ME 491
November 2017
v: June 27, 2019

Joystick Control of DC Motors
Gerald Recktenwald
gerry@pdx.edu

Introduction

This document describes the use of a joystick as input to a motor controller board. Figure 1 shows the block diagram of the system. A joystick with x-y motion and a momentary button provide input to an Arduino (or similar) microcontroller. The microcontroller supplies logic signals and power for the logic circuits of a TB6612 dual half-bridge motor controller. The output of the half-bridge is a PWM signal that controls the speed and direction of two DC motors.

Power from the motors comes from a separate supply (`Vm`) with a common ground shared by the Arduino. The TB6612 controller can work with supply voltages in the range $4.5 \leq$ `Vm` ≤ 13.5 VDC. For small motors, the $+5V$ supply of the Arduino may be sufficient for `Vm`, so that an external supply may not be necessary. As a demonstration for ME 491, two DC motors are powered from the $+5V$ supply on an Arduino Uno board.

The emphasis of this document is on the translation of the analog signals from the joystick to the speed and direction of the two DC motors. The TB6612 motor controller is described only in terms of the logic pins necessary to affect the desired response of the motors. The logic signals to the TB6612 are managed by a small library (MMEmotor) that simplifies the Arduino code. A sample Arduino program is presented that demonstrates how the block diagram in Figure 1 can be implemented.

A TB6612 can also be configured to drive a single stepper motor. Refer to the tutorial on the Adafruit web site[1] for a wiring guide and sample code using the Arduino Stepper library[2]. The stepper motor configuration is not discussed in this document.

The system described in this paper uses the operator of the joystick as the feedback control system. In other words, the software running on the Arduino is just an open-loop control system. To implement a feedback control system some source of measurement error, such as the desired position of an actuator or desired speed of the motor would be necessary. Feedback control is not defined in this document.

[1] `https://learn.adafruit.com/adafruit-tb6612-h-bridge-dc-stepper-motor-driver-breakout`
[2] `https://www.arduino.cc/en/Reference/Stepper`

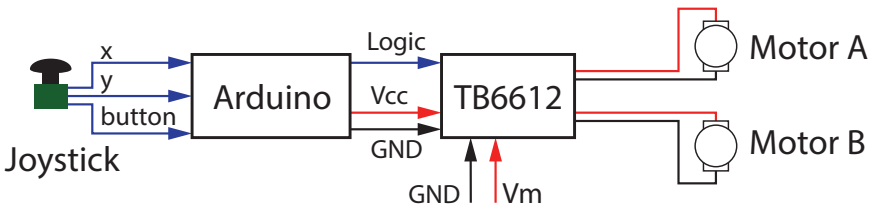

Figure 1: Block diagram for the joystick control of two DC motors.

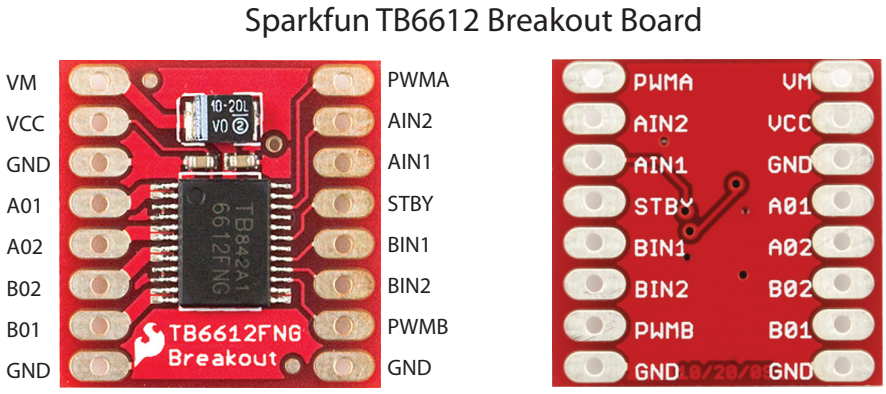

Figure 2: Top and bottom views of the Sparkfun TB6612 breakout board used in the joystick demonstration apparatus. Photographs are from `https://www.sparkfun.com/products/9457`. Annotations are added to the image on the left.

Table 1: Pin definitions for the TB6612 breakout board used in the joystick demonstration apparatus for controlling two brushed, DC motors. Valid alternatives exist for different pin numbers and wire colors. AO1, AO2, BO1 and BO2 are outputs from the TB6612 to DC motors A and B, and hence, do not have Arduino pins. One of the ground pins on the TB6612 is not connected, though for high current applications the ground pins serve as heat dissipation paths, so it should be connected.

Board label	Arduino pin	Wire color	Board label	Arduino pin	Wire color
VM	5V	red	PWMA	10	green
BCC	5V	red	AIN2	9	yellow
GND	GND	black	AIN1	8	blue
AO1	NA	red	STBY	7	white
AO2	NA	black	BIN1	5	blue
BO2	NA	red	BIN2	4	yellow
BO1	NA	black	PWMB	3	green
GND	GND	black	GND	GND	not used

Figure 3: Photograph of the Adafruit joystick board, from `www.adafruit.com`. On the left of the image, 5 input/output pads with labels, `Vcc`, `Xout`, `Yout`, `Sel` and `GND` are visible. `Vcc` and `GND` are for DC power compatible with the analog inputs of the microcontroller, e.g., 3.3 VDC or 5 VDC. `Xout` and `Yout` are the potentiometer outputs, which are connected to analog input channels of the microcontroller. `Sel` is the momentary output button, which is connected to one of the interrupt channels on the microcontroller.

Motor Controller Board

Figure 2 shows the top and bottom of the TB6612 breakout board from Sparkfun[3]. The Adafruit version of the board looks different, but has the same pins (in a different physical layout) and the same functionality[4]. The TB6612 is a dual half-bridge motor driver that can supply up to 1A to each of two brushed DC motors.

Table 1 is the wiring for the demonstration board for ME 491. Different pin assignments and (of course) wire colors are feasible. However, the pin assignments in Table 1 *must* match the code in Listing 2.

Joystick

Figure 3 is a photograph of an inexpensive ($5 USD) joystick from Adafruit[5]. Similar products are available from other vendors[6]. The key feature is that the joystick provides input control in two directions that we will call x and y. The directions correspond to the motion of the joystick and do not necessarily require the output to be assigned to a physical direction. In fact, direction x in this system is used to set the speed and rotation direction of Motor A and direction y is used to set the speed and rotation direction of Motor B.

The joystick has two potentiometers that have wipers moved by the knob. One potentiometer produces a control signal in the x direction. The other potentiometer produces a control signal in the y direction. The knob has springs that return the potentiometers to a center or neutral position when no lateral force is applied to the top of the knob. Typically the neutral position is associated with $x = 0$ and $y = 0$. The knob can be displaced in either the $+x$ or $-x$ direction while being simultaneously displaced in the $+y$ or $-y$ direction.

[3] `https://www.sparkfun.com/products/9457`
[4] `https://www.adafruit.com/product/2448`
[5] `https://www.adafruit.com/product/512`
[6] See `https://www.sparkfun.com/products/9760` and `https://www.amazon.com/dp/B00WH89RTS/`

Each potentiometer has a single output line that is connected to an analog input channel. The output is a voltage between `GND` and `Vcc`. The Arduino code reads the analog input to obtain the magnitude and direction of the control signals that are determined by the knob position.

Find Voltage Output in the Neutral Position

When the joystick knob is in the neutral position, the x- and y-direction potentiometers are nominally in the middle of their range. When digitized on the analog input channel of an Arduino Uno, the potentiometer reading would be roughly half of the 10-bit input range, or a value of 511. However, when the knob is in the neutral position, the analog input values are unlikely to be exactly 511 due to physical and electrical imperfections. A simple calibration can correct for the deviation from the midpoint of the analog input scale. Remember that both the x and y channels are independent, so when we refer to the analog input scale, we can mean either x or y inputs.

To calibrate the joystick, connect it in the final circuit configuration, i.e., use the circuit with the TB6612 and any other components. Run the simple Arduino code in Listing 1 to read and print the values on the analog input lines corresponding to the x and y axis of the joystick. During calibration keep the joystick knob in the neutral position. Record the output values for the two analog input channels.

The calibration experiment gives the values on the analog input channels when the joystick knob in the neutral position. Considering that the joystick allows motion in the forward and reverse direction, the neutral position corresponds to $x = 0$ and $y = 0$, even though both potentiometers are in the nominal middle of their range. The calibration values for the x- and y-directions are likely to be different.

Define the Deadband

We cannot assume that analog input value will be equal to the calibration values for $x = 0$ and $y = 0$ every time the knob is released to the neutral position. Small variations in the input signal are due to noise and drift, and because the potentiometer may not return to the same physical position every time the knob is released. To accomodate variations in the analog signal corresponding to the neutral position, we introduce a deadband range of analog input values that we consider to be equivalent to neutral.

```
//  File:  joystick_calibration.ino

void setup() {
  Serial.begin(9600);
}

void loop() {

  int xinput=0, yinput=1, xpot, ypot;

  xpot = AnalogRead(xinput);
  ypot = AnalogRead(yinput);

  Serial.print(xpot);  Serial.print("  ");  Serial.println(ypot);
}
```

Listing 1: Simple Arduino code to print analog input values corresponding to the joystick in the neutral position. Note that this code can be also useful as a check on the circuit for the joystick.

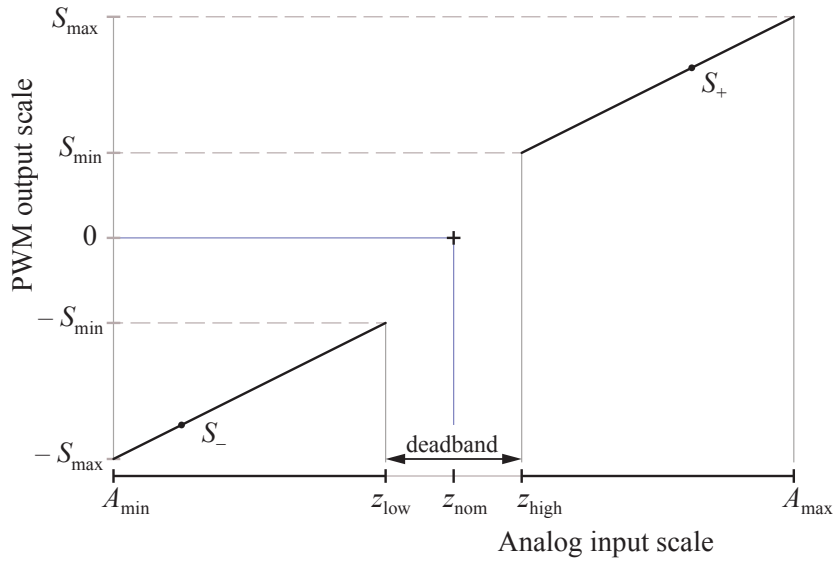

Figure 4: Nomenclature for mapping analog input values from the joystick (potentiometer) to the 8-bit PWM signals used to control motor speed. Motor direction is determined by the S_+ and S_- parts of the mapping. z_{nom} is the value of the analog input reading that corresponds to the joystick in the neutral (resting) physical position. In other words, z_{nom} is the analog input reading for the *zero* position of the joystick.

The deadband is a range around the neutral position of the knob that we consider to be zero output from the joystick. There are separate deadbands for the x- and y-direction potentiometers. Suppose the analog input reading on the x-direction pot is 502 (on a 10-bit scale). Then we might specify the deadband as 10 units on either size of 502, i.e., a reading in the range $492 \leq p \leq 512$ is considered to be zero. Therefore, the calibration experiment gives us the nominal values of analog input signals corresponding to $x = 0$ and $y = 0$, and we *choose* a range of input values slightly to account for variations in the signal. The size of the deadband is somewhat arbitrary. A smaller (± 5 units), or larger (± 15 units) may also work.

Map Analog Input to PWM Output

The joystick provides a control signal from the user. Direction of each motor is controlled by logical lines on the input of the TB6612. Speed of each motor is controlled by the PWM signals input to the TB6612. The MMEmotor library provides a programming interface to manage the logical lines and PWM levels. In this section we describe how to map the joystick inputs to the desired motor control outputs. Figure 4 and Table 2 define nomenclature used in that task.

Figure 4 is a diagram that shows how the deadband and analog input range are used to compute the motor speed setting from the reading of the analog input. Table 2 defines variables appearing in Figure 4. We use those variables to map the analog inputs from the joystick to PWM outputs sent to the TB6612 motor control board. Note that mapping is the same for both x and y directions.

In addition to the deadband for the potentiometer input, the motors exhibit their own deadband behavior. A small, brushed, DC motor will not turn unless the PWM duty cycle is above a threshold that we call S_{min} (minimum speed). It makes sense to map the deadband limits of the potentiometer to the value of S_{min}, since any smaller value of the duty cycle will not result in motion of the motor.

Table 2: Definition of parameters used to map the joystick potentiometers to the PWM duty cycle that controls motor speed. Refer to Figure 4 for a graphical interpretation of these parameters.

Parameter	Range or value	Comment
$A_{\min}$	0	Minimum 10-bit integer value for the analog input range. Constant.
$A_{\max}$	1023	Maximum 10-bit integer value for the analog input range. Constant. Use $A_{\max} = 4095$ for microcontrollers with 12-bit analog input resolution.
$S_{\min}$	$0 \leq S_{\min} < S_{\max}$	Minimum value of the duty cycle. $S_{\min}$ is the smallest value of the duty cycle that causes the motor to rotate.
$S_{\max}$	≤ 255	Maximum value of the duty cycle. $S_{\max}$ is used to limit maximum motor speed, if that is desired.
z_{nom}	≈ 511	Nominal value of analog input when the joystick knob is in the neutral position. z_{nom} is the analog input value for a position (x or y) equivalent to zero. The "z" is meant to suggest "zero".
z_{low}	$z_{\text{nom}} - \delta$	lower range of deadband for analog input reading, $\delta \approx 10$
z_{high}	$z_{\text{nom}} + \delta$	lower range of deadband for analog input reading, $\delta \approx 10$

We also define $S_{\max}$ (maximum speed) as the upper limit of the duty cycle for the PWM signal to the motor controller. Unlike $S_{\min}$, which is due to the physical limitations of the motor, $S_{\max}$ is a value that we choose in order to limit the speed of the motor. Choosing $S_{\max} = 255$ allows the maximum motor speed since the PWM duty cycle is specified as an 8-bit value.

The first step in the mapping is to determine whether the joystick indicates a forward or reverse direction. In other words, we have to determine whether the analog input value is greater than z_{high} (positive direction) or less than z_{low} (negative direction). Use S_+ and S_- to designate the PWM output values for forward and reverse direction of the motor, respectively.

From the geometry of Figure 4 we obtain the following formulas for S_+ and S_-.

$$S_+ = S_{\min} + \frac{p - z_{\text{high}}}{A_{\max} - z_{\text{high}}} \left(S_{\max} - S_{\min}\right) \tag{1}$$

$$\begin{aligned} S_- &= -S_{\max} + \frac{p - A_{\min}}{z_{\text{low}} - A_{\min}} \left(-S_{\min} - (-S_{\max})\right) \\ &= -S_{\max} + \frac{p - A_{\min}}{z_{\text{low}} - A_{\min}} \left(S_{\max} - S_{\min}\right) \end{aligned} \tag{2}$$

Note that $A_{\min} = 0$ and $A_{\max} = 1023$ for 10-bit analog input resolution. Equation (1) and Equation (2) use a linear variation of PWM output with potentiometer position. A non-linear function could be used instead, though the advantage of doing so is not obvious. Physical testing shows that the mapping in Equation (1) and Equation (2) works well.

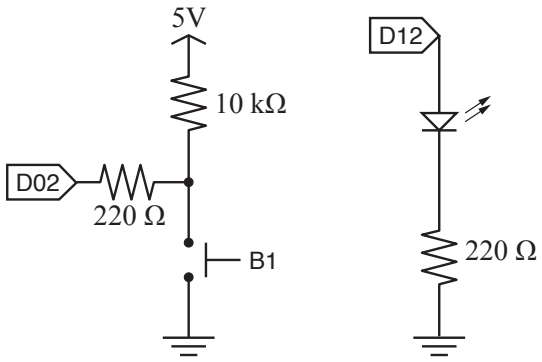

Figure 5: Button and LED circuits.

Arduino Code

The code in the `TB6612_two_motor_joystick.ino` sketch is spread over Listing 2, 3 and 4. The code can be downloaded from the web page for Lecture 7 in ME 491 for Fall 2017, `http://web.cecs.pdx.edu/~gerry/class/ME491/lecture/ME491_lecture_07.html`.

Listing 2 contains the code to define global variables and `setup` function for one-time initialization tasks. The `setup` function also contains code for the potentiometer calibration, which is only included if the `#define JOYSTICK_CALIBRATION` line is un-commented.

Listing 3 contains the `loop` function, which is repeated indefinitely and the `get_speed` function, which implements Equation (1) and Equation (2). Four lines in the `loop` function do the work of setting the motor speeds according to the joystick potentiometer voltages.

```
// -- Get speed value from joystick position
x_speed = get_speed(X_INPUT_PIN, XLOW, XHIGH, min_speed, max_speed, ANALOG_MAX);
y_speed = get_speed(Y_INPUT_PIN, YLOW, YHIGH, min_speed, max_speed, ANALOG_MAX);

// -- Spin both motors.  Direction determined by the sign of x_speed and y_speed
motorA.moveAtSpeed(x_speed);
motorB.moveAtSpeed(y_speed);
```

Of course, this compact code is made possible by the utility function `get_speed` and the code in the MMEmotor library that defines the `moveAtSpeed` method for the `MMEmotor` object.

The `Sel` button on the Adafruit joystick is used to toggle an LED on and off. The button input is managed via an interrupt on digital pin 2. The interrupt handler is specified via the call to `attachInterrupt` in the `setup` function. (See Listing 2.) The interrupt handler code is in Listing 4.

Figure 5 shows the two circuits used to implement the select button feature of the joystick. The circuit on the left side of Figure 5 ties digital pin 2 to the button circuit. A pull-up resistor maintains the pin at HIGH (+5V) until the button is depressed. The right side of Figure 5 is a standard LED circuit that is activated by digital pin 12. The LED is used for demonstration to indicate the button state. In a practical application, some additional action in response to a button click.

```
//   File: TB6612_two_motor_joystick.ino
//
//   Demonstrate use of the MME motor library for driving the Sparkfun TB6612
//   break-out board.  Run two motors at different directions and speeds
//   which are controlled by input from an Adafruit joystick board
//
//   Gerald Recktenwald, gerry@pdx.edu,  created 2017-11-05

#include <MMEmotor.h>

// -- Analog input pins for the joystick
const byte X_INPUT_PIN = 0;
const byte Y_INPUT_PIN = 1;
const int  ANALOG_MAX = 1023;    // Maximum value for analog input:  10-bit => 1023

// -- Interrupt pin and status for button clicks, Interrupt 0 is on pin 2 !!
const int  button_interrupt = 0;
const int  LED_pin = 12;
int  toggle_on = false;         //  Global variable stores state changed by button clicks

// Logic pins to control the motors: PWMA and PWMB must be pins capable of PWM output
// User needs to correctly wire these pins to corresponding contacts on TB6612 breakout
#define PWMA 10   // Motor A
#define AIN1 9
#define AIN2 8
#define STBY 7   // Standby pin of TB6612. Shared by both channels
#define PWMB 3   // Motor B
#define BIN1 4
#define BIN2 5

// -- Initialize MMEmotor objects
MMEmotor motorA = MMEmotor(AIN1, AIN2, PWMA, STBY);
MMEmotor motorB = MMEmotor(BIN1, BIN2, PWMB, STBY);

// ---------------------------------------------------------------------------
void setup() {

  Serial.begin(9600);

  attachInterrupt( button_interrupt, handle_click, RISING);  // Register interrupt handler

  pinMode(LED_pin, OUTPUT);    //  LED used to indicate value of toggle_on

  // -- Include joystick calibration code ONLY when a new joystick is used.
  //    In practice the resistance of the joystick potentiometers is not equal to
  //    half of the maximum resistance. Therefore, the analog input from the pot
  //    as a voltag divider is not in the middle of the analog input rage.
  //    The goal of the calibration is to find the analog input values on both
  //    axes corresponding to the neutral position of the joystick.  The following
  //    loop could be in a separate sketch, but we include it here for convenience.
  //    Using #ifdef determines whether the code is even compiled, so there is
  //    no cost of memory or execution speed to include the code here.  To run
  //    the calibration, just uncomment the #define JOYSTICK_CALIBRATION line

// #define JOYSTICK_CALIBRATION

#ifdef JOYSTICK_CALIBRATION

  while ( 1 ) {
    int xpot = analogRead(X_INPUT_PIN);
    int ypot = analogRead(Y_INPUT_PIN);
    Serial.print(xpot);   Serial.print("   ");   Serial.println(ypot);
  }
#endif

}
```

Listing 2: Part 1 (of 3) of the Arduino code to demonstrate joystick control of DC motors.

```
// ------------------------------------------------------------------------
void loop() {

  // -- Define thresholds used to set deadband around neutral position of the potentiometers
  const int XLOW = 492, XHIGH = 512;
  const int YLOW = 504, YHIGH = 524;

  int min_speed = 20, max_speed = 155, x_speed, y_speed, wait_time = 3000;

  // -- Get speed value from joystick position
  x_speed = get_speed(X_INPUT_PIN, XLOW, XHIGH, min_speed, max_speed, ANALOG_MAX);
  y_speed = get_speed(Y_INPUT_PIN, YLOW, YHIGH, min_speed, max_speed, ANALOG_MAX);

  // -- Spin both motors.  Direction determined by the sign of x_speed and y_speed
  motorA.moveAtSpeed(x_speed);
  motorB.moveAtSpeed(y_speed);

  // -- Turn LED on/off to indicate button state
  if ( toggle_on ) {
    digitalWrite(LED_pin, HIGH);
  } else {
    digitalWrite(LED_pin, LOW);
  }

  Serial.print(x_speed);   Serial.print("  ");   Serial.println(y_speed);
}

// ------------------------------------------------------------------
// Compute motor speed from potentiometer reading.  User supplies deadband
// limits and min/max motor speeds
//
// Input:
//   pot_pin = analog input pin for this channel (x or y) of the joystick pot
//   zlow, zhigh = deadband limits.  Analog inputs in the range zlow <= v <= zhigh
//                 are considered to be zero
//   Smin, Smax = minimum and motor speeds on 8-bit scale [0, 255]
//                Smin is usually a value below which the motor doesn't turn
//                Smax sets the maximum desired speed when joystick is at max position
//   Amax = maximum value for analog input range.  Amax = 1023 for Arduino Uno
//
// Output:
//   motor_speed = PWM value used to set speed, Smin <= |motor_speed| <= Smax

int get_speed(int pot_pin, int zlow, int zhigh, int Smin, int Smax, int Amax) {

  float s;                      //  Use float to avoid rounding during mapping
  int pot_val, motor_speed;

  pot_val = analogRead(pot_pin);

  if (pot_val > zhigh) {
    s = Smin + (pot_val - zhigh) * float( Smax - Smin ) / float(Amax - zhigh);
    motor_speed = constrain( int(s), Smin, Smax);   //  Make sure speed is in range
  } else if (pot_val < zlow) {
    s = -Smax + pot_val * float( Smax - Smin ) / float(zlow) ;   // Amin = 0
    motor_speed = constrain( int(s), -Smax, -Smin);   //  Make sure speed is in range
  } else {
    motor_speed = 0;  // potentiometer reading in the deadband
  }

  return(motor_speed);
}
```

Listing 3: Part 2 (of 3) of Arduino code to demonstrate joystick control of DC motors.

```
// ----------------------------------------------------------------------
//  Interupt handler to respond to button clicks.  This function has two
//  goals: (1) reject button bounces, and (2) toggle a global flag if a
//  lepitmate button click event occurs.

void  handle_click()
{
  static unsigned long last_interrupt_time = 0;       //  Zero only when code first runs

  unsigned long interrupt_time = millis();             //  Read the clock
  if ( interrupt_time - last_interrupt_time > 200 ) {  //  Only count clicks separated by 200 msec
    toggle_on = !toggle_on;
  }
  last_interrupt_time = interrupt_time;
}
```

Listing 4: Part 3 (of 3) of Arduino code to demonstrate joystick control of DC motors.