Design Competition 2013 – Milestone 6

Demonstrate to Nick by Wed. 3/17/2013

Your goal is to mount a wheel to a DC motor and read the rotation of the wheel with a sensor. Demonstrate the wheel rotating 180 degrees when the USER button is pushed on the NU32.

Driving a DC motor

You have several choices for drive motors this year:

- 12V 500rpm (250rpm at 6V), 2" long
- 6V 600rpm, long motor, 2" long
- 6V 300rpm, short motor, 1.5" long
- 3V 30rpm (60rpm at 6V), 1.6" long

Each motor uses slightly different amounts of current, and will have different amounts of torque. They all have the same cross section, so you can switch them in and out of your robot without much effort.

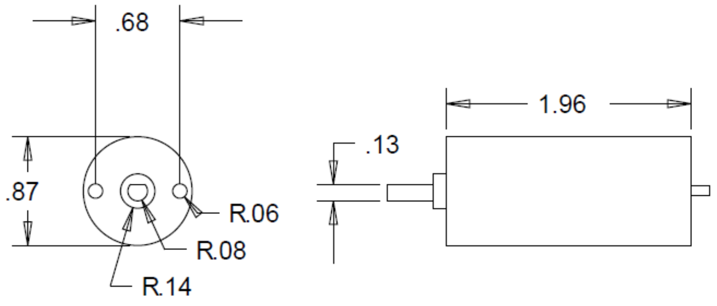
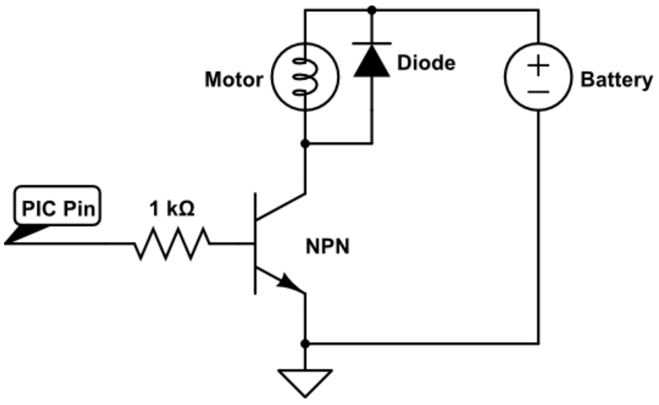
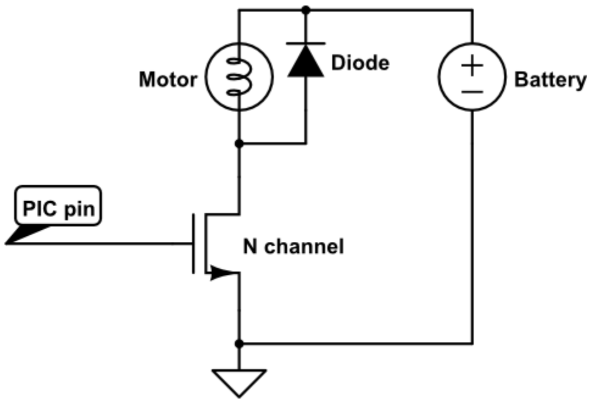


Figure 1: Other motor lengths: 2", 1.5", 1.6"

The output voltage of a PIC32 pin is 3.3V, with a maximum of ~20mA. This is not enough to drive a motor, so you have to build an amplifier to take the on/off signal from the NU32 and turn the motor on and off.

If you only need the motor to spin in one direction, you can use one of the following transistor circuits:





Note the diode in parallel with the motor, pointing towards the battery. This diode is called a "flyback diode". The motor is a coil of wire, like an inductor. An inductor does not want to change current instantaneously, so when you turn the motor off (trying to instantly change the current), the voltage across the motor will spike, potentially damaging the transistor, making voltage noise on the ground line and ruining your analog reads, and potentially causing enough noise to reset the NU32. The flyback diode reduces the voltage spike, and sends the current back to the battery. **But there will always be noise in your system due to the motor.** Noise can be further reduced by putting capacitors everywhere, using different batteries for the NU32 and motor, and ultimately by optically isolating the motor circuit.

Usually you want to drive your motor in both directions. To do this, you need to be able to send current through the motor in both directions. You could use 4 transistors, and turn them on and off carefully as to not short your battery, but a better choice is an **h-bridge**. We will use the L293D.

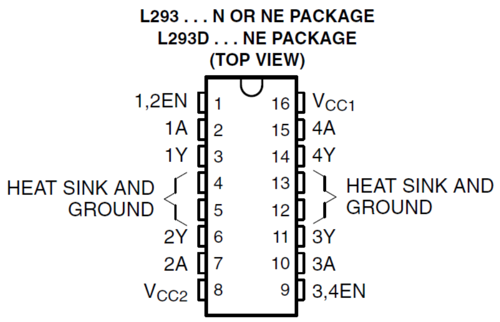
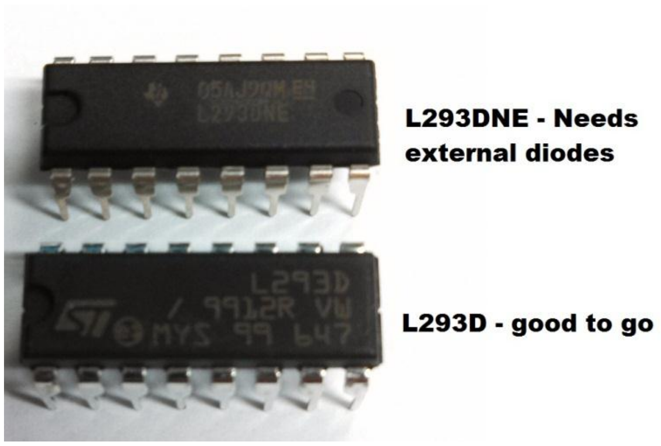


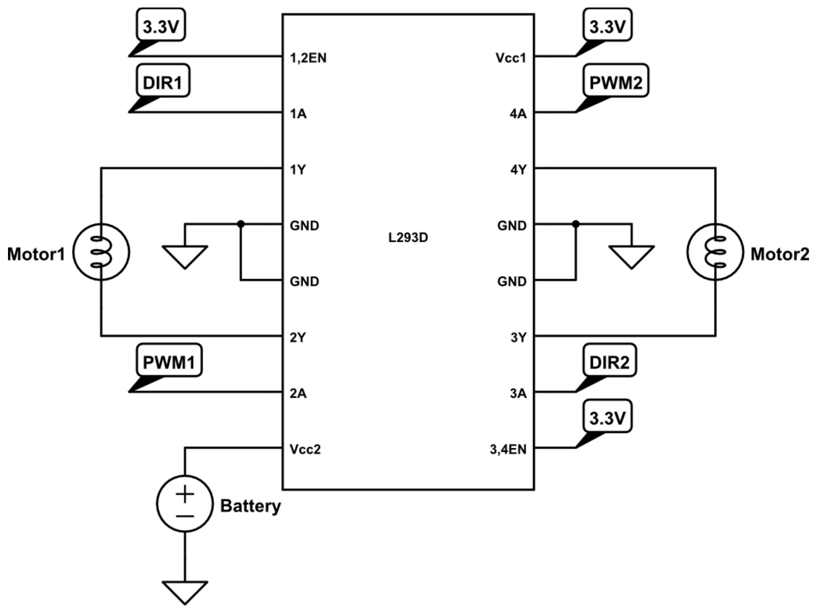
Figure 2: Vcc2 is battery voltage (6V), Vcc1 is logic voltage (5V or 3.3V)

Only use L293D chips that say L293D (not L293DNE). The L293D contains the flyback diodes internally, the L293DNE does not, and without the flyback diodes it does not work. You can add your own diodes if you want, but we have plenty of L293Ds.



The L293D is not a very good h-bridge (with a 6V battery, it only applies ~4.5V to the motor), but it comes in a convenient, bread-boardable DIP package. More modern h-bridges, like the TB6612 we used in ME333, use less “voltage overhead” than the L293D, but only come in tiny surface mount packages that need a breakout board.

The L293D is a “quadruple half-h bridge” chip, meaning that it has 4 outputs that can drive motors. You can drive 4 motors in one direction, 2 motors in 2 directions, or 1 motor in 2 directions and 2 motors in 1 direction. You can also drive 1 stepper motor in 2 directions. This is how you would drive 2 motors in 2 directions:



The 2 motors are controlled in the same way. Each motor has 2 inputs, DIR, for the direction of the motor, and PWM, for the speed of the motor. When DIR and PWM are the same voltage (0V or 3.3V), the motor is off. When DIR is HIGH and PWM is low, the motor will spin CW. When DIR is LOW and PWM is HIGH, the motor will spin CCW, as shown in the logic table for the L293D:

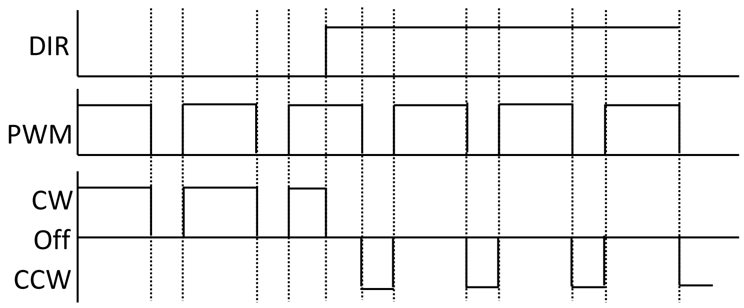
EN	1A	2A	FUNCTION
H	L	H	Turn right
H	H	L	Turn left
H	L	L	Fast motor stop
H	H	H	Fast motor stop
L	X	X	Fast motor stop

L = low, H = high, X = don't care

Figure 3: 1A is DIR, 2A is PWM (doesn't matter which is which)

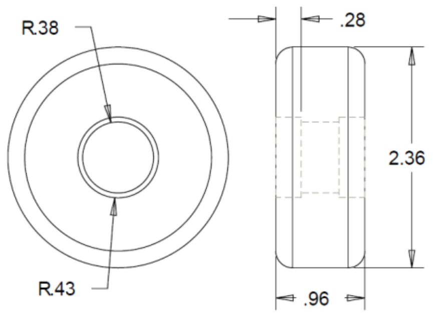
PWM stands for Pulse Width Modulation. At a fixed frequency, usually between 1kHz to 40kHz, a digital pin is turned on for a percentage of the period of the fixed frequency, called the "duty cycle". At 100% duty cycle, the pin is always HIGH. At 50% duty cycle, the pin would show a perfect square wave. PWM is useful in setting the speed of motors, setting how bright an LED is, and making analog voltages.

The L293D has one quirk, related to the PWM input. Let's say you want to move forward at 75% speed, so you set DIR low and PWM to 75% duty cycle. Now you want to move backwards at 75% speed, so you switch DIR to HIGH. But the motor only goes at 25%. Why? Take a look at how the logic table treats a HIGH DIR and 75% HIGH PWM:

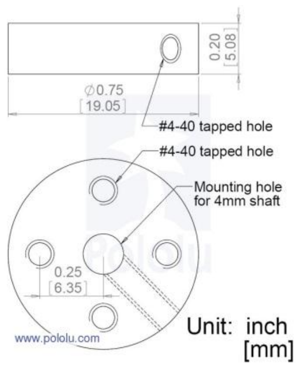


DIR and PWM are both HIGH 75% of the time, and opposite 25% of the time, so the speed is only 25%. So, in code, you must remember to use (100%-duty cycle) for PWM when DIR is HIGH, as is done in the milestone6.c example code.

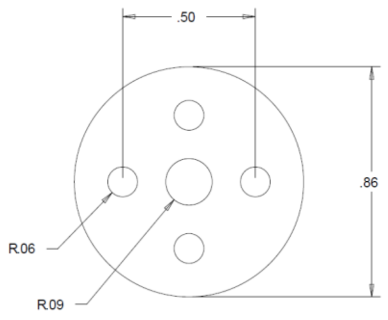
Now that you can make your motor spin, try to mount a wheel. We are using rollerblade wheels this year.



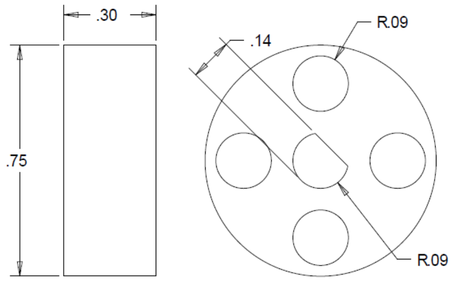
The rollerblade wheel gets nice traction, but it is meant to mount to bearings, not motors. To attach the motor shaft, you need some kind of hub. We have some aluminum hubs that mount directly to the motor shaft:



But the hub doesn't fit in the wheel. You can cut some wood inserts on the laser cutter, and glue the inserts in the wheel, lining up the screw holes. Then screw the wheel to the hub, and you've mounted the wheel to the motor.



The set screw in the hub doesn't grip the motor very well, so you could try 3D printing a hub with a D shaped hole that you can press the motor into:



Once you have the wheel mounted, think about how you can count the revolutions of the motor. An encoder is very useful: you can try to use the information to know the robot position on the arena, or use the information to set a motor speed in in/s, rather than % of the maximum speed, or use the information to make more accurate turns.

Most commercial encoders are "quadrature", meaning they have 2 channels. With 2 channels, you get 4 times the normal resolution of the encoder, and you can tell which direction the motor is moving. But you need twice the electronics, so let's build a more simple, single channel encoder.

For this, you need an encoder wheel. If you want to use an optointerrupter, the wheel looks like a gear. If you want to use an optoreflector, the wheel is a circular pattern of black and white dashes:

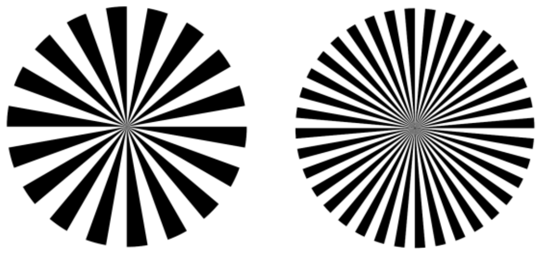
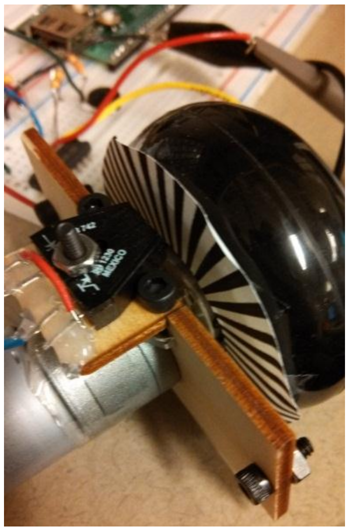
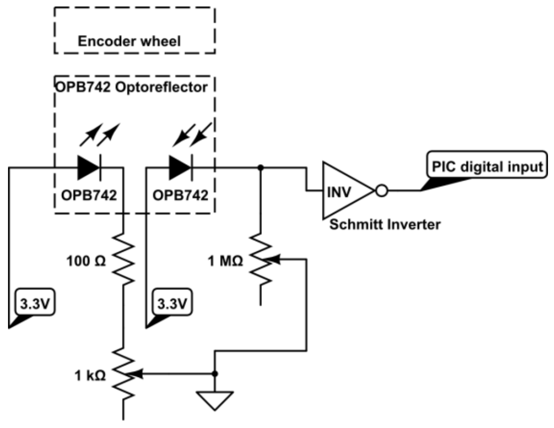


Figure 4: encoder.doc was used to make these, posted on the wiki

I think the optoreflector is a little easier, so I will mount it to the wheel with double-stick tape:



The output of the optoreflector isn't quite a square wave, so you need to add an inverting chip:



There are 3 ways to make this circuit work:

First, change the distance between the sensor and the wheel. There is an optimal distance, ~5mm. At 5mm, the LED is focused on the paper. Too close or too far, and the LED lights up a large amount of the paper and the sensor averages the black and white lines, and the sensor output is sinusoidal instead of a square wave. Try to mount the sensor with some adjustability.

Because your hub might not be perfectly perpendicular with the motor shaft, the output of the sensor might shift up or down with the angle of the wheel. This can be fixed with the second way to make this circuit work: use potentiometers to set the LED current, and sensor sensitivity. If the LED is too bright or if the sensor is too sensitive, the sensor will be saturated, and vice versa. Look at the output of the sensor on an oscilloscope and turn the potentiometer until the output looks like a square wave between 0V and 3.3V.

The third way to make this circuit work is by using a Schmitt Trigger inverting chip. The Schmitt inverter will take the raw sensor output and convert it to a square wave, and filter out noise that a normal inverter would keep by using hysteresis (that's what the Schmitt part does).

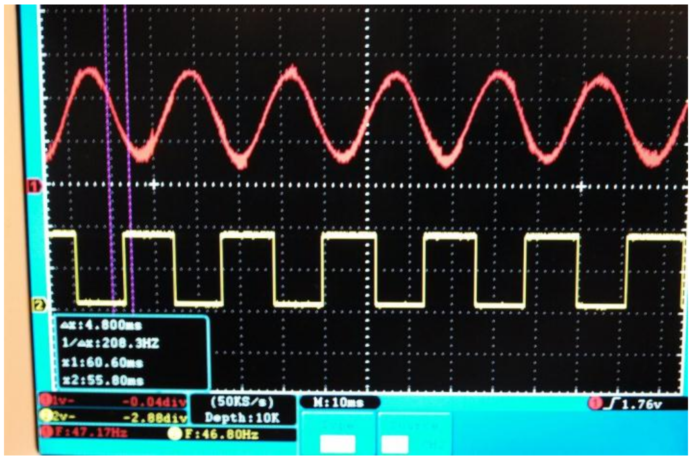


Figure 5: 1-Encoder output before Schmitt inverter, 2-Encoder output after Schmitt inverter

There are many ways to read this signal in code and interpret it. Here is a simple way:

- Set the motor speed and direction
- Start a while loop that reads the digital encoder signal
- Every time the digital signal changes, add one to a counter
- When the counter gets to a certain value (corresponding to the angle or distance you wanted to go), continue with your code

By counting the transitions, you will count twice as many times per revolution as there are dashes (but we don't really know if those transitions are in the same direction – just hope you don't hit anything!)

Assignment
Pick up a motor and roller blade wheel. Cut or print a hub. Add an encoder. Write code so that when a button is pushed (the USER button or some button you wire up) the motor turns on and spins 180 degrees and stops.

Demo it!