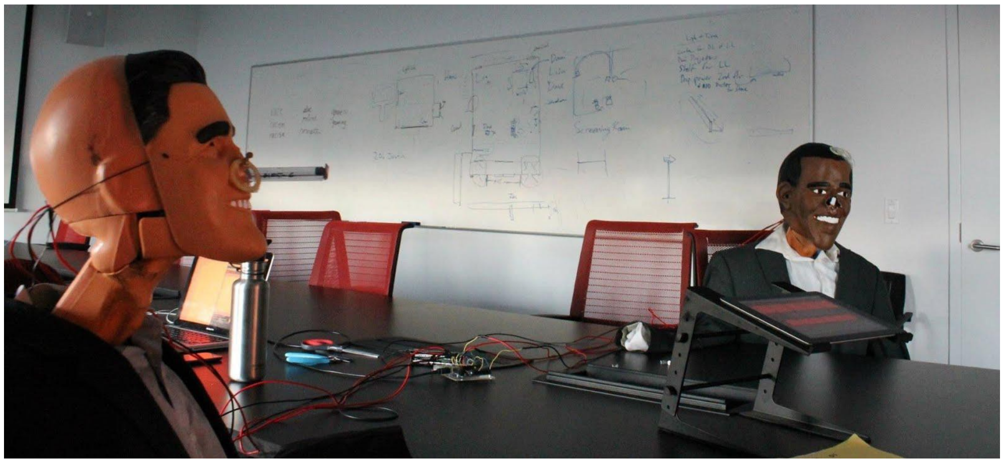
Justin Zupnick
DANM 219
Final Paper

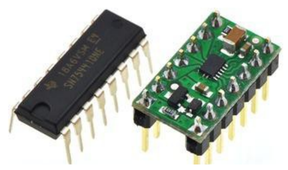
Creative Inspiration

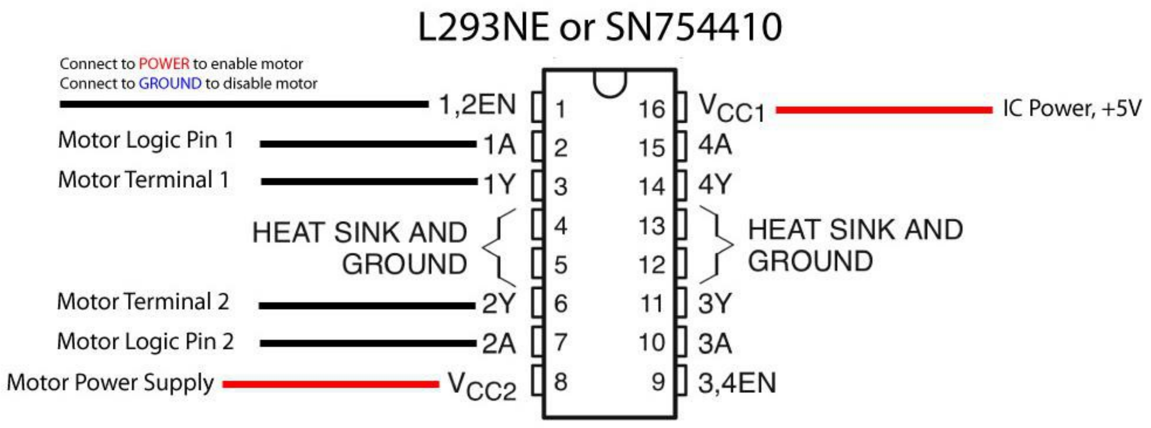
Liar, Liar was originally inspired by Romney pollster Neil Newhouse's reckless remark, "We're not going to let our campaign be dictated by fact-checkers," in response to criticism over a campaign ad deemed inaccurate by fact-checkers. With the extreme rise in "news" we are bombarded with, the recent influx of fact-checkers in the field of journalism should be more worthy of our attention. In order to shed light on the minute lies told throughout the 2012 presidential election and call forth attention to fact-checkers I decided it might be interesting to perform a re-enactment of the final presidential debates that mocks the candidates' facades. I chose to take on the Pinocchio metaphor as a means to physically realize the truths, half-truths and lies told during this last debate, with a growing nose that (ideally would have) pushed out a condom to express the dickish nature of our contemporary democracy. I also chose to make the piece interactive, with participants able to select soundbites. My thinking behind this was that it might capture the audience's attention greater than just having the motorized noses react to a time-based reproduction of the debate. I figured the audience may pay more attention if they were actively selecting sound bites themselves.

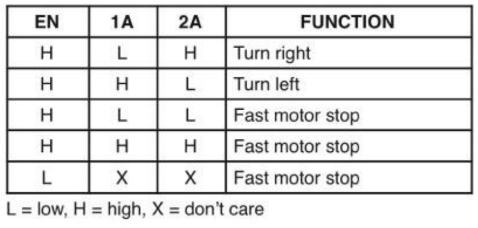


Technical Aspects

To make the noses move we'll need a small motor to move forwards and backwards. While a servo or linear actuator might have been more suitable, I ended up hacking this motorized linear slide potentiometer (see below left) with the use of a Texas Instruments SN754410 H-Bridge (see below right) in order to have the ability of reversing, not one, but two of these DC motors. Please refer to the H-bridge layout below (taken from NYU ITP'S DC Motor Control Using An H-Bridge) Wires need to be connected from the DC motors to motor terminals 1 and 2 on the H-bridge, which coincide to pins 3 & 6 for motor 1 and pins 11 & 14 for motor 2.



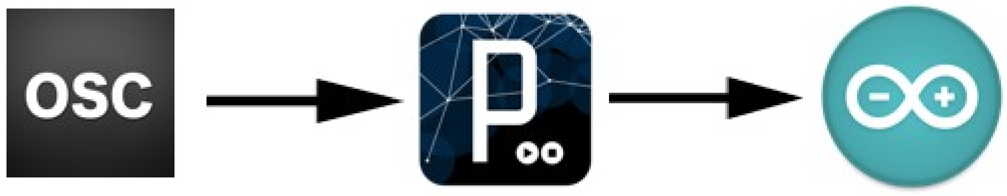




EN	1A	2A	FUNCTION
H	L	H	Turn right
H	H	L	Turn left
H	L	L	Fast motor stop
H	H	H	Fast motor stop
L	X	X	Fast motor stop

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

Now we need to make a whole bunch of things communicate. More specifically, an iPad interface to audio speakers (for the sound bites) and those motors. We'll be using TouchOSC to communicate the iPad to Processing, which will, in turn, communicate to the Arduino.



TouchOSC is an application that uses Open Sound Control, a messaging protocol typically used between networked multimedia devices to control audio applications and relay MIDI messages, but we can customize it for our purposes. TouchOSC Editor will be required to create our own interface and then upload it to the iPad. Both TouchOSC and TouchOSC Editor can be downloaded here. Additionally, the oscP5 library (downloadable here) for Processing will need to be installed. Make sure both IP addresses and proper incoming and outcoming ports are reported properly.

Having set up OSC to Processing, it's now imperative that we send Processing the proper messages to parse and organize in order to make stuff happen. These messages can be customized within TouchOSC Editor. I've set a messaging system with three parts to decode. Processing takes the incoming OSC message and converts it to a string in order to be parsed apart. The first part to be parsed is which candidate/motor has been selected (o = Obama, r = Romney). The second part is which direction the motor will move (tru = backwards, lie = forwards, mid = midway). The last part is a number, which coincides to the audio file to be play upon selection. I've combined the first two messages into a one single message (noseDir) that will be sent to the Arduino using the serial library (via truthVal). For audio, I used the minim library in Processing in order to play the chosen sound bites. It's important to pause all other audio files and rewind the selected audio file in order to avoid overlapping sounds and a file that begins midway from the last time it was selected.

The motor logic and enable pins are appropriately hooked into the pulse-width modulation output pins on the Arduino Uno. Speed can be controlled by using analogOutput, instead of digitalOutput, and adjusting the PWM out to the desired speeds (0-255). It's best to control the speed through the enable/disable pin, rather than getting confused by the relationship between two PWM outs working together through the motor logic pins.

As you can see in the code, the Arduino is setup to read incoming serial data. The Arduino program checks to see if there is any data coming in. If serial data is coming in and is greater than 0, the program then switches in and out the incoming bytes coming from Processing and then acts accordingly to the different cases that have been set up, respectively moving the selected motor and proper direction. For pulsing in the correct direction, see the H-bridge layout reference above for the proper PWM outputs that are given above for each pin.

One thing that could have used work in the code is how I stopped the motor. My motorStop function was originally set up to pulse a high output to the enable pin, while setting the same outputs to both motor pins. What this does is actually hard stop the motor, creating a difficult time for the motor to turn. Thus, if I were to try and move the linear pot back and forth it would be extremely difficult. This may have been where I ran into significant difficulties with motor direction. A better approach would have been to turn the enable pin off, to a low output, essentially disconnecting the motor from the circuit, allowing the linear pot to be pushed.