MTRE 2610 Engineering Algorithms and Visualization – Dr. Kevin McFall

Laboratory – H-bridge Motor Drivers

Introduction

Over the next several weeks, laboratory exercises will work towards programing the necessary sensors and actuators to control the color sorting robot depicted in Figure 1. The completed system involves all the major elements listed below:

A. Motor to rotate arm
B. Switch for rotary encoder
C. Limit switch to determine home position
D. Air compressor
E. Solenoid valve for generating suction
F. Solenoid valve for lowering arm
G. Piston-cylinder driving vacuum
H. Piston-cylinder delivering vacuum
I. Piston-cylinder to lower arm
J. Light source for part detection
K. Phototransistor for part detection
L. Analog color sensor
M. Cable channels
N. Proprietary Fischertechnik microcontroller to be replaced with an Arduino Uno

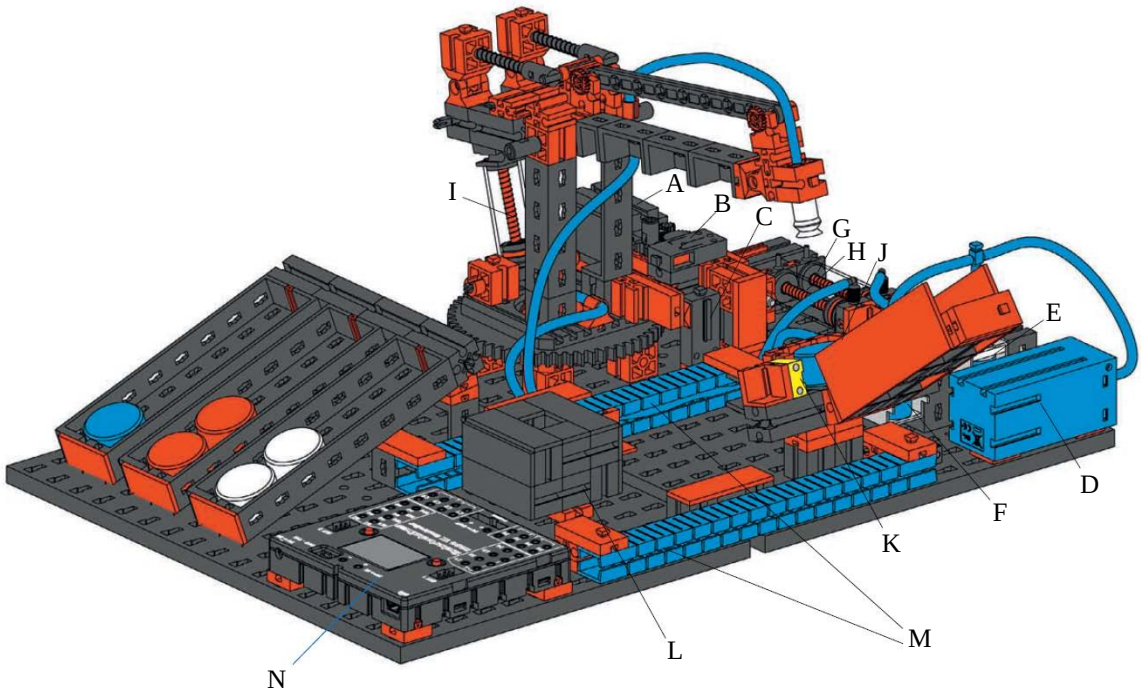

Figure 1: Schematic of completed color sorting robot including major elements.

This laboratory exercise deals primarily with element A, the DC motor used to rotate the robotic arm. The arm requires rotation in both directions, which will be accomplished using an H-bridge.

Important modification

To allow complete rotation of the arm, the hoses and limit switch stop identified in Figure 2 must be removed. This step *must* be completed prior to connecting the motor. To remove the hoses, pull one out of the base of cylinder I and pull the other out of the suction cup at the end of the arm. Lift the three red plastic pieces comprising the stop out of the arm base. After removing the hoses, reattach the suction cup to the end of a hose so that it will not be lost and set the stop aside to be reattached later.

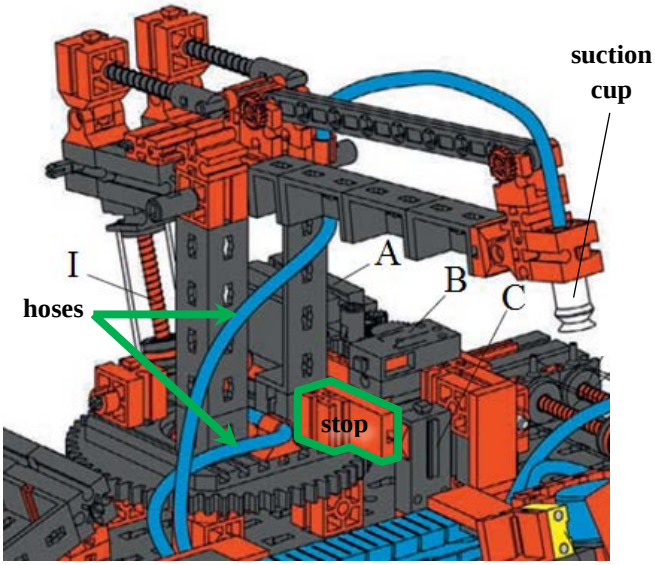

Figure 2: Parts to remove before attempting to operate arm rotation motor.

Motor motion in both directions

The previous laboratory used a transistor to switch current to a DC motor. Even though a 5 V electric potential was sufficient to power the DC motor in that activity, the transistor was necessary since the Arduino digital output pin is incapable of supplying the requisite current. Motor element A in Figures 1 and 2 requires 9 V to operate. The same transistor could be used to drive it if a 9 V potential were applied across the motor via the emitter and collector. However, the color sorting robot must rotate in both directions, which would only be possible by manually switching polarity across the emitter and collector. Employing an H-bridge integrated circuit allows the polarity to instead be switched automatically with output from the Arduino.

Integrated circuits

Because of advances in semiconductor technology, it is now possible to place millions of electrical components on a single piece of silicon, called an integrated circuit (or simply IC). ICs are also sometimes referred to as chips. ICs are typically very small (occupying less area than your smallest fingernail), requiring fine wires (about the diameter of a hair) for connections. Consequently, to be useful as a circuit element, ICs must be suitably packaged. Although somewhat confusing, the terms chip and IC are also used to refer to packaged integrated circuits. ICs come in a variety of plastic or ceramic packages with connecting pins designed for mounting on printed circuit boards. The most useful package for building prototype circuits is the dual inline package or DIP, which has two rows of pins that conveniently insert into a breadboard by straddling the trough. Pins are labeled numerically starting at 1, and each DIP package is marked to identify pin 1. In most cases, the package has a notch at the top or a small dimple adjacent to pin 1 as in Figure 3.

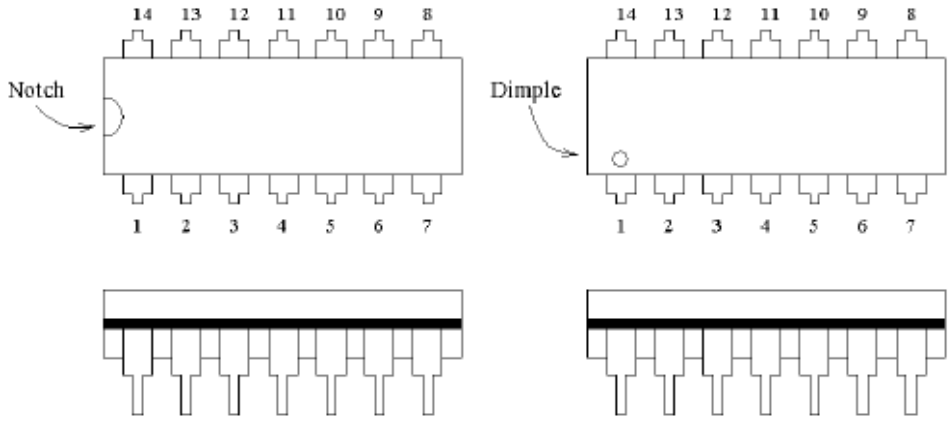

Figure 3: Top and side view of pin configuration for a typical 14-pin DIP IC.

H-bridge integrated circuit

A pin-out of the H-bridge IC, with part number SN754410, appears in Figure 4, identifying the purpose of each pin and how it should be connected to be used as a motor driver. Although appearing somewhat complicated, H-bridge operation itself is quite simple: if an external 5 V signal is applied to an A pin, the voltage applied to V_{CC2} will appear on the corresponding Y pin. Alternatively, if ground is applied to A, then its Y will also be grounded. However, if the EN pin (which stands for enable) is not connected to 5 V, then neither voltage nor ground will appear at Y regardless of the A value. This written description is summarized in the function table for the H-bridge appearing in Table 1. When both EN and A are H, i.e. high or 5 V, then the output Y is also high. If enabled (EN high), but A is low (L or grounded) then Y will also be low. The X for A in the last row of Table 1 indicates that is does not matter what A is when the enable is low: output Y will always be unconnected, which is represented by the high-Z or high impedance condition. Operation of ICs is generally described by a function table like Table 1 rather than written in words.

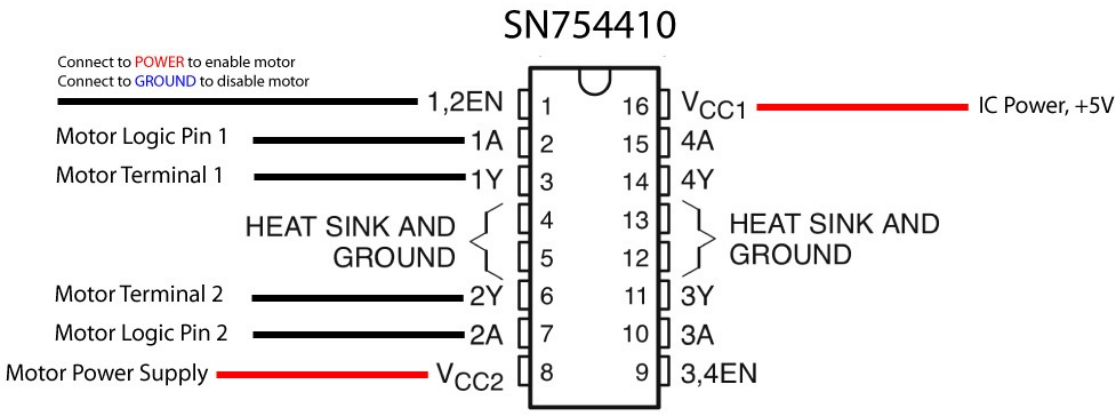

Figure 4: Pinout for the H-bridge and how it should be connected to the Arduino and motor in order to use it as a bi-directional motor controller.

Table 1: Function table for the H-bridge IC.

Inputs		Output
EN	A	Y
H	H	H
H	L	L
L	X	Z

The SN754410 H-bridge IC offers four independent pairs of A and Y pins which can be used for any purpose the designer can imagine. Notice that a single enable, pin 1, turns on both 1A/1Y and 2A/2Y; a similar enable, pin 9, is used to turn on 3A/3Y and 3A/4Y. In the case of this exercise, the desire is to apply either +9 V or –9 V across the leads of the DC motor to make the robotic arm rotate full speed in either the clockwise or counter-clockwise directions. Note that, if desired, motor speed could be controlled by applying a PWM signal with a duty cycle under 100% to the enable. By connecting the motor leads to 1Y and 2Y, +9V can be applied across the motor by the Arduino imposing high to 1A and low to 2A. Similarly, –9 V would be applied if the Arduino writes low to 1A and high to 2A. If the Arduino applies the same logic value (whether high or low) to both 1A and 2A, no current flows through the motor (and no movement would occur) since both leads would be at the same electric potential. Of course, 5 V must be applied to 1,2EN on pin 1 for this to work. Figure 5 illustrates how the connections in Figure 4 are accomplished when connecting the components using a breadboard.

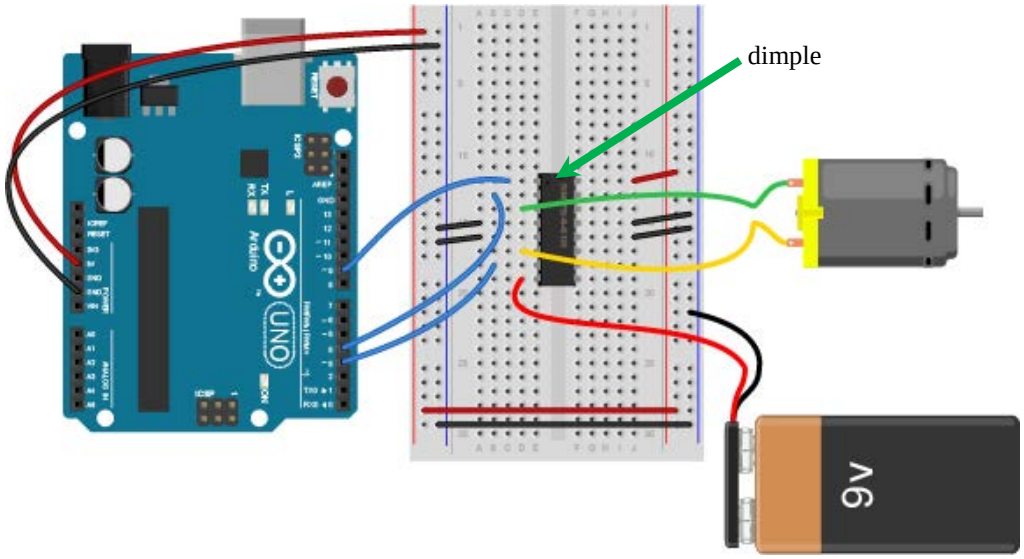

Figure 5: Wiring diagram for connecting the H-bridge IC to the Arduino, motor, and 9V battery using a breadboard.

Switch debouncing

When the contacts of a switch open (or close), dynamics of the moving parts can cause the contacts to momentarily open and close numerous times. This phenomenon is called switch bounce, which results in multiple readings from a pushbutton with only a single press. Electrical engineers traditionally eliminate switch bounce by adding capacitors to the switching circuit so that the resulting time constant for a change in value is longer than the bounce duration. Alternatively, switch bounce can be eliminated in software rather than hardware.

Consider programming an Arduino to take some action when the button is pressed. Every time through the loop function, button position in the previous loop iteration should be compared to its current position. Action should be taken when change occurs from the button's normal position to its depressed position. Such action is called edge-triggered. Two variables should be stored, one each for the previous and current button positions. At the end of each loop, the previous value replaces the current value in anticipation of beginning the next iteration.

To eliminate switch bounce, the time of the previous edge-trigger and the current time must also be compared. Edge signals should only trigger actions if sufficient time has elapsed since the last triggering event. If time is too short between edges, the current edge can be assumed a bounce and not an actual button press. The time threshold to trust an edge must be longer than the bounce duration. This will be different for every switch and should be determined experimentally.

Arduino interrupts

One issue with using an edge trigger to initiate an action is that counts will be missed if the Arduino is busy executing another command when an edge occurs. Essentially, the `digitalRead` on the encoder pin must be continually executed while the rest of the loop is running. An interrupt is used to accomplish this: the interrupt pin is constantly polled in parallel with execution of the loop. When an interrupt condition is met, the loop pauses while a separate interrupt function is called. The loop then resumes after the interrupt function completes. To create an interrupt, include the command `attachInterrupt(0, encoderInterrupt, CHANGE)` in the `setup` function. While `loop` is running, the function `encoderInterrupt` will execute whenever a change (either rising or falling edge) occurs in the logic value on pin 2 (for whatever reason interrupt number 0 on the Arduino Uno is connected to pin 2). Be sure to define body of the function `encoderInterrupt()`to handle what should happen when an edge is detected. Note that using `CHANGE` will execute `encoderInterrupt` both when the button is pressed and released. Using `RISING` or `FALLING` executes only once, on either a rising or falling edge, respectively.

Laboratory exercise procedure

Connect the H-bridge as in Figure 5. To confirm successful operation, have the motor change directions every two seconds, and demonstrate that the speed can be controlled by changing a constant in the Arduino sketch.

Add a pushbutton switch, with its accompanying resistor, to the breadboard as in the previous laboratory exercise. Without using an interrupt, program the Arduino to count every positive edge and print a running total of detected edges to the serial monitor. The occasional switch bounce may be apparent. Leave the motor off when working with the pushbutton.

Rewrite the sketch so that the counts are incremented in an interrupt rather than in `loop`. Be sure to declare the count variable as global so that changing it in `encoderInterrupt` will affect its value in `loop`. In fact, declare the count variable as a `volatile int` data type as recommended[1]; volatile variables are housed in RAM rather than storage registers, which speeds access of their data. Notice how the interrupt is faster; extra counts from switch bounce occur significantly more frequently. To avoid bounce, the count should only be incremented inside `encoderInterrupt` if sufficient time has passed since the previous edge.

Use the `micros` to obtain the current time in microseconds. The returned numbers will be rather large, so store them in variables of the `unsigned long` data type. In `setup`, store the time in a global variable. When a `CHANGE` is detected and the interrupt function executes, only register a click if the difference between the current time and last time (stored in `setup`) are significantly different. The threshold for "different" will need to be tuned: too short will count bounces and too long will miss clicks. Implement switch debouncing in the interrupt by increasing the time threshold to trust an edge until a single edge is counted for each change in button value. Show the edge count in the serial monitor to confirm a single detected edge per change in button value.

Finally, count the number of button presses (not button state changes) and toggle the motor direction every five times the button is pressed.

Grading rubric

1. 25 points: Motor changes direction every two seconds with variable motor speed
2. 25 points: Display button edge count in serial monitor, possibly including bounces
3. 25 points: Display button edge count in serial monitor using interrupt, and with all bounces eliminated
4. 25 points: Motor changes direction after every five button presses

[1] https://www.arduino.cc/en/Reference/attachInterrupt