Jon Grote

Dr. Brockhoff

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 two DRV8838 on the power distribution board to control the right and left motors of your robot. Locate the 6-pin header you soldered to the power distribution board. You will be connecting the associated DIR and PWM pins to pins on your Arduino Mega. An image of the connectors is shown below.

For simplicity I will refer to the DIR and PWM pins for the left motors as LDIR and LPWM. I will refer to the DIR and PWM for the right motor as RDIR and RPWM. As such, Connect PIN ***41*** of the Arduino to ***LDIR***, PIN ***42*** of the Arduino to ***RDIR***, PIN ***45*** to ***LPWM***, and PIN 46 to ***RPWM***.

The expected operation is as follows: Applying 5V to ***LDIR*** and 5V to ***LPWM*** results in the left motor turning in forward direction at full speed. Applying 5V to ***RDIR*** and 5V to ***RPWM*** results in the right motor turning in forward direction at full speed.

You will also need to mount your servo arm to the chassis and connect the power for all three servos to the ***VREG*** pin on your power distribution board. The digital line of the lift servo must be attached to Arduino pin ***3***, the digital line of the wrist servo must be attached to Arduino pin ***11***, and the digital line of the gripper servo must be attached to Arduino pin ***10***. **The following information is extremely important!! The servo library uses**

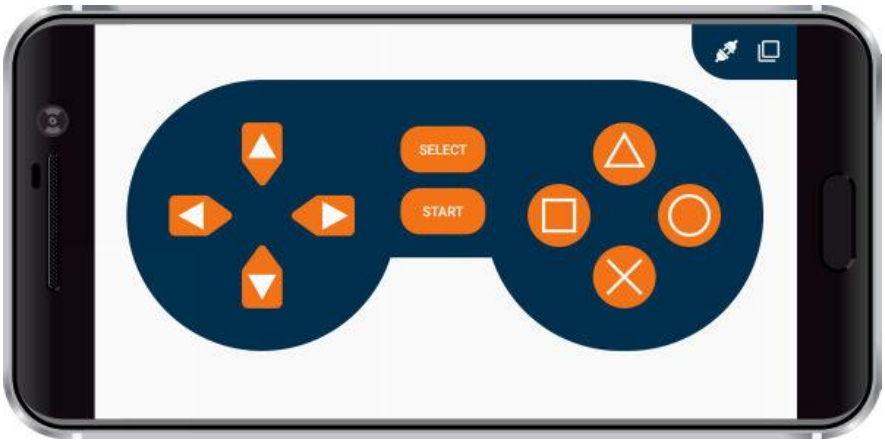
timer5 by default and this interferes with the PWM output on pins 44, 45, and 46. Modify the file C:\arduino-1.8.9\libraries\Servo\src\avr\ServoTimers.h and change the following:

```
#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. 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%.

    c. If the start button is pressed decrease the duty cycle of both motors by 10%

    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 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.

    j. 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.

    k. 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.

    l. 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.

    m. 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.

    n. The wrist servo should be commanded to a position that is roughly parallel to the ground when the robotic arm is the down position.

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 a item roughly the size of a single 6 side die, transport the item to an alternate location, and then place the die at a predetermined location. 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.

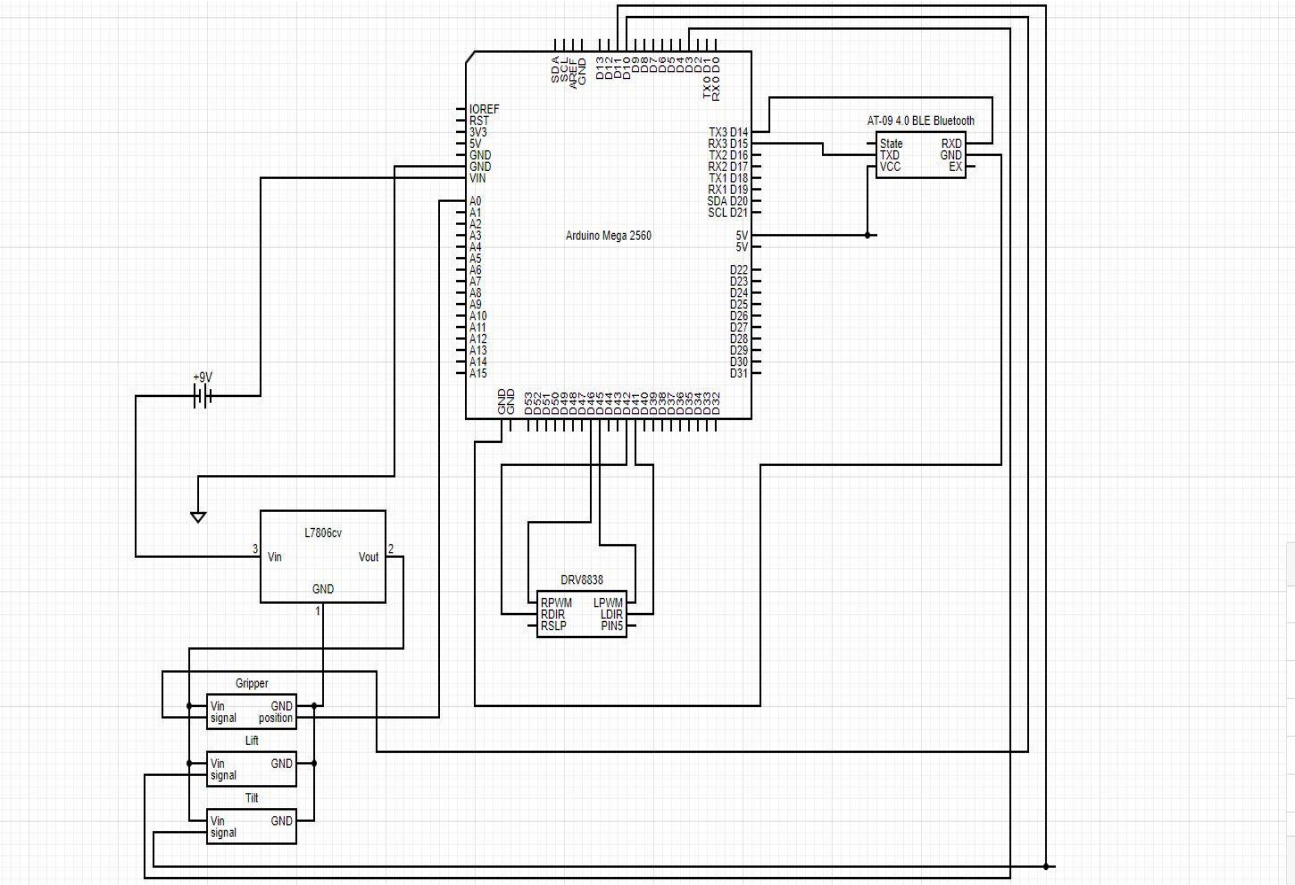
2. Your program will be evaluated the ensure that the program successfully compiles.

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

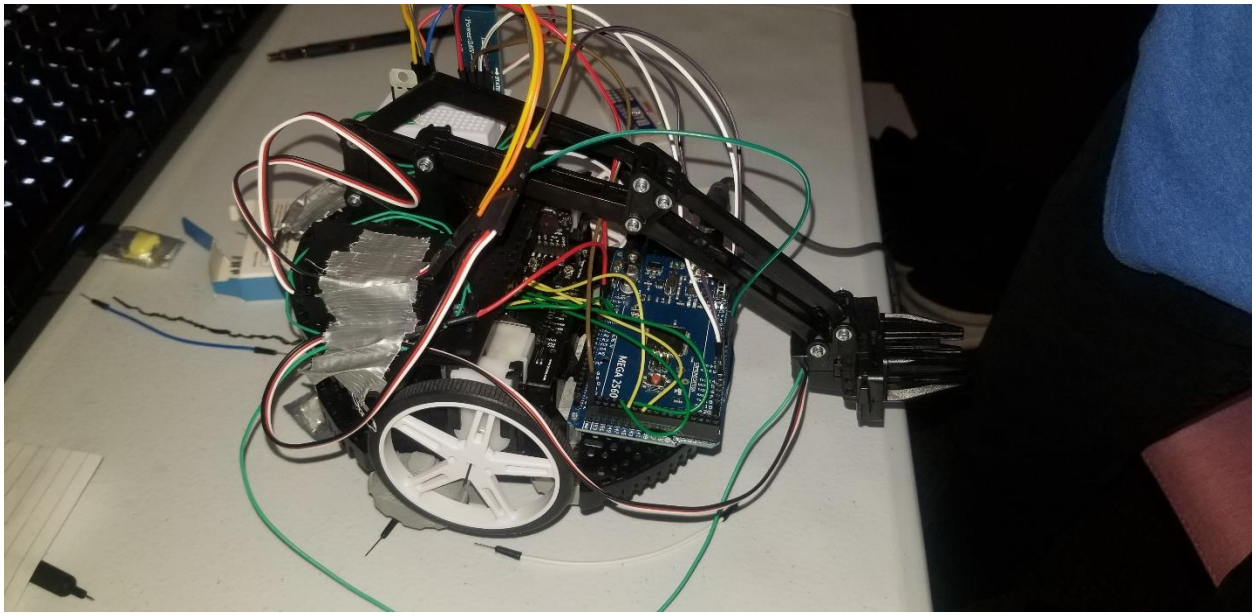
4. Your program will be evaluated to ensure that it runs correctly, collects the input as described, and outputs information as described.

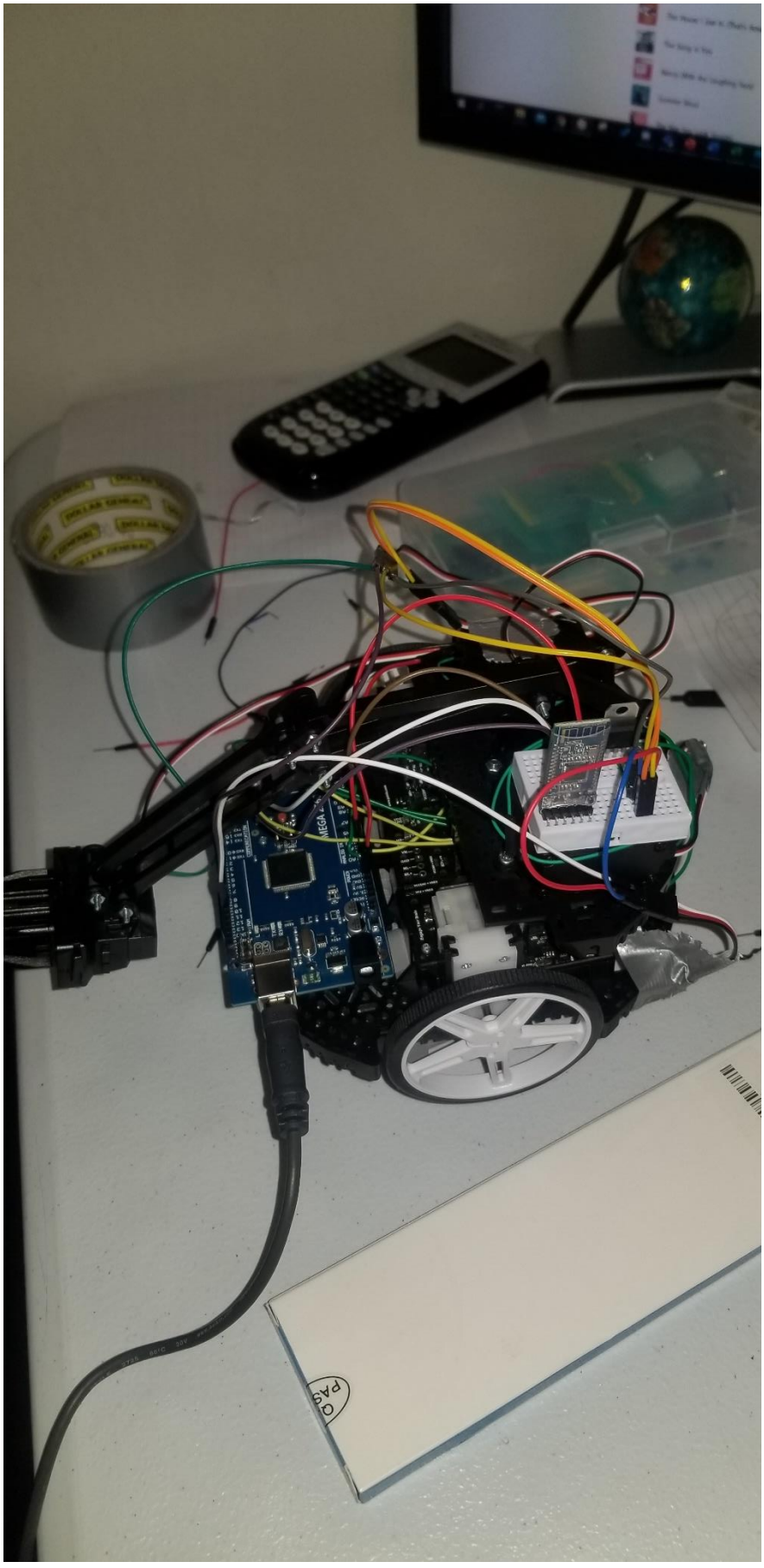
You are also expected to include proper documentation for this exercise in your project notebook.

MY CIRCUIT DIAGRAM:


Arduino Mega 2560
AT-09 4.0 BLE Bluetooth
State
TXD
VCC
RXD
GND
EX
L7806cv
Vin
Vout
GND
+9V
DRV8838
RPWM
RDIR
RSLP
LPWM
LDIR
PINS
Gripper
Vin
signal
GND
position
Lift
Tilt

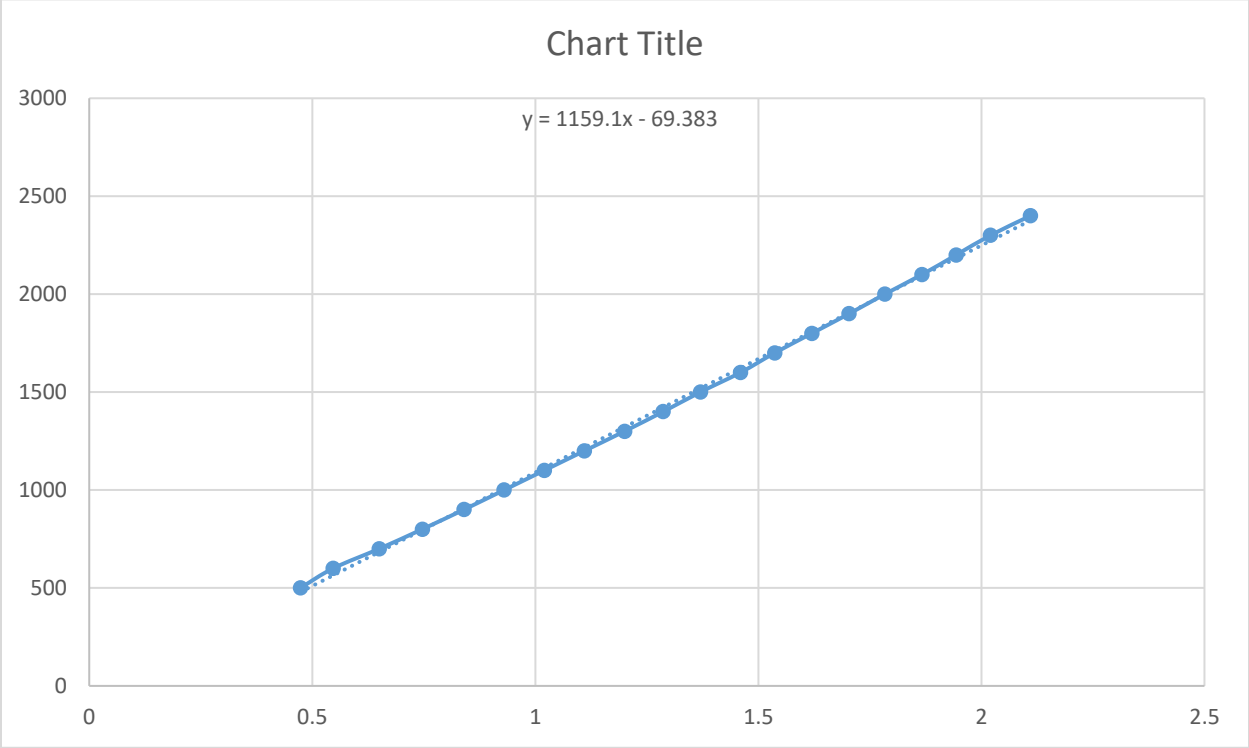
PICTURES:





REPORT:

My goal with the project was to accomplish all tasks efficiently without issues. Some parts were not so elegant, but they were effective. The first major deviation was the calculated position element located in the gripper close functionality. By using excel, I was able to map the calculated positions vs the known (written) positions by equating them through the analog position read at pin A0. The equation line was given as y = 1159.1x – 69.383, where y = calculated position and x = analog position. Y could then be compared to the written value, and during the close function, if these values are ever too distant, the gripper has “gripped”, triggering the arm to be raised via a separate function.



The second deviation was just an issue that had to be circumvented. I didn’t expect the select and start functionality for duty cycle alteration to be do troublesome, yet it was. Any “tap” of either button sends ~8-12 signals of the button being pressed (true). Thus a tap, which was meant to raised the duty cycle just 10%, raised it 100%. Or something along those lines. Thus a debounce delay was used for them to ensure only a single press was registered as significant, while the others within the delay time would be ignored.

The entirety of the wheel implementation was copied over from my code in exercise 6. The voltage regulator performed well, allowing sufficient voltage and current to all 3 servos and the wheel motors.