Arduino MEGA 2560 & Genuino MEGA 2560," [Online]. Available: https://www.arduino.cc/en/pmwiki.php?n=Main/arduinoBoardMega2560. [Accessed 21 September 2021].

ME 615: Applications in in Mechatronics

Exercise #4

Instructions:

Complete each of the following steps. When you have completed all steps assigned, upload the files associated with the exercise on Canvas. Make sure the filenames you upload as part of your exercise submission have the exact names specified in each steps of this exercise. If the filenames do not match exactly, then your submission will not be graded.

Background

Build an H-bridge motor driver circuit to control both of the brushed DC motors attached to the wheels of the Romi chassis. You will replace the motor control circuit you created for the last exercise and use a new motor driver to drive both motors.

Brushed DC Motor

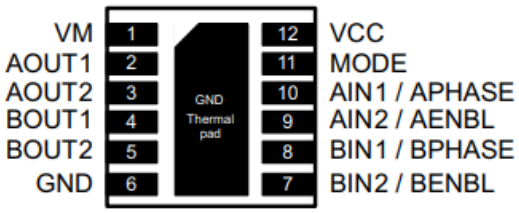
The brushed DC motor is a type of motor that operates on DC voltage and current. It is bidirectional, meaning if the supply terminals are swapped the motor turns in the opposite direction. An H-bridge helps in achieving this functionality without physically having to swap the terminals.

The DC motor assemblies (with gearbox and encoder already installed) used by the Romi chassis are manufactured by Pololu. Specifications for this motor can be found at: https://www.pololu.com/product/1520. This motor is capable of high speeds, with a rated output speed of 150 rpm.

H-Bridge

H-Bridges are commonly used to switch direction of current through a motor. They allow a digital device to activate motor rotation in both directions. We will use for this exercise a DRV8835 dual H-bridge driver IC for this exercise. The datasheet for DRV8835 can be found at: DRV8835 Dual Low-Voltage H-Bridge IC datasheet (Rev. H) (ti.com). The pinout for the IC is shown below:

DSS Package
12-Pin WSON With Exposed Thermal Pad
Top View



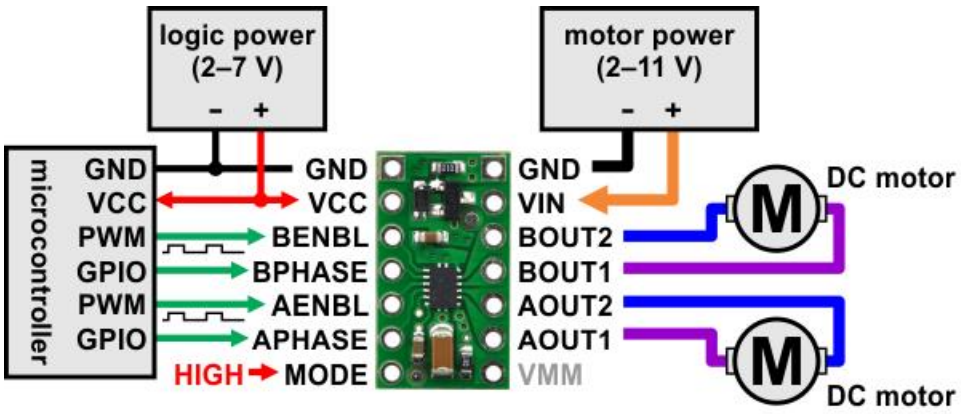
The pins of the DRV8835 H-bridge IC are described in table shown below. For more information refer to the specification sheet for the IC.

Pin Functions

PIN		I/O(1)	DESCRIPTION	EXTERNAL COMPONENTS OR CONNECTIONS
NAME	NO.			
POWER AND GROUND				
GND, Thermal pad	6	—	Device ground	
VM	1	—	Motor supply	Bypass to GND with a 0.1-μF (minimum) ceramic capacitor
VCC	12	—	Device supply	Bypass to GND with a 0.1-μF (minimum) ceramic capacitor
CONTROL				
MODE	11	I	Input mode select	Logic low selects IN/IN mode Logic high selects PH/EN mode Internal pulldown resistor
AIN1/APHASE	10	I	Bridge A input 1/PHASE input	IN/IN mode: Logic high sets AOUT1 high PH/EN mode: Sets direction of H-bridge A Internal pulldown resistor
AIN2/AENBL	9	I	Bridge A input 2/ENABLE input	IN/IN mode: Logic high sets AOUT2 high PH/EN mode: Logic high enables H-bridge A Internal pulldown resistor
BIN1/BPHASE	8	I	Bridge B input 1/PHASE input	IN/IN mode: Logic high sets BOUT1 high PH/EN mode: Sets direction of H-bridge B Internal pulldown resistor
BIN2/BENBL	7	I	Bridge B input 2/ENABLE input	IN/IN mode: Logic high sets BOUT2 high PH/EN mode: Logic high enables H-bridge B Internal pulldown resistor
OUTPUT				
AOUT1	2	O	Bridge A output 1	Connect to motor winding A
AOUT2	3	O	Bridge A output 2	
BOUT1	4	O	Bridge B output 1	Connect to motor winding B
BOUT2	5	O	Bridge B output 2	

(1) Directions: I = input, O = output

The DRV8835 bare IC consists of a small leadless package that is difficult to connect to directly. As such, we will be using the DRV8835 breakout board that consists of the DRV8835 IC and a FET for reverse battery protection mounted in a 14-pin DIP package. When connecting to the DRV8835, the motor and motor power connections are made on one side of the board and logic power and control connections are made on the other as shown below:

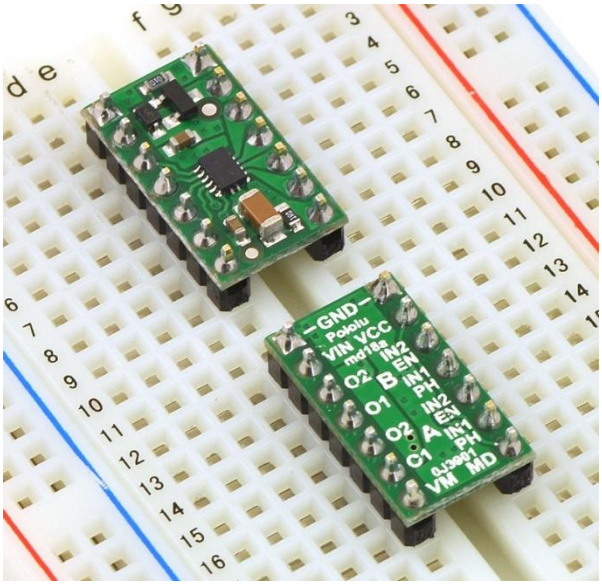


The value of VIN can range between 0 V and 11 V, and in our case, we will supply the voltage from VSW on our power distribution board. A logic voltage between 1.8 V and 7 V must be supplied to the VCC pin, and is typically connected to the 5V pin of the Arduino Mega.

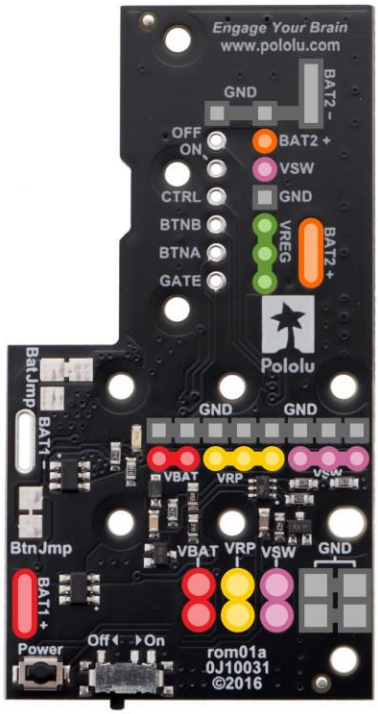
The DRV8835 features two possible control modes: IN/IN and PHASE/ENABLE. For this exercise, we will use the PHASE/ENABLE control mode. To enable this control mode, the MODE pin must be set high, either with a pull-up resistor or by connecting the pin to a driving-high I/O line. Once this mode is enabled, then the PHASE pin determines the motor direction and the ENABLE pin can be supplied with a PWM signal to control the motor speed.

Problem Statement

For this exercise you will using the DRV8835 dual motor driver carrier in unison with the power distribution board to control the right and left motors of your robot. Two 1×7-pin breakaway 0.1″ male headers are included with the DRV8835 dual motor driver carrier, which can be soldered in to use the driver with your breadboard or 0.1″ female connectors. ***Note: The headers might pship as a single 1×14 piece that can be broken in half.*** You can mount these headers in one of two ways, and the choice of method is based on how you plan to connect and mount the module. The two possible configurations are shown below:

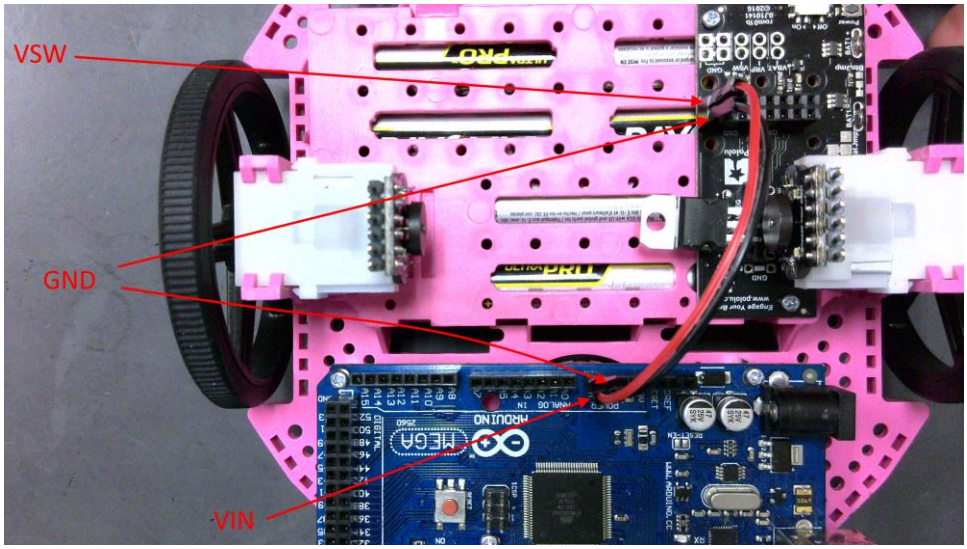


For this exercise you will using the DRV8835 dual motor driver carrier in unison with the power distribution board to control the right and left motors of your robot. Locate the 2X8 pin header you soldered to the power distribution board and identify the pins that are connected to ***GND*** and ***VSW***. These connections are shown in the next figure. There are 8 possible ***GND*** pins that can be used, and three pins connected to ***VSW***. You can use any of these pins for making the required connections.

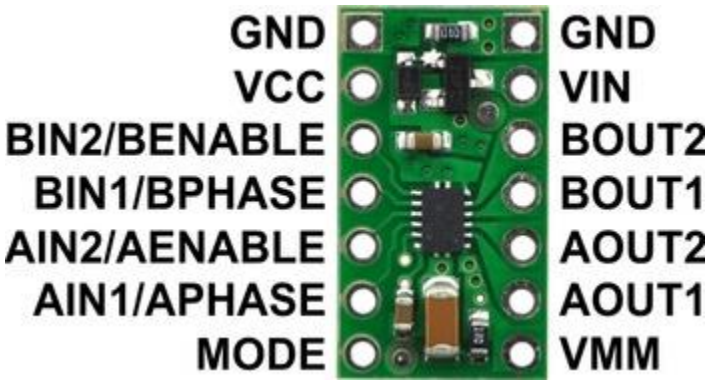


- **VBAT** (BAT1+)
- **VRP** (after reverse protection)
- **VSW** (after switch)
- **VREG** (distribution bus for optional added regulator)
- **BAT2+**
- **Ground** (0 V)

You should already have one of the ***VSW*** pins connected to the ***VIN*** terminal and the Arduino Mega, and one of the ***GND*** pins connected to a ***GND*** pins on the Arduino Mega. These connections are shown in the next figure.

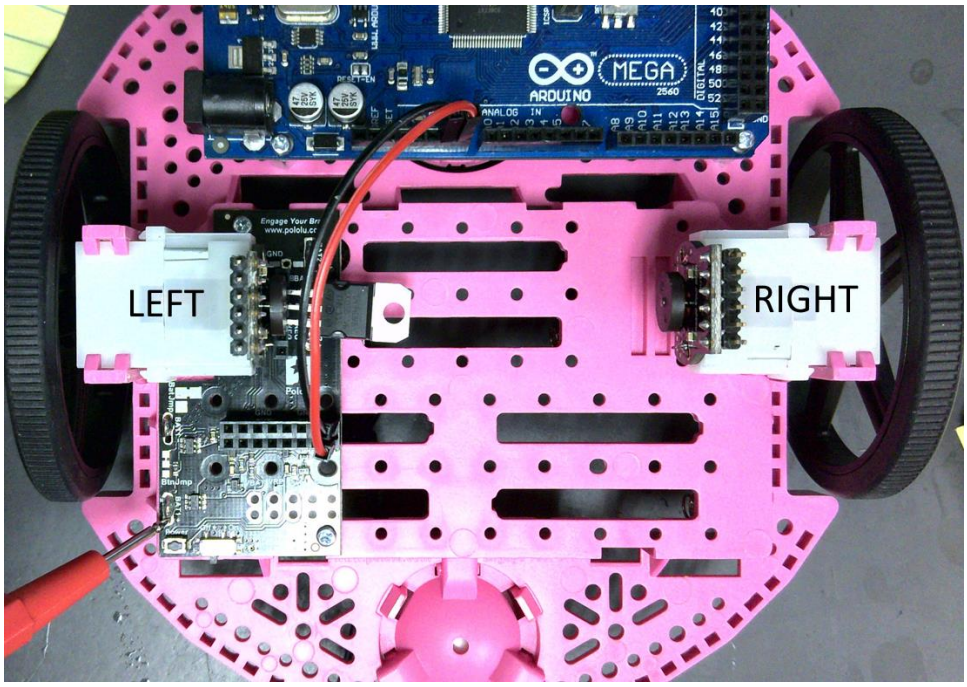


You will need to connect a ***GND*** pin of the power distribution board to the ***GND*** pin on the DRV8835. To power the motors, a connection from a ***VSW*** pin of the power distribution board must be connected to the ***VIN*** pin on the DDRV8835. For your convenience, an image of the DDRV8835 connections is shown below.



To power the driver logic, the VCC pin of the DRV8835 will need to be connected to a 5V terminal on the Arduino Mega, and the MODE pin needs to be pulled HIGH.

Now that the power connections have been established and the driver setup in the ENABLE/PHASE mode, its time to work on connecting the motors to the driver. Let's start by agreeing on the which side of the robot represents the right side, and which side represents the left side. In our configuration, we will establish the front of the robot as the side with the Arduino mounted, such that the right and left side can be discerned from the following graphic:



With the orientation established, lets use the DRV8835 connectors labelled B to control the left motor and the connectors labelled A to control the right motor. As such, connect terminals ***AOUT1*** and ***AOUT2*** to terminals ***M1*** and ***M2*** on the right motor, respectively.

Then connect the terminals ***BOUT1*** and ***BOUT2*** to terminals ***M1*** and ***M2*** on the left motor, respectively.

Now, the only remaining connections that need to be made are related to controlling the speed and direction of the motors. For these connections, connect PIN ***41*** of the Arduino to ***BPHASE***, PIN ***42*** of the Arduino to ***APHASE***, PIN ***45*** to ***BENABLE***, and PIN ***46*** to ***AENABLE***.

Connect one terminal of the push button to GND and the other terminal to PIN ***40***. Setup the button using the internal pull up resistor. As such, when the button is pressed it will be in the LOW state.

The expected operation is as follows: Applying 0V to ***BPHASE*** and 5V to ***BENABLE*** results in the left motor turning in clockwise direction at full speed. Applying 5V to ***BPHASE*** and 5V to ***BENBLE*** results in the left motor turning in counterclockwise direction at full speed. Similar behavior should be observed for the right motor.

Using these components write a program, which meets the requirements listed below:

Requirements

1. Create a sub directory named “ex04a”, and an associated Arduino sketch file named “ex04a.ino”, in your Arduino directory.
2. Create a program that includes a ***setup()*** and ***loop()*** routine. Add code to the ***setup()*** routine that opens a connection to the serial port at a baud rate of 115,200 bits/second.
2. Within the ***loop()*** routine implement the following code when the push button is depressed:
    a. Rotate the right motor in the clockwise direction at a 40% duty cycle for two seconds while the left motor is stationary.
    b. Immediately after Step 2a completes, rotate the left motor counterclockwise at a 40% duty cycle for two seconds while the right motor is stationary.

       Note: If the right motor is run clockwise and the left motor is run counterclockwise at the same time, then the robot will move forward. This is not what is asked for in Steps 2a and 2b but is something you will take advantage of in the next exercise.
    c. Immediately after Step 2b completes, rotate the right motor clockwise at a 80% duty cycle and rotate the left motor clockwise at a 80% duty cycle for two seconds.
    d. Immediately after Step 2c completes, rotate the right motor counterclockwise at a 30% duty cycle and rotate the left motor counterclockwise at a 30% duty cycle for two seconds.
    e. Immediately after Step 2D completes, implement code such that neither one of two motors rotates.

3. In your project notebook provide pertinent spec sheets and detailed pictures that show your breadboard setup, your connections to the Arduino, connections to the motor, connections to the motor driver, and connections to the Romi chassis. Clearly label connection points in your pictures. Supporting diagrams are also suggested.

4. You also need to record a short video (1 minute or less) and post the video to canvas. The video should demonstrate that when the button is pressed the robot will perform the actions outlined in Step 2a through Step 2d. Verbal narration is required.

Submission

Upload the files ex04a.ino to canvas once the exercise has been completed. The code will be evaluated as follows:

1. It will be evaluated to make sure the code is implemented using the methods outlined in this exercise.

2. Your program will be evaluated the ensure that the program successfully compiles.

3. Your program will be evaluated to ensure that it includes proper comments that explain the functionality of the source code.

4. Your program will be evaluated to ensure that it runs correctly, collects the input as described, and outputs information as described.

You are also expected to include proper documentation for this exercise in your project notebook.

EXERCISE #5 MEMORANDUM

TO: RON BROCKHOFF

FROM: KAVIAN KALANTARI

SUBJECT: EXERCISE #5 DETAILED MEMORANDUM

DATE: 10/5/21

This exercise is a build up from Exercise4. In that exercise, I was tasked with the hookups of the motors to the DRV8835 driver [1] and using a button to make the robot move in a predetermined way. In this exercise, the button is removed, and is replaced with the HM10 Bluetooth [2] module. This Bluetooth module allows for the Dabble phone application [3] to be interfaced with the robot, and allow for inputs from the GamePad to be sent to the Arduino [4]. Therefore, the movement of the robot is no longer pre-set in the code. It is now fully controllable from the phone interface (forward, backward, turn left and right).

One of the first tasks with this exercise was renaming the Bluetooth module to something non-default, to prevent other people mistaking it for their own. Most ship with a default name so reprogramming it via the Arduino serial interface was a must, especially for in-class use of the robot.

After renaming was complete, it could be seen from analyzing the Dabble interface that the gamepad has a simple Boolean state for button presses. By checking these button states, and then running an associated function for the indicated direction pressed, creation of the code was fairly trivial, and most of the motor movement code was re-used from Exercise4.

The DRV8825 motor driver is externally powered [5] and used to drive the motors [6].

Figure 1 shows the Romi Chassis [7] with all hookups, and figure 2 shows the schematic view of the components.

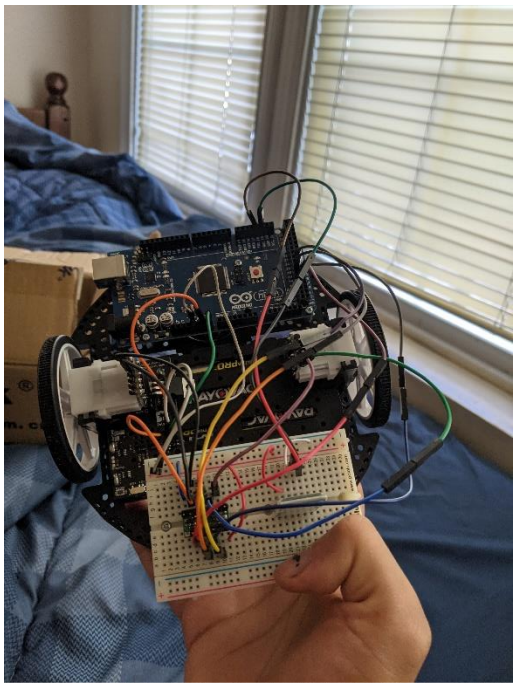


Figure 1 Assembly on Romi Chassis

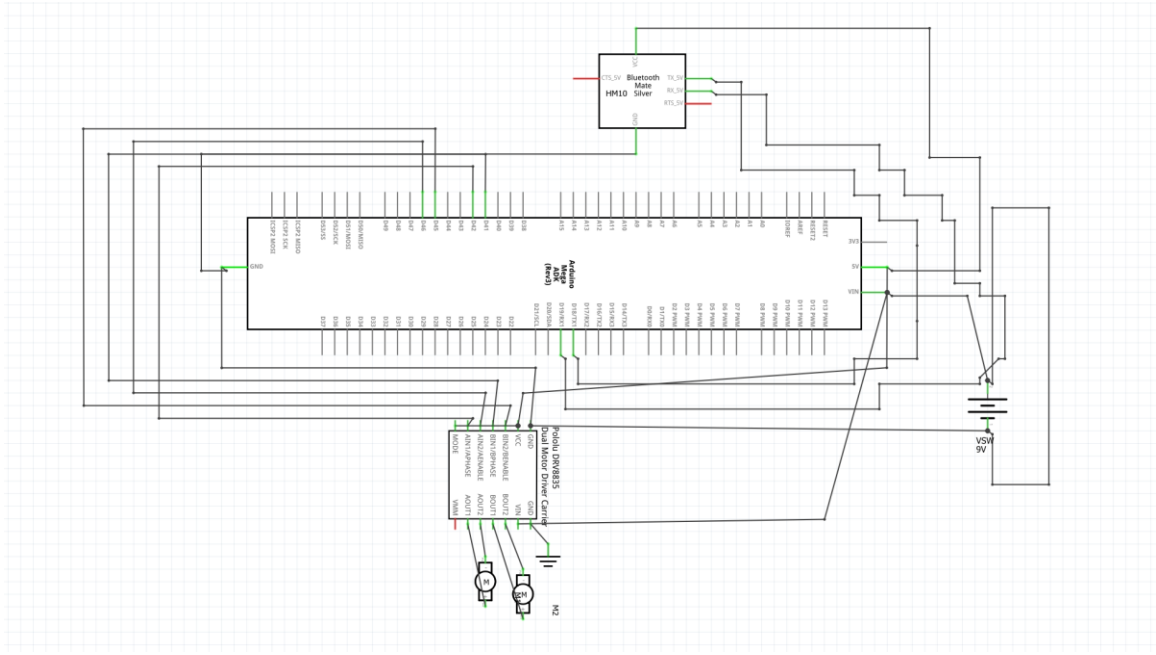


Figure 2, schematic view

Relevant Arduino Code:

```
#define CUSTOM_SETTINGS
#define INCLUDE_GAMEPAD_MODULE
#include <Dabble.h>

const int pwmPinLeft = 45;
const int pwmPinRight = 46;
const int hbPinRight = 41;
const int hbPinLeft = 42;
//creating const ints for pins used in project

float dcPercent;
//float dc255;
//used for easily setting duty cycle from percentage

void setup()
{
   Serial.begin(115200);
   //set serial baud rate

   Dabble.begin(9600);
   //set dabble baud rate

   pinMode(hbPinRight, OUTPUT);
   pinMode(hbPinLeft, OUTPUT);
   pinMode(pwmPinLeft, OUTPUT);
   pinMode(pwmPinRight, OUTPUT);
   //setting all H-Bridge pins as outputs
}

void StopMotion()
{
   //creating function to easily set all pins to low
   //and stop robot from moving
   //adding small delays boefre/after
   //because i was noticing this function didn't always work properly
   delayMicroseconds(8);
   digitalWrite(hbPinRight, LOW);
   digitalWrite(hbPinLeft, LOW);
   digitalWrite(pwmPinLeft, LOW);
   digitalWrite(pwmPinRight, LOW);
   Serial.println("All motors have been stopped.");
   delayMicroseconds(8);
}

//RIGHT CLOCKWISE
//digitalWrite(hbPinRight, HIGH);

//LEFT CLOCKWISE
//digitalWrite(hbPinLeft, HIGH);
```

```
void UpPressed()
{
   //function for moving forward
   dcPercent = 50.0;
   digitalWrite(hbPinRight, HIGH);
   analogWrite(pwmPinRight, dcPercent * 255 / 100.0);

   digitalWrite(hbPinLeft, LOW);
   analogWrite(pwmPinLeft, dcPercent * 255 / 100.0);
}

void DownPressed()
{
   //function for moving backwards
   dcPercent = 50.0;
   digitalWrite(hbPinRight, LOW);
   analogWrite(pwmPinRight, dcPercent * 255 / 100.0);

   digitalWrite(hbPinLeft, HIGH);
   analogWrite(pwmPinLeft, dcPercent * 255 / 100.0);
}

void LeftPressed()
{
   //function for turning left
   dcPercent = 50.0;
   digitalWrite(hbPinRight, LOW);
   analogWrite(pwmPinRight, dcPercent * 255 / 100.0);

   digitalWrite(hbPinLeft, LOW);
   analogWrite(pwmPinLeft, dcPercent * 255 / 100.0);
}

void RightPressed()
{
   //function for turning right
   dcPercent = 50.0;
   digitalWrite(hbPinRight, HIGH);
   analogWrite(pwmPinRight, dcPercent * 255 / 100.0);

   digitalWrite(hbPinLeft, HIGH);
   analogWrite(pwmPinLeft, dcPercent * 255 / 100.0);
}
void loop()
{
   Dabble.processInput();
   //refresh data from phone interface

   //checking button states and activating move functions accordingly
   if (GamePad.isUpPressed() == 1)
   {
```

```
        UpPressed();
        Serial.println("Forward");
    }

    else if (GamePad.isDownPressed() == 1)
    {
        DownPressed();
        Serial.println("Backwards");
    }

    else if (GamePad.isLeftPressed() == 1)
    {
        LeftPressed();
        Serial.println("Left");
    }

    else if (GamePad.isRightPressed() == 1)
    {
        RightPressed();
        Serial.println("Right");
    }

    else
    {
        Serial.println("Stopped");
        StopMotion();
    }
}
```

BIBLIOGRAPHY

[1] Polulu, "DRV8835 Dual Motor Driver Carrier," [Online]. Available: https://www.pololu.com/product/2135. [Accessed 28 September 2021].

[2] JNHuaMao Technology Company, "Bluetooth 4.0 BLE module Datasheet," [Online]. Available: ME615 Canvas Page. [Accessed 15 October 2021].

[3] Agilo Research Pvt., "Dabble: One App for Sensing & Control," [Online]. Available: https://thestempedia.com/product/dabble/. [Accessed 15 October 2021].

[4] Arduino, "Arduino MEGA 2560 & Genuino MEGA 2560," [Online]. Available: https://www.arduino.cc/en/pmwiki.php?n=Main/arduinoBoardMega2560. [Accessed 21 September 2021].

[5] Polulu, "Power Distribution Board for Romi Chassis," [Online]. Available: https://www.pololu.com/product/3541. [Accessed 22 September 2021].

[6] Polulu, "120:1 Mini Plastic Gearmotor HP, Offset 3mm D-Shaft Output, Extended Motor Shaft," [Online]. Available: https://www.pololu.com/product/1520. [Accessed 21 September 2021].

[7] Polulu, "Romi Chassis Kit - Black," [Online]. Available: https://www.pololu.com/product/3500. [Accessed 28 September 2021].

ME 615: Applications in in Mechatronics

Exercise #5

Instructions:

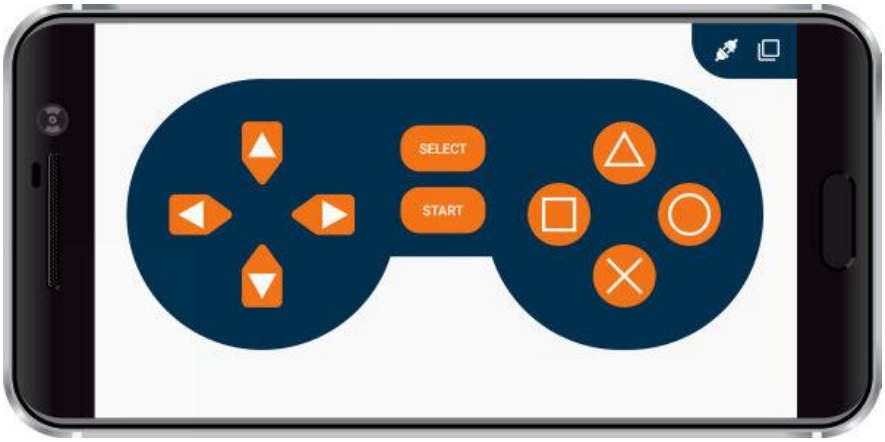
Complete each of the following steps. When you have completed all steps assigned, upload the files associated with the exercise on Canvas. Make sure the filenames you upload as part of your exercise submission have the exact names specified in each steps of this exercise. If the filenames do not match exactly, then your submission will not be graded.

Problem Statement

For this exercise you will using the DRV8835 dual motor driver carrier in unison with the power distribution board to control the right and left motors of your robot. The connections to the motor driver will be the same as used in Exercise #4. As it relates to Arduino Mega, PIN ***41*** of the Arduino should be connected to ***BPHASE***, PIN ***42*** of the Arduino should be connected to ***APHASE***, PIN ***45*** should be connected to ***BENABLE***, and PIN ***46*** should be connected to ***AENABLE***.

The expected operation is as follows: Applying 0V to ***BPHASE*** and 5V to ***BENABLE*** results in the left motor turning in clockwise direction at full speed. Applying 5V to ***BPHASE*** and 5V to ***BENBLE*** results in the left motor turning in counterclockwise direction at full speed. Similar behavior should be observed for the right motor.

You will be using the Dabble gamepad interface to control the operation of the motors for this exercise. A basic image of the Dabble gamepad interface is shown below:



To support the Dabble interface, you will need to connect the Bluetooth sensor as described in Lecture 17. In particular, you must connect the interface to the Arduino using the TX1 and RX1 connections on your Arduino Mega. Once connected, implement code to use the gamepad interface to control the motion of your robot. You application must meet the requirements listed below:

Requirements

1. Create a sub directory named “ex05a”, and an associated Arduino sketch file named “ex05a.ino”, in your Arduino directory.

2. Create a program that includes a ***setup()*** and ***loop()*** routine. Add code to the ***setup()*** routine that opens a connection to the serial port at a baud rate of 115,200 bits/second. You must also set the baud rate to 9600 bits/second for communication between the Dabble Bluetooth interface and the Bluetooth module.

2. Within the ***loop()*** routine implement the following:

    a. If the up button is pressed, then interface with your motors using the built-in motor drivers to move both motors in the forward direction using a duty cycle of 50%. To move forward, the right motor must turn clockwise, and the left motor must turn counterclockwise.

    b. If the down button is pressed, then interface with your motors using the built-in motor drivers to move both motors in the reverse direction using a duty cycle of 50%. To move in reverse, the right motor must turn counterclockwise, and the left motor must turn clockwise.

    c. If the right button is pressed, then interface with your motors using the built-in motor drivers to turn the robot right. This can be accomplished by moving the left motor in the counterclockwise direction using a duty cycle of 50%. At the same time move the right motor in the counterclockwise direction using a duty cycle of 50%.

    d. If the left button is pressed, then interface with your motors using the built-in motor drivers to turn the robot left. This can be accomplished by moving the right motor in the clockwise direction using a duty cycle of 50%. At the same time move the left motor in the clockwise direction using a duty cycle of 50%.

    e. If the up, down, right, and left buttons are not pressed, then interface with your motors to set the duty cycle to 0% for both motors. This should stop movement of the robot.

3. In your project notebook provide pertinent spec sheets and detailed pictures that show your connections to the Arduino, connections to the motors, connections to the Bluetooth modules and connections to the Romi chassis. Clearly label connection points in your pictures. Supporting diagrams are also suggested.

4. You also need to record a short video (1 minute or less) and post the video to canvas. The video should demonstrate that you can move the robot forward, left, right, and in reverse using the Dabble interface. It should also show that the robot stops when none of the buttons are pressed. Please record a video such that both the phone interface and the robot are simultaneously visible. Verbal narration is required.

Submission

Upload the file ex05a.ino and the associated video file to canvas once the exercise has been completed. The code will be evaluated as follows:

1. It will be evaluated to make sure the code is implemented using the methods outlined in this exercise.
2. Your program will be evaluated the ensure that the program successfully compiles.
3. Your program will be evaluated to ensure that it includes proper comments that explain the functionality of the source code.
4. Your program will be evaluated to ensure that it runs correctly, collects the input as described, and outputs information as described.

You are also expected to include proper documentation for this exercise in your project notebook.

EXERCISE #6 MEMORANDUM

TO: RON BROCKHOFF

FROM: KAVIAN KALANTARI

SUBJECT: EXERCISE #6 DETAILED MEMORANDUM

DATE: 10/12/21

This exercise is another build off Exercise4 and Exercise5. Previously, the DRV8825 motor driver [1] was used to drive both motors [2] on the Romi chassis [3] off of external power [4], as well as use a Bluetooth module driven [5] via the Dabble interface [6] as a way to control the robot. In this exercise, the motor's encoders are now used and are read as an input. After the robot moves 10 feet in any direction (either motor's encoder value), the motion of the robot should stop until the start button is pressed, resetting the encoder values and allowing motion again.

To properly implement the encoder values, an interrupt pin has to be made for each encoder. This interrupt pin had to be declared explicitly with a function to run, as well as what should determine the interrupt to fire (in this case the rising edge of a signal). When this rising edge is detected, the selected function runs. In all implementations of interrupts, especially on low power MCU's, it is important that the function being run is very simple. Having long winded calculation based code in this functions will slow down the performance of the Arduino [7] heavily. In our case, the function simply increments a long int (a counter) for either encoder. This allows the main functionality of the code to be run alongside the encoder functions.

Other than the encoder interrupts, the code is very similar to Exercise5. The only differences are related to resetting the encoder values to 0 when the Dabble interface detects "Start" is pressed, as well as checking the encoder values constantly to stop the robot after a certain value is reached. This value was determined mathematically, as the encoder has 360 detections for every rotation. By measuring the wheel diameter (2.75"), and knowing the formula for circumference (pi*diameter), we can ascertain that one revolution of the wheels is 8.639 inches traveled. If we divide 120" by this value and multiply by 360, we can obtain our maximum encoder value for 10 feet traveled (5000).

Many serial print statements were used here in an attempt to make sure encoders were firing correctly as well as resetting correctly. Overall the implementation wasn't very difficult.

Figure 1 shows assembly on the Romi Chassis, and figure 2 shows the associated schematic view.

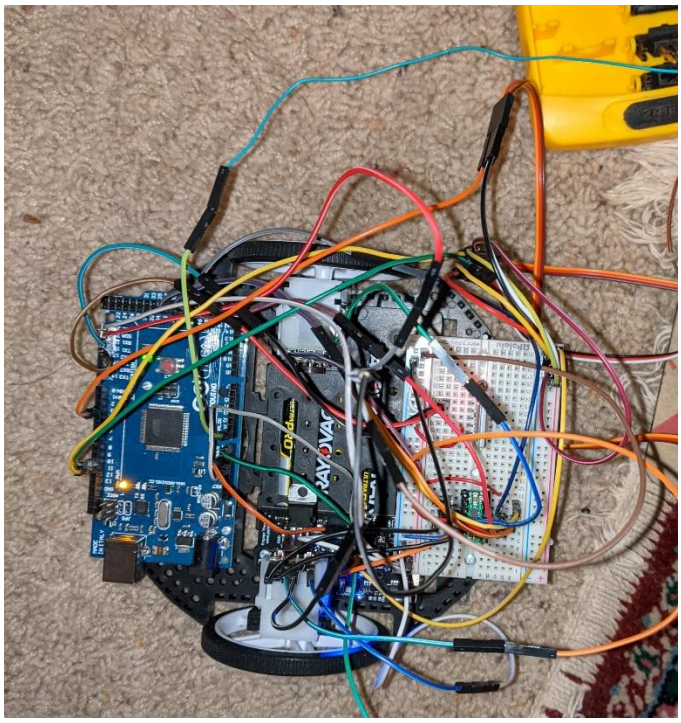


Figure 1, Assembled on Romi Chassis

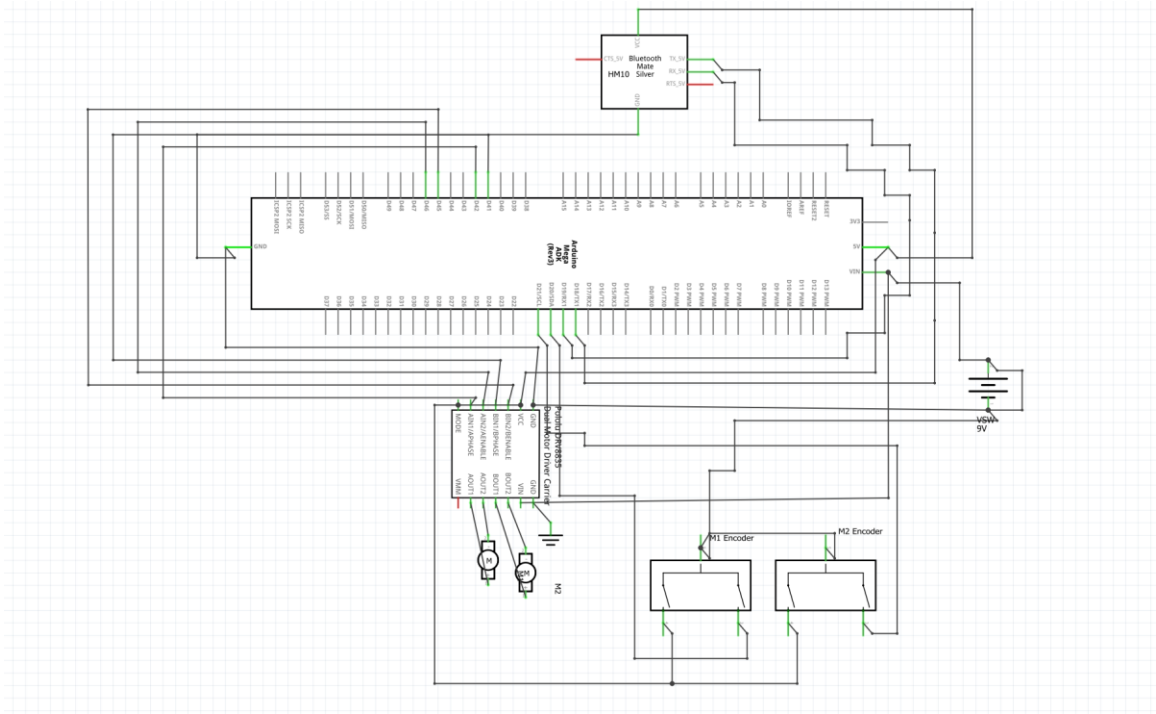


Figure 2, Schematic view

Relevant Arduino Code:

```
#define CUSTOM_SETTINGS
#define INCLUDE_GAMEPAD_MODULE
#include <Dabble.h>

const int pwmPinLeft = 45;
const int pwmPinRight = 46;
const int hbPinRight = 41;
const int hbPinLeft = 42;
const int rightEncoder = 20;
const int leftEncoder = 21;
//creating const ints for pins used in project

float dcPercent;
//float dc255;
//used for easily setting duty cycle from percentage

long int eracount;
long int elacount;
long int maxEncoderVal = 5000; //360 counts per rev, total revs for 10ft calculated to be 13.89
later in code; 360*13.89=5000
//creating global vars for encoders, era for right ela for left

bool startPressed = false;

void setup()
{
   Serial.begin(115200);
   //set serial baud rate

   Dabble.begin(9600);
   //set dabble baud rate

   pinMode(hbPinRight, OUTPUT);
   pinMode(hbPinLeft, OUTPUT);
   pinMode(pwmPinLeft, OUTPUT);
   pinMode(pwmPinRight, OUTPUT);
   //setting all H-Bridge pins as outputs

   attachInterrupt(digitalPinToInterrupt(rightEncoder), RightEncoderIncrement, RISING);
   attachInterrupt(digitalPinToInterrupt(leftEncoder), LeftEncoderIncrement, RISING);
   //creating interrupt pins for encoders
}

void LeftEncoderIncrement()
{
   elacount++;
   //every time interrupt triggered, increment encoder val
}

void RightEncoderIncrement()
```

```
{
   eracount++;
   //every time interrupt triggered, increment encoder val
}

void StartPressed()
{
   eracount = 0;
   elacount = 0;
   Serial.println("Counters reset");
   //wheel diameter is 2.75 inches. circumfrence = pi * diameter,
   //so 1 rev ~= 8.639 inches traveled
   //therefore 120 in/8.639 in = 13.89 revolutions for every 10 feet traveled
}

void PrintOutput()
{
   //Serial.println("ela=" + elacount + " " + "era=" + eracount);
   //why can't everything be like python?
   Serial.print("ela=");
   Serial.print(elacount);
   Serial.print(" era=");
   Serial.print(eracount);
   Serial.println("");
}

void StopMotion()
{
   //creating function to easily set all pins to low
   //and stop robot from moving
   //adding small delays before/after
   //because i was noticing this function didn't always work properly
   delayMicroseconds(8);
   digitalWrite(hbPinRight, LOW);
   digitalWrite(hbPinLeft, LOW);
   digitalWrite(pwmPinLeft, LOW);
   digitalWrite(pwmPinRight, LOW);
   //Serial.println("All motors have been stopped.");
   delayMicroseconds(8);
}

//RIGHT CLOCKWISE
//digitalWrite(hbPinRight, HIGH);

//LEFT CLOCKWISE
//digitalWrite(hbPinLeft, HIGH);

void UpPressed()
{
   //function for moving forward
   dcPercent = 25.0;
```

```
   digitalWrite(hbPinRight, HIGH);
   analogWrite(pwmPinRight, dcPercent * 255 / 100.0);

   digitalWrite(hbPinLeft, LOW);
   analogWrite(pwmPinLeft, dcPercent * 255 / 100.0);
}

void DownPressed()
{
   //function for moving backwards
   dcPercent = 25.0;
   digitalWrite(hbPinRight, LOW);
   analogWrite(pwmPinRight, dcPercent * 255 / 100.0);

   digitalWrite(hbPinLeft, HIGH);
   analogWrite(pwmPinLeft, dcPercent * 255 / 100.0);
}

void LeftPressed()
{
   //function for turning left
   dcPercent = 25.0;
   digitalWrite(hbPinRight, LOW);
   analogWrite(pwmPinRight, dcPercent * 255 / 100.0);

   digitalWrite(hbPinLeft, LOW);
   analogWrite(pwmPinLeft, dcPercent * 255 / 100.0);
}

void RightPressed()
{
   //function for turning right
   dcPercent = 25.0;
   digitalWrite(hbPinRight, HIGH);
   analogWrite(pwmPinRight, dcPercent * 255 / 100.0);

   digitalWrite(hbPinLeft, HIGH);
   analogWrite(pwmPinLeft, dcPercent * 255 / 100.0);
}
void loop()
{
   Dabble.processInput();
   //refresh data from phone interface

   PrintOutput();
   //printing output states of encoders

   //checking button states and activating move functions accordingly
   if (GamePad.isUpPressed() == 1 && elacount <=maxEncoderVal && eracount
<=maxEncoderVal)
   {
```

```
      UpPressed();
      Serial.println("Forward");
   }

   else if (GamePad.isDownPressed() == 1 && elacount <=maxEncoderVal && eracount
<=maxEncoderVal)
   {
      DownPressed();
      Serial.println("Backwards");
   }

   else if (GamePad.isLeftPressed() == 1 && elacount <=maxEncoderVal && eracount
<=maxEncoderVal)
   {
      LeftPressed();
      Serial.println("Left");
   }

   else if (GamePad.isRightPressed() == 1 && elacount <=maxEncoderVal && eracount
<=maxEncoderVal)
   {
      RightPressed();
      Serial.println("Right");
   }

   else if (GamePad.isStartPressed() == 1)
   {
      StartPressed();
   }

   else
   {
      StopMotion();
   }
}
```

BIBLIOGRAPHY

[1] Polulu, "DRV8835 Dual Motor Driver Carrier," [Online]. Available: https://www.pololu.com/product/2135. [Accessed 28 September 2021].

[2] Polulu, "120:1 Mini Plastic Gearmotor HP, Offset 3mm D-Shaft Output, Extended Motor Shaft," [Online]. Available: https://www.pololu.com/product/1520. [Accessed 21 September 2021].

[3] Polulu, "Romi Chassis Kit - Black," [Online]. Available: https://www.pololu.com/product/3500. [Accessed 28 September 2021].

[4] Polulu, "Power Distribution Board for Romi Chassis," [Online]. Available: https://www.pololu.com/product/3541. [Accessed 22 September 2021].

[5] JNHuaMao Technology Company, "Bluetooth 4.0 BLE module Datasheet," [Online]. Available: ME615 Canvas Page. [Accessed 15 October 2021].

[6] Agilo Research Pvt., "Dabble: One App for Sensing & Control," [Online]. Available: https://thestempedia.com/product/dabble/. [Accessed 15 October 2021].

[7] Arduino, "Arduino MEGA 2560 & Genuino MEGA 2560," [Online]. Available: https://www.arduino.cc/en/pmwiki.php?n=Main/arduinoBoardMega2560. [Accessed 21 September 2021].

ME 615: Applications in in Mechatronics

Exercise #6

Instructions:

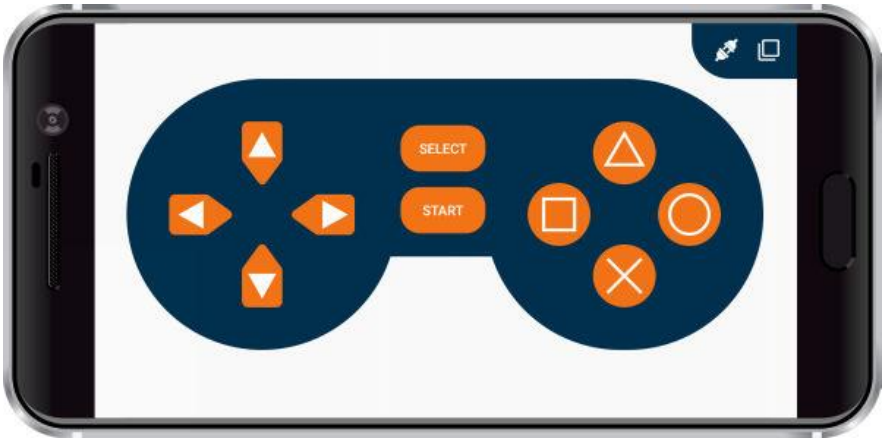
Complete each of the following steps. When you have completed all steps assigned, upload the files associated with the exercise on Canvas. Make sure the filenames you upload as part of your exercise submission have the exact names specified in each steps of this exercise. If the filenames do not match exactly, then your submission will not be graded.

Problem Statement

For this exercise you will using the DRV8835 dual motor driver carrier in unison with the power distribution board to control the right and left motors of your robot. The connections to the motor driver will be the same as used in Exercise #4. As it relates to Arduino Mega, PIN ***41*** of the Arduino should be connected to ***BPHASE***, PIN ***42*** of the Arduino should be connected to ***APHASE***, PIN ***45*** should be connected to ***BENABLE***, and PIN ***46*** should be connected to ***AENABLE***.

The expected operation is as follows: Applying 0V to ***BPHASE*** and 5V to ***BENABLE*** results in the left motor turning in clockwise direction at full speed. Applying 5V to ***BPHASE*** and 5V to ***BENBLE*** results in the left motor turning in counterclockwise direction at full speed. Similar behavior should be observed for the right motor. The outputs of the motor driver should be connected to the ***M1*** and ***M2*** terminals in the same manner as your prior exercises.

You will be using the Dabble gamepad interface to control the operation of the motors for this exercise. A basic image of the Dabble gamepad interface is shown below:



You will be connecting to the encoders on your motors for this exercise. An image of one of the motor encoders is shown below:

An encoder is attached to both the right and left motor, and you should already have the ***M1*** and ***M2*** terminals connected to the motor driver. For this exercise you will need to connect the three of the remaining connectors on both encoders. As such, connect ***VCC*** on the encoder to ***5V*** on the Arduino Mega and ***GND*** on the encoder to ***GND*** on the Arduino Mega. Then, connect ***OUT A*** on the right motor to pin ***20*** of the Arduino Mega and ***OUT A*** on the left motor to pin ***21*** on the Arduino Mega. You will setup interrupts on these pins to monitor the encoder output for the two motors.

You must implement code to use the gamepad interface to control the motion of your robot. Your application must meet the requirements listed below:

Requirements

1. Create a sub directory named “ex06a”, and an associated Arduino sketch file named “ex06a.ino”, in your Arduino directory.
2. Create a program that includes a ***setup()*** and ***loop()*** routine. Add code to the ***setup()*** routine that opens a connection to the serial port at a baud rate of 115,200 bits/second. You must also set the baud rate to 9600 bits/second for communication between the Dabble Bluetooth interface and the Bluetooth module.
2. Create two global variables to hold ***long*** integer values. One variable (i.e. ***eracount***) will keep track of the number of falling and rising edge events detected on pin ***20*** and the other variable (i.e., ***elacount***) will keep track of the number of falling edge events detected on pin ***21***.
3. Within the ***setup()*** routine setup pins ***20*** and ***21*** with an internal pull up resistor, and also setup each pin as an external interrupt. Setup pin ***20*** to trigger an ISR when a change is detected. Setup pin ***21*** to trigger an ISR when a falling edge is detected.
4. Create two ISRs that trigger as outlined in Step 3. For the ISR tied to the right motor, increment the variable ***eracount*** every time the ISR executes. For the ISR tied to the left motor, increment the variable ***elacount*** every time the ISR executes.

5. Within the ***loop()*** routine implement the following (These are similar to the requirements in the last exercise except we have reduced the speed):

    a. If the up button is pressed, then interface with your motors using the built-in motor drivers to move both motors in the forward direction using a duty cycle of 25%.

    b. If the down button is pressed, then interface with your motors using the built-in motor drivers to move both motors in the reverse direction using a duty cycle of 25%.

    c. If the right button is pressed, then interface with your motors using the built-in motor drivers to move the left motor in the forward direction using a duty cycle of 25%. At the same time move the right motor in the reverse direction using a duty cycle of 25%.

    d. If the left button is pressed, then interface with your motors using the built-in motor drivers to move the right motor in the forward direction using a duty cycle of 25%. At the same time move the left motor in the reverse direction using a duty cycle of 25%.

    e. If the up, down, right, and left buttons are not pressed, then interface with your motors to set the duty cycle to 0% for both motors. This should stop movement of the robot.

3. Modify the ***loop()*** routine as follows:

    a. When the ***start*** button is pressed in the Dabble application, reset the variables ***eracount*** and ***elacount*** to zero.

    b. Add code to output the values of ***eracount*** and ***elacount*** to the serial port in the following format:

```
ela=10165 era=5283
```

    c. Determine the circumference of the wheel along with the number of counts per revolution of the wheel. Implement code that will stop the robot if either wheel has travelled (in either direction) 10 feet. The code must also prevent the user from restarting movement unless they press the ***start*** button again. Pushing the ***start*** button will reset the counters and allow movement again until one of the wheels travels 10 feet. To test your implementation, check to see if the robot moves 10 feet before stopping when the robot is moved forward for the entire test period (i.e., you do not turn or move the robot in reverse).

4. In your project notebook provide pertinent spec sheets and detailed pictures that show your connections to the Arduino, connections to the motors, connections to the encoders, connections to the Bluetooth modules and connections to the Romi chassis. Clearly label connection points in your pictures. Supporting diagrams are also suggested.

Submission

Upload the file ex06a.ino and the associated video file to canvas once the exercise has been completed. The code will be evaluated as follows:

1. It will be evaluated to make sure the code is implemented using the methods outlined in this exercise.
2. Your program will be evaluated the ensure that the program successfully compiles.
3. Your program will be evaluated to ensure that it includes proper comments that explain the functionality of the source code.
4. Your program will be evaluated to ensure that it runs correctly, collects the input as described, and outputs information as described.

You are also expected to include proper documentation for this exercise in your project notebook.

PROJECT #1 MEMORANDUM

TO: RON BROCKHOFF

FROM: KAVIAN KALANTARI

SUBJECT: PROJECT #1 DETAILED MEMORANDUM

DATE: 10/17/21

This project was a conglomeration of both Exercise2 and Exercise6. Previously, we assembled the MeArm and mounted 4 separate servos to it (turning the arm, lifting the arm, tilt control of the arm, and gripping the arm). For this project, all servos are used except for the one for turning the arm, as we want to keep it as straight as possible. The ending goal of this project/project demonstration is to be able to pick up a small (.75") object from the ground and lift it into the air automatically after the gripper detects an object has been grabbed.
Specifically, the robot [1] should be able to move [2] fully in all directions [3]and have full control over the mounted arm such that it can raise, lower, and grip an object only through the control of the Dabble interface [4] via Bluetooth [5].

To power the servos specifically, we use a regulated 5V [6]from our battery pack. The Arduino [7] could probably handle driving at least 1 or 2 of these 3 servos on its own without using external power [8], but because its an option that is available to us it's best practice to use the Arduino as less as possible for powering any sort of semi-unstable current drawing device.
To control the duty cycle of the motors, the start and select buttons are used as +10 -10 increments respectively. The duty cycle caps out at 0 and 100, to stop the user from accidently inputting too many presses of either button. Specifically, one difficult portion of this task was registering only one press of the start/select buttons. For all the rest of the code, a simple check statement is used to determine a button press, and the associated motor code is ran (forward, back, left, right). But for this implementation, we needed to be able to discern distinct button presses as a singular press, and not a conglomeration of rapid fire presses. To do this the code stores the state of the previous press and compares it to the current state. This is for both the start and select button presses.

The lift servos move mostly smoothly by incrementing the angle they should be set by 5 and delaying for 100ms. This allows non-jerky motion and cleaner activation when the corresponding button (cross or triangle) is pressed.

To detect if an object is gripped, the feedback line of the gripper servo is read as an input. This input has a feedback of values from 0 to 1023. By scaling this feedback value, adding some room for error, and comparing it to the set position of the gripper, it can be discerned whether or not it is gripping an object.

Operation is as follows:
Cross, arm moves up
Triangle, arm moves down
Square, gripper opens

Circle, gripper closes
Start, duty cycle decreases by 10
Select, duty cycle increases by 10
Object detect in gripper, close gripper and lift arm

Figure 1 shows the associated assembly, and figure 2 shows the schematic.

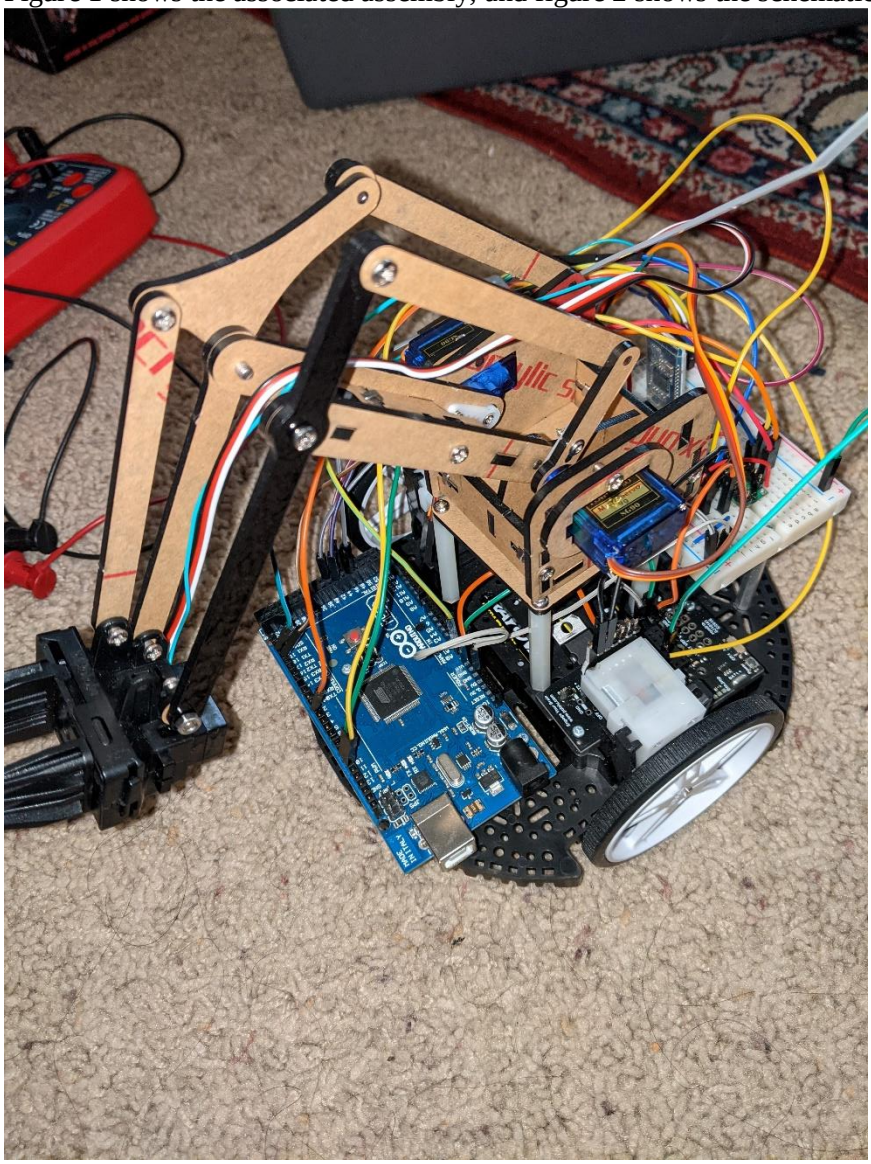


Figure 1, MeArm mounted to Romi Chassis via in class mounting system

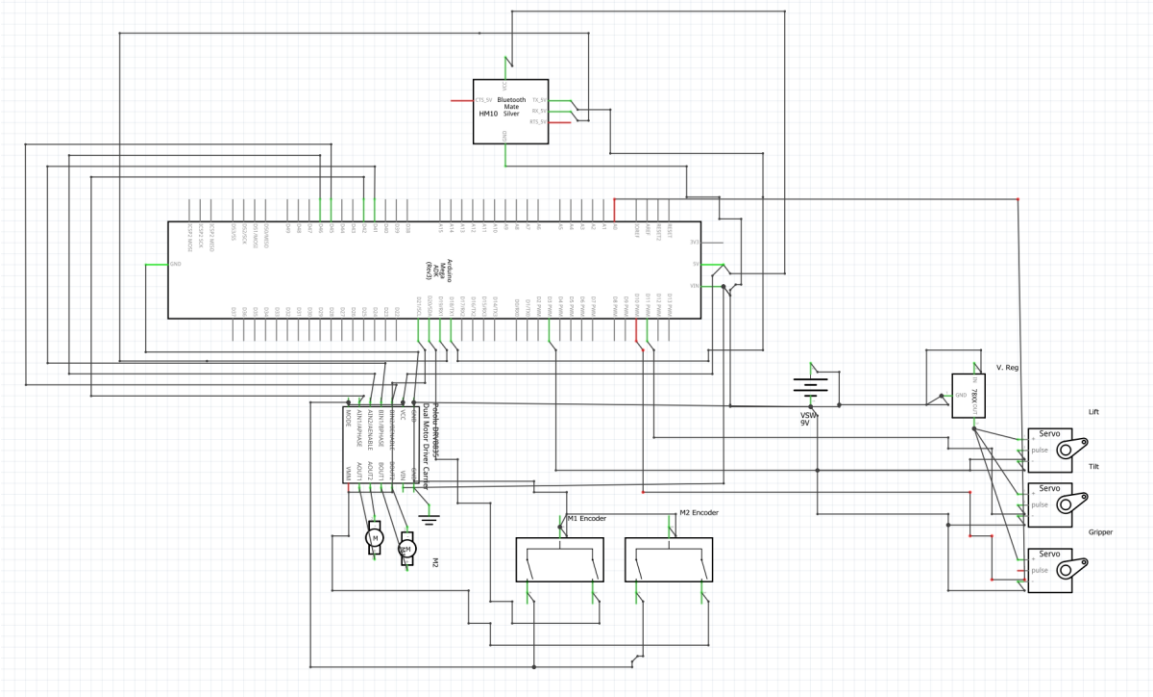


Figure 2, Schematic for all wiring

Relevant Arduino Code:

```
#define CUSTOM_SETTINGS
#define INCLUDE_GAMEPAD_MODULE
#include <Dabble.h>
#include <Servo.h>
//including libraries

Servo liftServo;
Servo tiltServo;
Servo gripperServo;
//setting up servos

const int pwmPinLeft = 45;
const int pwmPinRight = 46;
const int hbPinRight = 41;
const int hbPinLeft = 42;
const uint8_t gripperServoReader = A0;
//creating const ints for pwmPins/enable pins on left/right motor, and reading gripper analog
signal

const int liftServoPin = 3;
const int tiltServoPin = 11;
const int gripperServoPin = 10;
//creating const ints for servos used in project

float dcPercent = 50.0;
//used for easily setting duty cycle from percentage

int liftAngle = 35;
int gripperMicroSecs = 500;
//constantly changing vars

double gripperFeedback;
int gripperFeedbackMS;
int gripperFeedbackDifference;
//vars to read from gripperServo

int startState;
int lastStartState = HIGH;
int selectState;
int lastSelectState = HIGH;
//vars used for making sure select/start only read once per press/release

void setup()
{
   Serial.begin(115200);
   //set serial baud rate

   Dabble.begin(9600);
   //set dabble baud rate
```

```
    pinMode(hbPinRight, OUTPUT);
    pinMode(hbPinLeft, OUTPUT);
    pinMode(pwmPinLeft, OUTPUT);
    pinMode(pwmPinRight, OUTPUT);
    //setting all H-Bridge pins as outputs

    pinMode(gripperFeedback, INPUT);
    //reading gripperFeedback as input

    liftServo.attach(liftServoPin);
    tiltServo.attach(tiltServoPin);
    gripperServo.attach(gripperServoPin);
    //telling arduino what pins go to servos
}

void LiftArm()
//creating function that will totally lift servo arm if called
{
    while (liftAngle >= 40)
    {
        liftAngle = liftAngle - 5;
        liftServo.write(liftAngle);
        delay(100);
    }
}

//GAMEPAD NON MOVEMENT BUTTONS IMPLEMENTATION
void CrossPressed()
{
    if (liftAngle <= 160)
    //incrementing liftAngle by 5 every 100ms button is pressed
    {
        liftAngle = liftAngle + 5;
        liftServo.write(liftAngle);
        Serial.println("Lift angle: ");
        Serial.print(liftAngle);
        delay(100);
    }
}

void TrianglePressed()
{
    if (liftAngle >= 40)
    //decrementing liftAngle by 5 every 100ms button is pressed
    {
        liftAngle = liftAngle - 5;
        liftServo.write(liftAngle);
        Serial.println("Lift angle: ");
        Serial.print(liftAngle);
        delay(100);
    }
```

```
}

void SquarePressed()
{
   if (gripperMicroSecs <= 2300)
   //opening gripperServo by incrementing muSec by 100 with 100ms delay
   {
      gripperMicroSecs = gripperMicroSecs + 100;
      gripperServo.writeMicroseconds(gripperMicroSecs);
      Serial.println("Gripper Microseconds");
      Serial.print(gripperMicroSecs);
      delay(100);
   }
}

void CirclePressed()
{
   if (gripperMicroSecs >= 600)
   //closing gripperServo by decrementing muSec by 100 with 100ms delay
   {
      gripperMicroSecs = gripperMicroSecs - 100;
      gripperServo.writeMicroseconds(gripperMicroSecs);
      Serial.println("Gripper Microseconds");
      Serial.print(gripperMicroSecs);
      delay(100);
   }
}

void SelectPressed()
{
   selectState = GamePad.isSelectPressed();
   //selectState goes to 1 on press
   if (selectState != lastSelectState)  //if difference in old and new state
   {
      if (dcPercent < 100.0 && GamePad.isSelectPressed() == 1) //if dcPercent condition is met,
and button is pressed
      {
         dcPercent = dcPercent + 10.0;
         //increase dcPercent by 10;
      }

      Serial.println("DC Percent: ");
      Serial.print(dcPercent);
   }

   lastSelectState = selectState;
   //setting old state for next time button is pressed
}

void StartPressed()
{
```

```
   startState = GamePad.isStartPressed();
   //startState goes to 1 on press
   if (startState != lastStartState)  //if difference in old and new state
   {
      if (dcPercent > 0.0 && GamePad.isStartPressed() == 1)  //if dcPercent condition is met and
button is pressed
      {
         dcPercent = dcPercent - 10.0;
         //decrease dcPercent by 10;
      }

      Serial.println("DC Percent: ");
      Serial.print(dcPercent);
   }

   lastStartState = startState;
   //setting old state for next time button is pressed
}

void StopMotion()
{
   //creating function to easily set all pins to low
   //and stop robot from moving
   //adding small delays before/after
   //because i was noticing this function didn't always work properly
   delayMicroseconds(8);
   digitalWrite(hbPinRight, LOW);
   digitalWrite(hbPinLeft, LOW);
   digitalWrite(pwmPinLeft, LOW);
   digitalWrite(pwmPinRight, LOW);
   //Serial.println("All motors have been stopped.");
   delayMicroseconds(8);
}

//RIGHT CLOCKWISE
//digitalWrite(hbPinRight, HIGH);

//LEFT CLOCKWISE
//digitalWrite(hbPinLeft, HIGH);

void UpPressed()
{
   //function for moving forward
   digitalWrite(hbPinRight, HIGH);
   analogWrite(pwmPinRight, dcPercent * 255 / 100.0);

   digitalWrite(hbPinLeft, LOW);
   analogWrite(pwmPinLeft, dcPercent * 255 / 100.0);
}

void DownPressed()
```

```
{
   //function for moving backwards
   digitalWrite(hbPinRight, LOW);
   analogWrite(pwmPinRight, dcPercent * 255 / 100.0);

   digitalWrite(hbPinLeft, HIGH);
   analogWrite(pwmPinLeft, dcPercent * 255 / 100.0);
}

void LeftPressed()
{
   //function for turning left
   digitalWrite(hbPinRight, LOW);
   analogWrite(pwmPinRight, dcPercent * 255 / 100.0);

   digitalWrite(hbPinLeft, LOW);
   analogWrite(pwmPinLeft, dcPercent * 255 / 100.0);
}

void RightPressed()
{
   //function for turning right
   digitalWrite(hbPinRight, HIGH);
   analogWrite(pwmPinRight, dcPercent * 255 / 100.0);

   digitalWrite(hbPinLeft, HIGH);
   analogWrite(pwmPinLeft, dcPercent * 255 / 100.0);
}

void loop()
{
   Dabble.processInput();
   //refresh data from phone interface

   gripperFeedback = analogRead(gripperServoReader);
   gripperFeedbackMS = gripperFeedback * 5.0 * 1000.0/1024.0; //converting feedback to
microseconds value
   gripperFeedbackDifference = gripperMicroSecs - gripperFeedbackMS; //getting a difference
between set and actual value
   Serial.println("gripperFeedback: ");
   Serial.println(gripperFeedback);
   Serial.println("gripperFeedbackDifference");
   Serial.println(gripperFeedbackDifference);
   //could not totally get this section of code to work
   if(gripperFeedbackDifference >= 900 || gripperFeedbackDifference <= 900)
   {
      //LiftArm();
   }


   //checking button states and activating move functions accordingly
```

```
  StartPressed();
  SelectPressed();
  //checking button press implemented in StartPressed() and SelectPressed(), so they are
continually run

  if (liftAngle == 165)
  //part L, once lift arm is at 165 degrees tiltServo should be brought parallel
  {
    tiltServo.write(120);
    //setting arm parallel to floor
  }

  if (GamePad.isUpPressed() == 1)
  {
    UpPressed();
    Serial.println("Forward");
  }

  else if (GamePad.isDownPressed() == 1)
  {
    DownPressed();
    Serial.println("Backwards");
  }

  else if (GamePad.isLeftPressed() == 1)
  {
    LeftPressed();
    Serial.println("Left");
  }

  else if (GamePad.isRightPressed() == 1)
  {
    RightPressed();
    Serial.println("Right");
  }

  else if (GamePad.isCrossPressed() == 1)
  {
    CrossPressed();
    Serial.println("Cross");
  }

  else if (GamePad.isTrianglePressed() == 1)
  {
    TrianglePressed();
    Serial.println("Triangle");
  }

  else if (GamePad.isSquarePressed() == 1)
  {
```

```
        SquarePressed();
        Serial.println("Square");
    }

    else if (GamePad.isCirclePressed() == 1)
    {
        CirclePressed();
        Serial.println("Circle");
    }

    else
    {
        StopMotion();
    }

}
```

BIBLIOGRAPHY

[1] Polulu, "Romi Chassis Kit - Black," [Online]. Available: https://www.pololu.com/product/3500. [Accessed 28 September 2021].

[2] Polulu, "120:1 Mini Plastic Gearmotor HP, Offset 3mm D-Shaft Output, Extended Motor Shaft," [Online]. Available: https://www.pololu.com/product/1520. [Accessed 21 September 2021].

[3] Polulu, "DRV8835 Dual Motor Driver Carrier," [Online]. Available: https://www.pololu.com/product/2135. [Accessed 28 September 2021].

[4] Agilo Research Pvt., "Dabble: One App for Sensing & Control," [Online]. Available: https://thestempedia.com/product/dabble/. [Accessed 15 October 2021].

[5] JNHuaMao Technology Company, "Bluetooth 4.0 BLE module Datasheet," [Online]. Available: ME615 Canvas Page. [Accessed 15 October 2021].

[6] STMicroelectronics, "Positive Voltage Regulator IC's," [Online]. Available: ME615 Canvas Page. [Accessed 21 October 2021].

[7] Arduino, "Arduino MEGA 2560 & Genuino MEGA 2560," [Online]. Available: https://www.arduino.cc/en/pmwiki.php?n=Main/arduinoBoardMega2560. [Accessed 21 September 2021].

[8] Polulu, "Power Distribution Board for Romi Chassis," [Online]. Available: https://www.pololu.com/product/3541. [Accessed 22 September 2021].

ME 615: Applications in in Mechatronics

Project #1

Instructions:

Complete each of the following steps. When you have completed all steps assigned, upload the files associated with the exercise on Canvas. Make sure the filenames you upload as part of your exercise submission have the exact names specified in each steps of this exercise. If the filenames do not match exactly, then your submission will not be graded.

Problem Statement

For this exercise you will using the DRV8835 dual motor driver carrier in unison with the power distribution board to control the right and left motors of your robot. The connections to the motor driver will be the same as used in Exercise #4. As it relates to Arduino Mega, PIN ***41*** of the Arduino should be connected to ***BPHASE***, PIN ***42*** of the Arduino should be connected to ***APHASE***, PIN ***45*** should be connected to ***BENABLE***, and PIN ***46*** should be connected to ***AENABLE***.

The expected operation is as follows: Applying 0V to ***BPHASE*** and 5V to ***BENABLE*** results in the left motor turning in clockwise direction at full speed. Applying 5V to ***BPHASE*** and 5V to ***BENBLE*** results in the left motor turning in counterclockwise direction at full speed. Similar behavior should be observed for the right motor. The outputs of the motor driver should be connected to the ***M1*** and ***M2*** terminals in the same manner as your prior exercises.

You will also need to mount your servo arm to the chassis and connect the power for the three specified servos to the ***VREG*** pin on your power distribution board. The digital line of the lift servo (attached to right side of the arm when viewed from behind) must be attached to Arduino pin ***3***, the digital line of the wrist (i.e., tilt) servo must be attached to Arduino pin ***11***, and the digital line of the gripper servo must be attached to Arduino pin ***10***. The robotic arm must also be mounted to the romi chassis such that the base of the robotic arm is centered over the center of the chassis. ***DO NOT CONNECT THE SWING SERVOS. If you do connect it, then spurious signals can make them move at unexpected times.***

NOTE: WE ARE USING THE LIFT SERVO FOR THIS EXERCISE. IT MUST BE ADJUSTED SO THAT WHEN THE ATTACHED SERVO IS COMMAND TO 165 DEGRESS, THE ARM MOVES TO THE FULL DOWN POSITION. IF IT DOES NOT, THEN YOU NEED TO DETACT THE SERVO ARM FROM THE SERVO AND ADJUST THE POSITION. FOR REFERNCE, THE ARM MUST ALSO MOVE TO A UP POSITION WHEN COMMANDED TO AN ANGLE OF 35 DEGREES.

The following information is extremely important!! The servo library uses timer5 by default and this interferes with the PWM output on pins 44, 45, and 46. This change has already been made on the lab computers and setup for ME615 posted on canvas. However, if you are using the setup for ME400 that was deployed in a

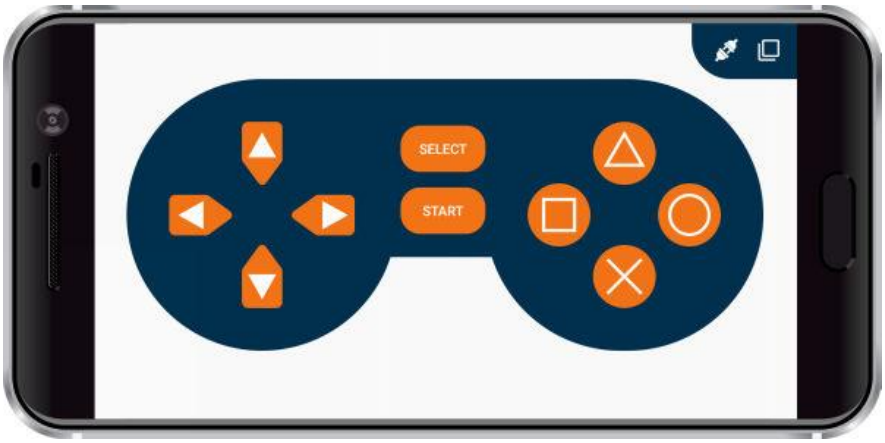
prior semester, then you will need to make the following changes. Modify the file C:\arduino-1.8.9\libraries\Servo\src\avr\ServoTimers.h and change the following:

```
#if defined(__AVR_ATmega1280__) || defined(__AVR_ATmega2560__)
#define _useTimer5
#define _useTimer1
#define _useTimer3
#define _useTimer4
typedef enum {_timer5, _timer1, _timer3, _timer4, _Nbr_16timers } timer16_Sequence_t;
```

to the following (changes displayed in red):

```
if defined(__AVR_ATmega1280__) || defined(__AVR_ATmega2560__)
//#define _useTimer5
#define _useTimer1
#define _useTimer3
#define _useTimer4
typedef enum {_timer1, _timer3, _timer4, _timer5, _Nbr_16timers } timer16_Sequence_t;
```

You will be using the Dabble gamepad interface to control the operation of the motors for this exercise. A basic image of the Dabble gamepad interface is shown below:



You must implement code to use the gamepad interface to control the motion of your robot and the robotic arm. Your application must meet the requirements listed below:

Requirements

1. Create a sub directory named “project1a”, and an associated Arduino sketch file named “project1a.ino”, in your Arduino directory.
2. Create a program that includes a ***setup()*** and ***loop()*** routine. Add code to the ***setup()*** routine that opens a connection to the serial port at a baud rate of 115,200 bits/second. You must also set the baud rate to 9600 bits/second for communication between the Dabble Bluetooth interface and the Bluetooth module.
2. In the ***setup()*** routine, set the position of the gripper servo to the fully open position, the position of the lift arm to the fully elevated position, and the wrist servo to an

angle such that the gripper is parallel to the ground when in the down position. This ensures the robotic arm starts in the same position every time the Arduino is restarted.

3. Within the ***loop()*** routine implement the following code when the push button is depressed:

    a. By default, the duty cycle should be set to 50% until it changed by the end user.

    b. If the select button is pressed increase the duty cycle of both motors by 10%. Once the duty cycle reaches 100%, additional presses of the select button should not continue to increase the duty cycle. This is harder than it may sound because each time you press the select button the ***loop()*** routine will execute many times. As such, you have to keep track of the prior state of the button and only change the value if the current state does not match the prior state.

    c. If the start button is pressed decrease the duty cycle of both motors by 10%. Once the duty cycle reaches 0%, additional presses of the start button should not continue to increase the duty cycle.

    d. If the up button is pressed, then interface with your motors using the built-in motor drivers to move both motors in the forward direction.

    e. If the down button is pressed, then interface with your motors using the built-in motor drivers to move both motors in the reverse direction.

    f. If the right button is pressed, then interface with your motors using the built-in motor drivers to move the left motor in the forward direction. At the same time move the right motor in the reverse direction

    g. If the left button is pressed, then interface with your motors using the built-in motor drivers to move the right motor in the forward. At the same time move the left motor in the reverse direction.

    h. If the up, down, right, and left buttons are not pressed, then interface with your motors to set the duty cycle to 0% for both motors. This should stop movement of the robot.

    i. If the cross is pressed, then implement code to lower the robotic arm. Do not do this in one step. The lowering of the arm should be gradual so that the arm does not suddenly jump to the down position. In general, the servo connected to the lift should move from 35 to 165 degrees in 5-degree intervals with a 100-millisecond delay between each interval. Stop movement of the servo if the cross button is released. Also stop movement once tilt servo reaches the 165-degree position.

    j. If the triangle is pressed, then implement code to raise the robotic arm. Do not do this in one step. The lift should be gradual so that the arm does not suddenly jump to the up position. In general, the servo connected to the lift should move from 165 to 35 degrees in 5-degree intervals with a 100-

millisecond delay between each interval. Stop movement of the servo if the triangle button is released. Also stop movement once lift servo reaches the 35-degree position.

k. If the triangle or cross button is pressed multiple times, then movement of the lift servo should resume at the position where movement stopped in Step i or Step j. In other words, it should allow fine tuning the positioning of the tilt by pressing and releasing the UP and DOWN buttons.

l. When the lift arm is in the lowest position of 165 degrees, adjust the wrist servo so that the gripper is parallel to the ground.

m. If the square is pressed, then implement code to close the gripper. Do not do this in one step. The closing motion should be gradual so that the arm does not suddenly jump to the closed position. Stop movement of the servo if the square button is released. Also stop movement once grip servo is fully closed.

n. If the circle is pressed, then implement code to open the gripper. Do not do this in one step. The opening motion should be gradual so that the arm does not suddenly jump to the open position. Stop movement of the servo if the circle button is released. Also stop movement once grip servo is fully open.

o. If the square or circle button is pressed multiple times, then movement of the gripper servo should resume at the position where movement stopped in Step l or Step m. In other words, it should allow fine tuning the positioning of the gripper by pressing and releasing the square and circle buttons.

p. Attach the feedback line (i.e., the green wire) of the gripper servo to Arduino Pin ***A0*** so you can determine if the gripper did not close completely. If the gripper servo did not close completely when commanded to close, then your code should automatically lift the robotic arm. This basically means the gripper is gripping an object and we want the arm to automatically lift it. The item lifted will be at least 0.75 inches in diameter.

3. In your project notebook provide pertinent spec sheets and detailed pictures that show your connections to the Arduino, connections to the motors, connections to the Bluetooth modules, connections to the servo arms, and connections to the Romi chassis. Clearly label connection points in your pictures. Supporting diagrams are also suggested.

4. You must demonstrate your code to your instructor in-person. Time slots for the demonstration will be made available after the due date for the project. You will be expected to demonstrate that the robot can navigate through a simple movement exercise, pick up an item roughly at least 0.75 inches in diameter, transport the item to an alternate location, and then place the item at a predetermined location. A competition will also be performed as it relates to this project. The details will be provided in lecture. You will also be expected to turn in your project notebook after the demonstration for review by your instructor.

Submission

Upload the file project1a.ino to canvas once the exercise has been completed. The code will be evaluated as follows:

1. It will be evaluated to make sure the code is implemented using the methods outlined in this exercise.
2. Your program will be evaluated the ensure that the program successfully compiles.
3. Your program will be evaluated to ensure that it includes proper comments that explain the functionality of the source code.
4. Your program will be evaluated to ensure that it runs correctly, collects the input as described, and outputs information as described.

You are also expected to include proper documentation for this exercise in your project notebook.

PROJECT #2 MEMORANDUM

TO: RON BROCKHOFF

FROM: KAVIAN KALANTARI

SUBJECT: PROJECT #2 DETAILED MEMORANDUM

DATE: 11/8/21

The initial setup of this project mainly revolved around altering some code that was provided to us to make our robot follow a black line on a white background. Previously, we made connections between the HM10 Bluetooth module and the Arduino, as well as connecting the motors to a motor driver. Specifically in this project, the robot [1] should be able to move [2] fully in all directions [3], be able to self-calibrate a QTR sensor array mounted on the chassis, follow a black line of varying angles and sharpness (on a white-ish background), and detect the proximity of an obstacle placed in front of it, all while communicating with the Dabble interface [4] via Bluetooth [5]. Due to issues with failure from the DRV8835 driver (probably occurring due to rapid direction changes at high velocities and a high frequency), the DRV8833 was substituted. This new driver is much less susceptible to these types of failure, but also operates in a different mode than was previously used with the DRV8835.

As with the previous driver, both motors need two Arduino pins to fully control the motor (totaling 4 pins). The difference in this case is how, now, all 4 of these pins must be PWM enabled. This is due to how the DRV8833 works in IN/IN mode, where you must set one motor pin to "HIGH" and give the other a PWM signal to adjust its speed. Passing one side of the motor a "LOW" signal results in a non-linear response where a PWM signal cannot have full linear control of the motor's output speed. Because my DRV8835 driver broke so suddenly, it took a little bit of work to transition my code to be compatible with the new driver (passing a PWM signal of 255 to one side of the motor is for having it remain stationary, whereas giving it a PWM signal of 0 is for full speed).

Operation is as follows:
User presses "Start" to initialize calibration routine
After ~10 seconds when calibration is complete, user moves robot to starting position
User presses "Cross" to initialize line tracking
At any time, user can press "Cross" again to abort line tracking
If object comes within 5cm of TOFsensor, line tracking is aborted until object is removed

Line tracking
Done via a PD controller.
For every loop, an error is calculated and used to choose and left and right motor speed to try and reach set point.
Maximum motor speed, minimum motor speed, and base speed are variables used to minimize time spent line following.
Kp and Kd are experimental values that are set by user that hopefully work well with the motors and driver setup.

The general form of a PID controller is shown below.

$$u(t) = K_p e(t) + K_i \int e(t)dt + K_p \frac{de}{dt}$$

For the purposes of a simple line follower, research online indicates that including the integral term is not needed. Therefore, the equation simplifies to:

$$u(t) = Kp * error + Kd * (error - lastError)$$

Which includes both the current and last state of the error. Kp and Kd were found through trial and experimentation to be work semi-well at values of .8 and 5, respectively (although many values produced similar results).

Mounting of QTR Sensor and TOF Sensor

Both the QTR sensor and the TOF (time of flight) sensor are mounted via a small 3D printed attachment to the front of the Romi chassis. This 3D printed piece is held onto the chassis with sticky putty. A zoomed in picture is shown in figure 1 of this mounting "situation".

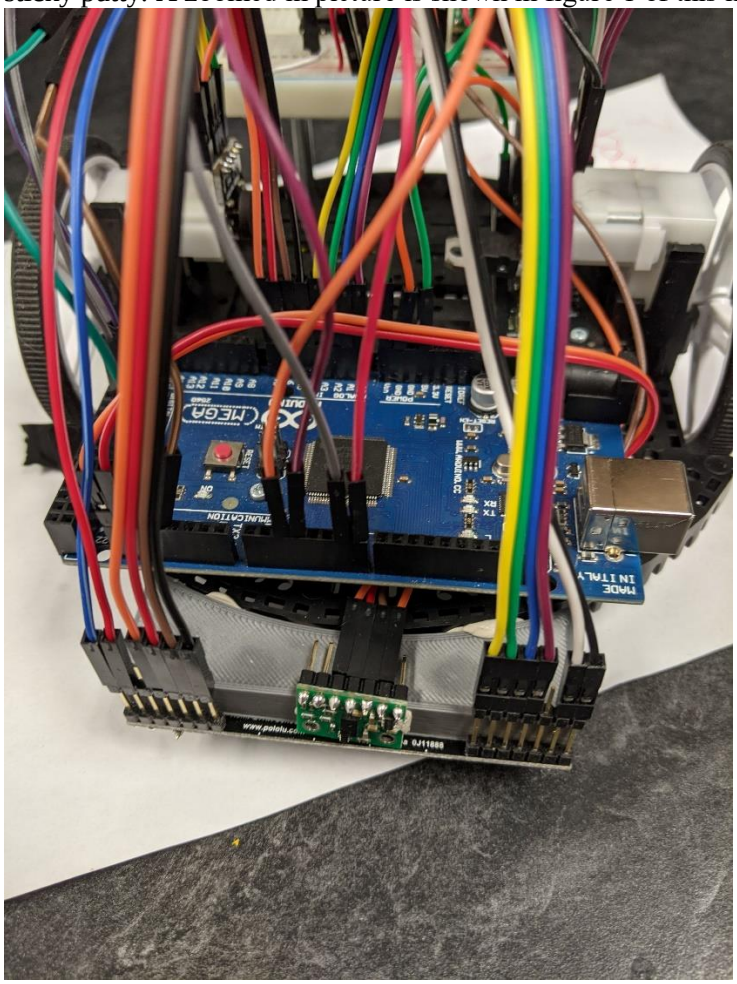


Figure 1, mounting of QTR, TOF sensors to chassis

Figure 2 shows the associated assembly, and figure 3 shows the schematic.

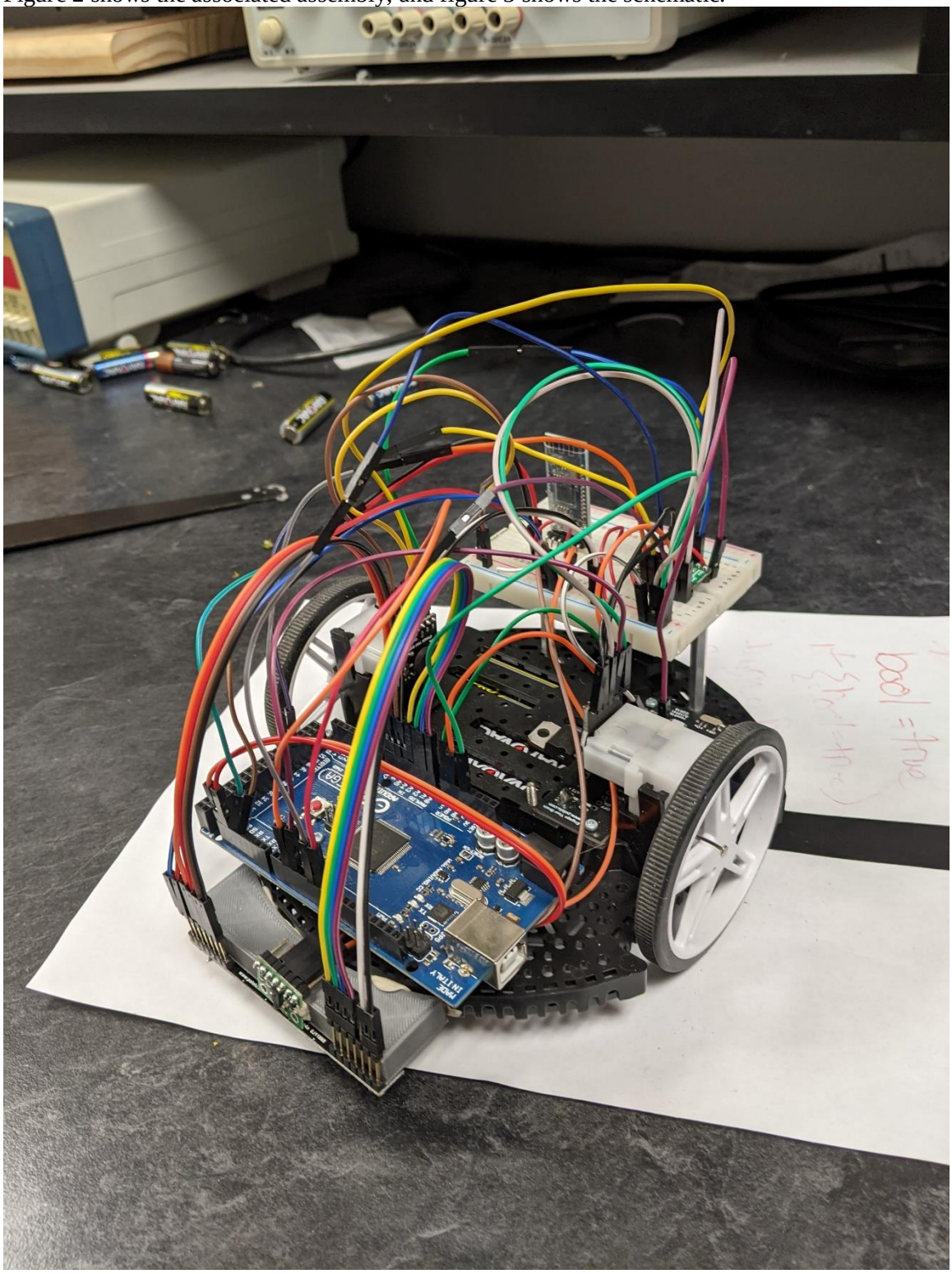


Figure 2, total robot assembly

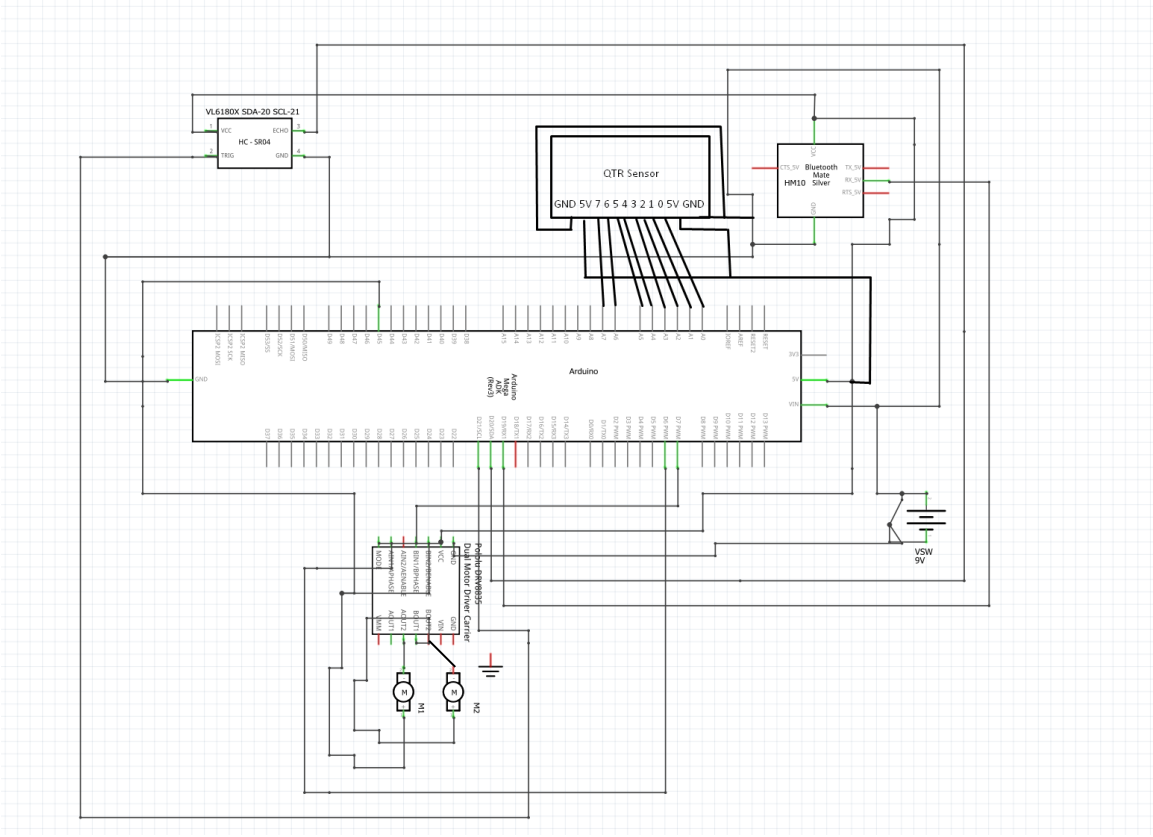


Figure 3, schematic of wiring hookups

Relevant Arduino Code:

```
#define CUSTOM_SETTINGS
#define INCLUDE_GAMEPAD_MODULE
#include <Dabble.h>
#include <QTRSensors.h>
#include <PID_v1.h>
#include <Wire.h>
#include <VL6180X.h>
//including necessary libraries for Dabble interface, QTRSensor, PID control, and TOF sensor

//
// WIRING SETUP for DRV8835 motor Driver
//
//  A1OUT to left M2
//  A2OUT to left M1
//  B1OUT to right M1
//  B2OUT to right M2
//  Arduino hookups listed below

const int rINB1 = 6;
const int rINB2 = 7;
const int lINA1 = 44;
const int lINA2 = 46;
//creating const ints for motor controller and encoder pins

//
//  This code uses with eight analog QTR sensors connected to analog pins A0 to A7.
//  Two emitter pins are specicifed by calling setEmitterINB1ins().
//
//  Calibration is setup to run for 10 seconds.  During this phase, you should expose
//  each reflectance sensor to the lightest and darkest readings they will encounter.
//  For example, if you are making a line follower, you should slide the sensors across
//  the line during the calibration phase so that each sensor can get a reading of how
//  dark the line is and how light the ground is. Improper calibration will result in
//  poor readings.
//
//  The loop() routine the calibrated sensor values and uses them to estimate the position
//  of a line. You can test this by taping a piece of 3/4" black electrical tape to a piece
//  of white paper and sliding the sensor across it.  The sensor values range from
//  0 (maximum reflectance) to 1000 (minimum reflectance) followed
//  by the estimated location of the line as a number from 0 to 5000. 1000 means
//  the line is directly under sensor 1, 2000 means directly under sensor 2,
//  etc. 0 means the line is directly under sensor 0 or was last seen by sensor
//  0 before being lost. 5000 means the line is directly under sensor 5 or was
//  last seen by sensor 5 before being lost.
//

//-----PD-----
#define Kp .8            // experimental value
```

```
#define Kd 5              // experimental value (Note: Kp < Kd)
#define maxSpeed 240      // max speed of the robot (0-255)
#define baseSpeed 165     // base speed when robot is directly on line
#define minSpeed 0        // min speed of robot
#define sharpSpeed 100    // when encountering sharp turn, this is speed wheel turns in reverse on
corresponding side
const int sharpDelay = 20; // ms delay for sharp turns
//using #define because it might be faster than using const int?
//varying opinions on internet but i like to be reckless so ill use it

int lastError = 0;
int error;
int speed;
int leftSpeed;
int rightSpeed;
bool sharpLeft = false;
bool sharpRight = false;
//creating vars to be used for PD control
//---END PD---

bool gStartMaze = false;
//global var for telling robot whether it should start navigating maze/line

QTRSensors qtr;
VL6180X sensor;
//instantiation of QTR sensor array and TOF sensor

const uint8_t SensorINB2ount = 8;
uint16_t sensorValues[SensorINB2ount];
//creating array to represent all 8 sensor values

int crossState;
int lastCrossState = HIGH;
//used for checking state of cross button

void StopMotion()
{
  //creating function to easily set all pins to low
  //and stop robot from moving
  digitalWrite(rINB1, LOW);
  digitalWrite(rINB2, LOW);
  digitalWrite(lINA2, LOW);
  digitalWrite(lINA1, LOW);
  //Serial.println("All motors have been stopped.");
  //delayMicroseconds(8);
}

void CalibrateSensorArray()
//creating function to be used when sensor array needs to be calibrated
{
  digitalWrite(LED_BUILTIN, HIGH);
```

```
  digitalWrite(lINA2, HIGH);
  digitalWrite(rINB2, HIGH);
  analogWrite(rINB1, 100);
  analogWrite(lINA1, 100);
  //turn on led and make robot move so as to calibrate with no human interaction (as is the way
engineers prefer)  :)

  // turn on Arduino's LED to indicate we are in calibration mode
  // analogRead() takes about 0.1 ms on an AVR.
  // 0.1 ms per sensor * 4 samples per sensor read (default) * 8 sensors
  // * 10 reads per calibrate() call = ~32 ms per calibrate() call.
  // Call calibrate() 400 times to make calibration take about 13 seconds.
  for (uint16_t i = 0; i < 400; i++)
  {
    qtr.calibrate();
  }

  digitalWrite(LED_BUILTIN, LOW); // turn off Arduino's LED to indicate we are through with
calibration

  StopMotion();

  for (uint8_t i = 0; i < SensorINB2ount; i++)
  {
    Serial.print(qtr.calibrationOn.minimum[i]);
    Serial.print(' ');
  }
  Serial.println();

  // print the calibration maximum values measured when emitters were on
  for (uint8_t i = 0; i < SensorINB2ount; i++)
  {
    Serial.print(qtr.calibrationOn.maximum[i]);
    Serial.print(' ');
  }
  Serial.println();
}

void CrossPressed()
{
  crossState = GamePad.isCrossPressed();
  //crossState goes to 1 on press
  if (crossState != lastCrossState) //if difference in old and new state
  {
    if (GamePad.isCrossPressed() == 1)
    {
      Serial.println("state flip");
      gStartMaze = !gStartMaze;
      //flip state of gStartMaze
    }
```

```
    if (gStartMaze == false)
    {
      StopMotion();
      //if after flip gStartMaze is false, stop robot motion
    }
  }

  lastCrossState = crossState;
  //setting old state for next time button is pressed
}

void setup()
{
  pinMode(rINB1, OUTPUT);
  pinMode(rINB2, OUTPUT);
  pinMode(lINA2, OUTPUT);
  pinMode(lINA1, OUTPUT);
  //setting all motor driver pins as outputs

  pinMode(LED_BUILTIN, OUTPUT);
  //using built in LED to tell when sensor is calibrating

  StopMotion();
  //make sure robot is stopped

  qtr.setTypeRC();
  qtr.setSensorPins((const uint8_t[]){A0, A1, A2, A3, A4, A5, A6, A7}, SensorINB2ount);
  //setup qtr sensor
  //setting A0-A7 to be inputs on sensor array

  Serial.begin(115200);
  Dabble.begin(9600);
  //set serial and dabble baud rate

  Wire.begin();  //loads required settings onto VL6180X to init sensor
  sensor.init(); //init Wire library and connects i2c bus
  sensor.configureDefault();
  sensor.setTimeout(100); //adds timeout value in ms, where read function will abort
  //required commands for instantiation of TOF sensor

  /*
  analogWrite(rINB1, 10);
  digitalWrite(rINB2, HIGH);
  Serial.println("right forward 50?");
  delay(1000);
  StopMotion();

  Serial.println("inbetween");

  analogWrite(lINA2, 10);
  digitalWrite(lINA1, HIGH);
```

```
  Serial.println("left forward 50?");
  delay(1000);
  StopMotion();
 */
  //testing motor outputs with new 8833 driver...hopefully more reliable than 8835
}

void loop()
{
  Dabble.processInput();
  //get inputs from dabble interface

  CrossPressed();
  //always checking status of Cross button

  if (GamePad.isStartPressed() == 1)
  {
    CalibrateSensorArray();
    //run calibration if Start is pressed
  }

  if (gStartMaze == true && sensor.readRangeSingleMillimeters() >= 50)
  //if gStartMaze is set true from pressing Cross, and robot doesn't detect object within 5cm, then
start line following
  {
    // read calibrated sensor values and obtain a measure of the line position
    // from 0 to 7000 (for a white line, use readLineWhite() instead)
    double position = qtr.readLineBlack(sensorValues);

    Serial.print("P=");
    Serial.println(position);

    error = position - 3500;
    speed = Kp * error + Kd * (error - lastError);
    lastError = error;
    //using PD control equation to create equation for "speed", which is applied to each motor in
opposite direction
    // to try and reach set point of qtr sensor reading 3500

    rightSpeed = baseSpeed - speed;
    leftSpeed = baseSpeed + speed;
    //using speed var to set individual left and right motor speeds

    if (rightSpeed > maxSpeed)
    {
      rightSpeed = maxSpeed;
    }

    if (leftSpeed > maxSpeed)
    {
      leftSpeed = maxSpeed;
```

```
  }
  //setting max limits on motor speeds

  if (rightSpeed < minSpeed)
  {
   rightSpeed = minSpeed;
   //sharpRight = true;
  }

  if (leftSpeed < minSpeed)
  {
   leftSpeed = minSpeed;
   //sharpLeft = true;
  }
  //setting min limits on motor speeds. was experimenting with corner handling but didn't get it
to work

  if (sharpLeft == false && sharpRight == false)
  {
   digitalWrite(lINA2, HIGH);
   analogWrite(lINA1, 255 - leftSpeed);
   Serial.println("left speed: ");
   Serial.print(255 - leftSpeed);
   Serial.println("");
   //writing speed to left motor

   digitalWrite(rINB1, HIGH);
   analogWrite(rINB2, 255 - rightSpeed);
   Serial.println("right speed: ");
   Serial.print(255 - rightSpeed);
   Serial.println("");
   //writing speed to right motor
  }
  /*
  else if (sharpLeft == true)
  {
   digitalWrite(lINA1, HIGH);
   analogWrite(lINA2, 255 - sharpSpeed);
   //reverse left wheel

   digitalWrite(rINB1, HIGH);
   analogWrite(rINB2, 255 - rightSpeed);
   delay(sharpDelay);

   sharpLeft = false;
  }

  else
  {
   digitalWrite(lINA2, HIGH);
   analogWrite(lINA1, 255 - leftSpeed);
```

```
    //reverse right wheel

    digitalWrite(rINB2, HIGH);
    analogWrite(rINB1, 255 - sharpSpeed);
    delay(sharpDelay);

    sharpRight = false;
   }
    */

  //sharp corner experimentation
 }

 else
 {
  StopMotion();
  Serial.println("motion stopped");
  //making sure robot stops moving when TOFsensor reads object within 5cm
 }
}
```

BIBLIOGRAPHY

[1] Polulu, "Romi Chassis Kit - Black," [Online]. Available: https://www.pololu.com/product/3500. [Accessed 28 September 2021].

[2] Polulu, "120:1 Mini Plastic Gearmotor HP, Offset 3mm D-Shaft Output, Extended Motor Shaft," [Online]. Available: https://www.pololu.com/product/1520. [Accessed 21 September 2021].

[3] Polulu, "DRV8835 Dual Motor Driver Carrier," [Online]. Available: https://www.pololu.com/product/2135. [Accessed 28 September 2021].

[4] Agilo Research Pvt., "Dabble: One App for Sensing & Control," [Online]. Available: https://thestempedia.com/product/dabble/. [Accessed 15 October 2021].

[5] JNHuaMao Technology Company, "Bluetooth 4.0 BLE module Datasheet," [Online]. Available: ME615 Canvas Page. [Accessed 15 October 2021].

[6] STMicroelectronics, "Positive Voltage Regulator IC's," [Online]. Available: ME615 Canvas Page. [Accessed 21 October 2021].

[7] Arduino, "Arduino MEGA 2560 & Genuino MEGA 2560," [Online]. Available: https://www.arduino.cc/en/pmwiki.php?n=Main/arduinoBoardMega2560. [Accessed 21 September 2021].

[8] Polulu, "Power Distribution Board for Romi Chassis," [Online]. Available: https://www.pololu.com/product/3541. [Accessed 22 September 2021].

ME 615: Applications in in Mechatronics

Project #2

Instructions:

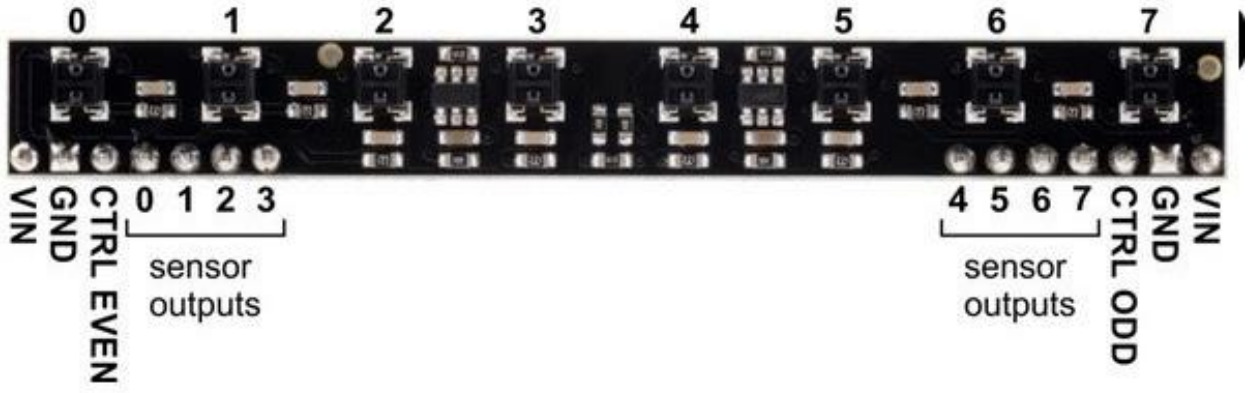
Complete each of the following steps. When you have completed all steps assigned, upload the files associated with the project on Canvas. Make sure the filenames you upload as part of your exercise submission have the exact names specified in each steps of this exercise. If the filenames do not match exactly, then your submission will not be graded.

Problem Statement

For this exercise you will using the DRV8835 dual motor driver carrier in unison with the power distribution board to control the right and left motors of your robot. The connections to the motor driver will be the same as used in Exercise #4. As it relates to Arduino Mega, PIN ***41*** of the Arduino should be connected to ***BPHASE***, PIN ***42*** of the Arduino should be connected to ***APHASE***, PIN ***45*** should be connected to ***BENABLE***, and PIN ***46*** should be connected to ***AENABLE***.

The expected operation is as follows: Applying 0V to ***BPHASE*** and 5V to ***BENABLE*** results in the right motor turning in clockwise direction at full speed. Applying 5V to ***BPHASE*** and 5V to ***BENBLE*** results in the right motor turning in counterclockwise direction at full speed. Similar behavior should be observed for the right motor. The outputs of the motor driver should be connected to the ***M1*** and ***M2*** terminals in the same manner as your prior exercises.

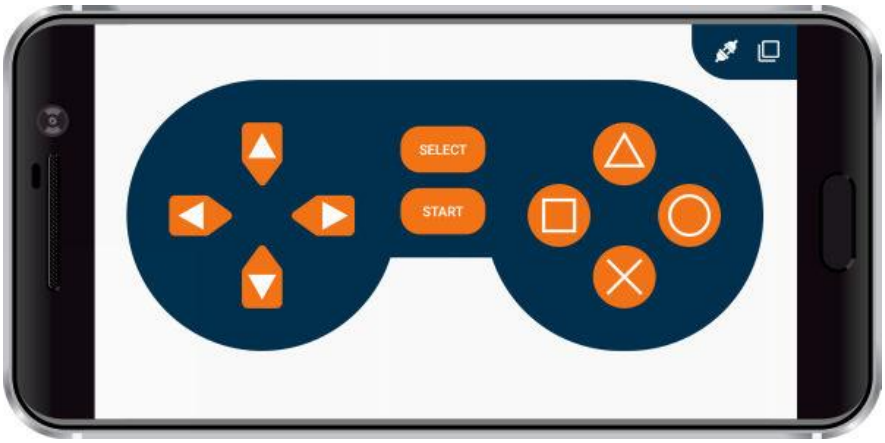
Your lab kit also includes a reflectance sensor. Connect pins ***0*** through ***7*** on the sensor to pins ***A0*** though ***A7*** of the Arduino Mega. Connect the ***VIN*** pins to the ***5V*** source on the Arduino Mega, and the ***GND*** pin to the ***GND*** pin on the Arduino Mega.



You also need to mount the VL6180X Time-of-Flight Distance Sensor to the front of the robot so that it can detect if a physical obstacle is placed in front of the robot. The sensor must be positioned as close to the front of the robot as possible, and must be mounted perpendicular to the driving surface. Once mounted connect the ***SDA*** connector of the sensor to pin ***20*** of the Arduino Mega, the ***SCL*** connector of the color sensor to pin ***21*** of

the Arduino Mega, the ***VIN*** connector of the color sensor to a ***5V*** source on the Arduino Mega, and the GND connector of the color sensor to a GND source on the Arduino Mega.

You will be using the Dabble gamepad interface to instruct the robot to start navigating a line follower maze and to implement an emergency stop if the robot stays outside the bounds of the maze. A basic image of the Dabble gamepad interface is shown below:



Note: You may remove the robotic arm from the robot for this project. I will not be used for this project, and it may be cumbersome to navigate the robot with the robotic arm attached.

You must implement code to use the gamepad interface to control the motion of your robot and the robotic arm. Your application must meet the requirements listed below:

Requirements

1. Create a sub directory named “project2a”, and an associated Arduino sketch file named “project2a.ino”, in your Arduino directory.
2. Create a program that includes a ***setup()*** and ***loop()*** routine. Set the baud rate to 9600 bits/second for communication between the Dabble Bluetooth interface and the Bluetooth module.
2. Within the ***loop()*** routine implement the following:
    a. If the select button is pressed within the Dabble interface start a routine that calibrates the QTRX sensor array. This calibration should take between 10 and 20 seconds and you should move the sensor array of the robot over a strip of black tape on a white background during the calibration cycle. To help you identify the start and end of the calibration cycle, illuminate the built-in LED attached to pin 13 of the Arduino Mega during the calibration. Turn off the LED when the calibration when the calibration completes.
    b. If the start button is pressed within the Dabble interface set a global variable to true that indicates the robot is ready to start navigating the maze.

c. If the cross button is pressed within the Dabble interface, then the robot should immediately stop and reset the global variable to false that indicates the robot should start navigating the maze.

d. If the global variable is set to true that indicates the robot is ready to start navigating the maze. This does not mean navigation should start as soon as the start button is pressed. It simply means the robot should start monitoring the time of flight sensor. Maze navigation should not begin unless the time of flight sensor does not detect an obstacle in front of the robot. Once navigation begins the robot should start following the black line. Your robot should continue to follow the black line based on the following criteria:

    i. If an obstacle is not detected within 5 centimeters of the front of the robot, then start navigating the maze.

    ii. The line course will be constructed using black electrical tape. The robot should maintain position over the line over the entire course. Some portions of the course will include angles of greater of 90 degrees. As such, special handling may need to be implemented to ensure the robot continues in the forward direction when navigating these portions of the course.

    iii. The robot should stop line navigation if an obstacle is detected within 5 centimeters of the front of the robot, and continue navigation if the obstacle is removed.

    iv. The robot should also stop line navigation if the cross button is pressed on the Dabble interface, and restart navigation if the start button is depressed.

    v. The code provided in lecture may not be optimal for line navigation. You are expected to consider alterations to the finite state machine to make it track more reliably and efficiently. Submissions will be timed with a stopwatch and ranked from fastest to slowest.

3. In your project notebook provide pertinent spec sheets and detailed pictures that show your connections to the Arduino, connections to the motors, connections to the Bluetooth modules, connections to the color sensor, connections to the reflectance array, and connections to the Romi chassis. Clearly label connection points in your pictures. Supporting diagrams are also suggested.

4. You must demonstrate your code to your instructor in-person. Time slots for the demonstration will be made available after the due date for the project. You will be expected to demonstrate that the robot can navigate through a simple line maze that includes straightaways, turns, and stop conditions. The speed at which your robot completes the maze will be considered in your evaluation as well as its ability to deal with different maze conditions. Note: You will also be expected to turn in your project notebook after the demonstration for review by your instructor.

Submission

Upload the file project2a.ino to canvas once the exercise has been completed. The code will be evaluated as follows:

1. It will be evaluated to make sure the code is implemented using the methods outlined in this exercise.
2. Your program will be evaluated the ensure that the program successfully compiles.
3. Your program will be evaluated to ensure that it includes proper comments that explain the functionality of the source code.
4. Your program will be evaluated to ensure that it runs correctly, collects the input as described, and outputs information as described.

You are also expected to include proper documentation for this project in your project notebook.