

MADELEINE WEAVER

Engineering
Portfolio

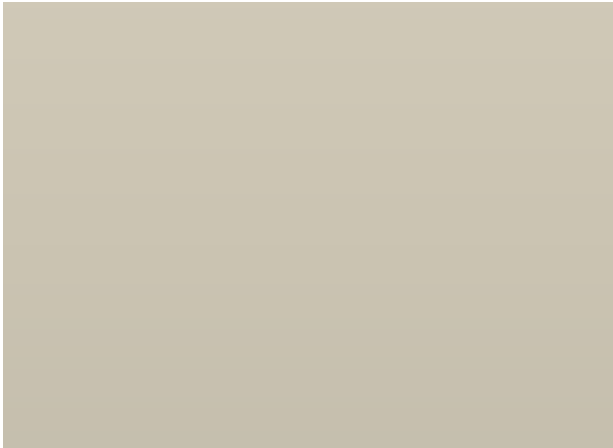
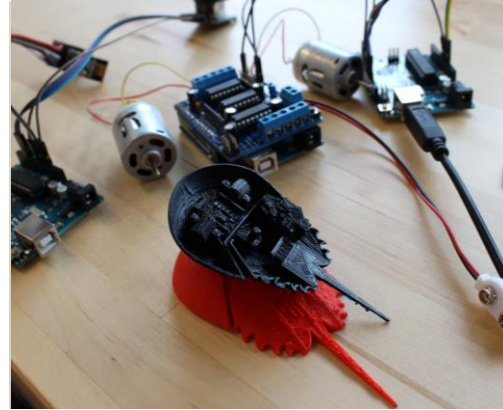


TABLE OF CONTENTS

Augmented Reality
Lever
~3~

Horseshoe Crab
Robot
~9~

Streetlight
~15~



AUGMENTED REALITY LEVER

Research Experience for Undergrads
University of Southern California MxR Lab
(Summer 2018)

Project Overview

The challenge our team of three was tasked with was to build an augmented reality environment in which a person would be faced with a moral dilemma, a version of the trolley problem, and have the option to pull a physical lever. The environment would be used to determine the effect different immersive factors have on user response as it pertains to moral decision-making.

My unique contribution to the project was the fabrication of the physical lever. The lever needed to be built so the amount of pressure it would require to pull the lever could be changed in discreet values. The physical lever also needed to have a corresponding virtual object that could represent the movement of the physical lever in the virtual environment.

As of March 18th, 2019 a poster submission for this project is under review for the IEEE VR conference, and I have been included as a co-author.

ADMIT

Advanced Decision Making in Immersive Training

Jake Chanenson

Peter Cowal

Maddy Weaver

Our team is developing a simulation for a study to measure the influence of immersive factors on trainees' decision making ability in virtual reality.

Scenario: The Trolley Problem

A trolley is heading towards five workers on a track. There is a switch that diverts the trolley to a track with only one worker.

Do you decide to pull a lever to divert the trolley onto the track with fewer people?

Our Simulation

We use the trolley problem as a case study for measuring the effectiveness of immersive training. Our simulation presents participants with an ethical dilemma in virtual reality and tasks them with deciding whether or not to pull a real-world lever, represented by a virtual lever with matching position.

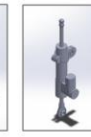
The Hardware



HTC Vive With Leap Motion Controller



3D Modeled Railroad Switch

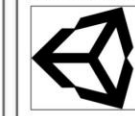


Model of Motorcycle Damper



Damper in Place

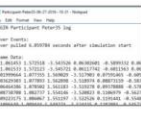
The Software



Simulation Built in Unity 3D



LeapMotion Hand Tracking



Automatic Data Collection

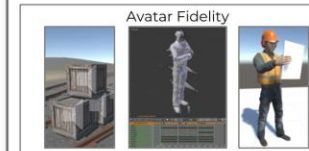
Immersive Factors



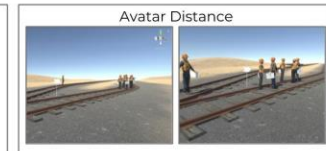
Lever Resistance



User Perspective



Avatar Fidelity

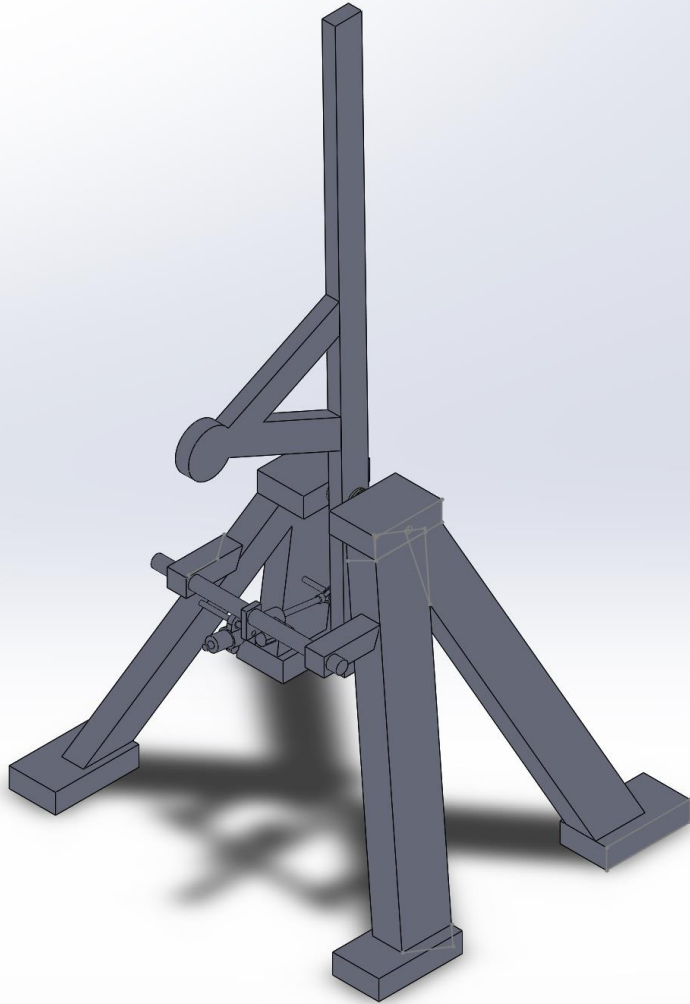


Avatar Distance



Railroad Switch in Last Stages of Completion

- Damper employed for variable resistance
- Damper placement and mechanics verified in virtual simulations
- Mount for HTC Vive Tracker for data collection



Design

I chose to design the lever in SolidWorks to take advantage of the software's motion simulation features, and because I could download part models of hardware from McMaster Carr's website. This made both generating a bill of materials and modification simple.

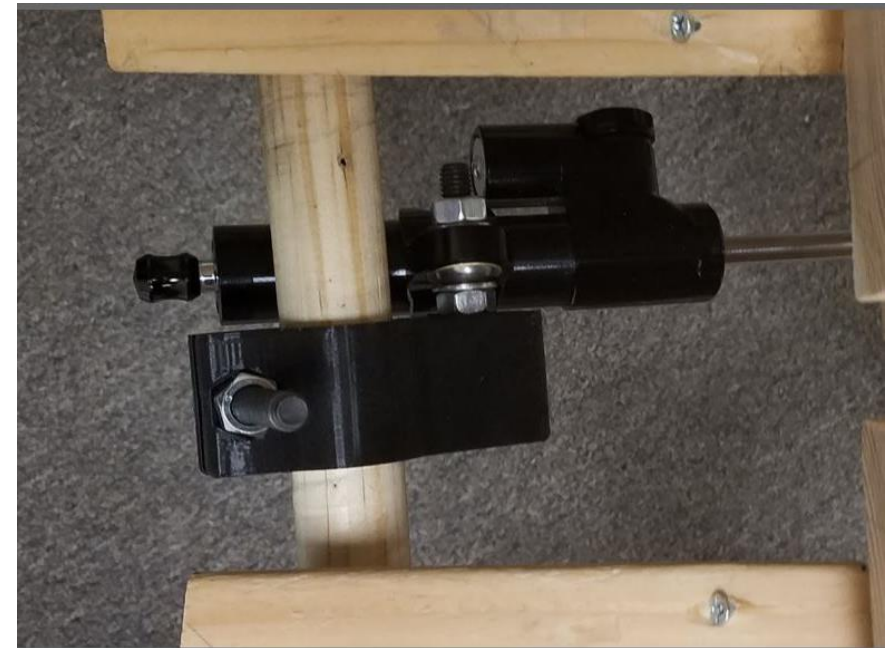
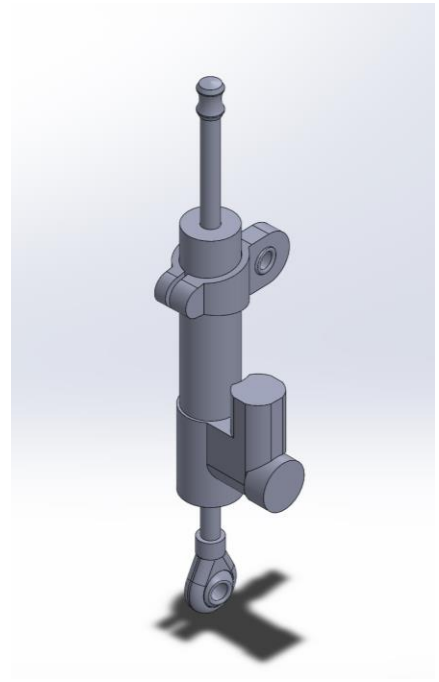
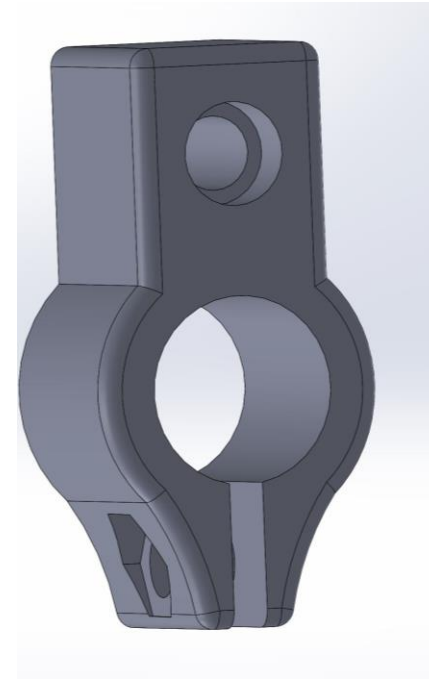
I used the model of the lever to determine the optimal placement of the hardware I chose to control the lever's resistance.

Hardware

I chose to use a motorcycle steering damper to give the lever its variable resistance. The damper had a dial that allowed us to adjust the resistance and assign a discreet value to it that could be repeated for each trial.

I modeled the damper in SolidWorks with its true-to-life motion qualities so that I could ensure it was mounted in such a way that the force of the lever was directed through the damper, and not toward spinning the damper at either of its two connections.

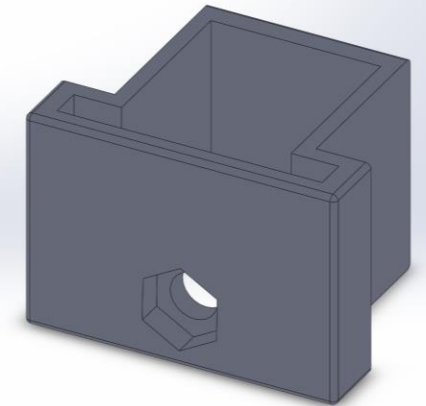
I also designed and printed a clamping bracket to clamp the damper to the frame of the lever.



Hardware

I designed and 3D printed a second bracket to mount a Vive Tracker to the lever. Two iterations of this bracket design were used to sandwich the top and bottom of a tripod plate, which was screwed to the Tracker. They were then bolted in place.

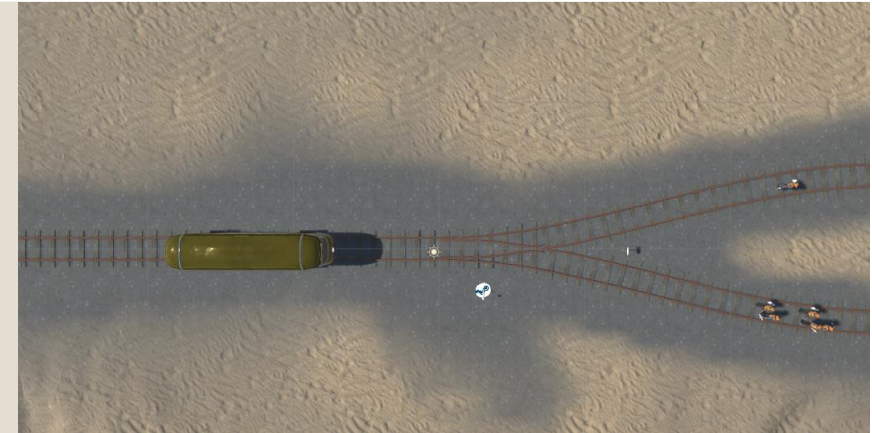
The Vive Tracker provided positioning data to enable the virtual representation of lever to synchronize with the physical one.



Virtual Environment

Our human models were full-body 3D scans of previous interns. To enhance immersion the figures were given helmets and ear protection, which I modeled in Blender, to provide the user with an explanation as to why they would not hear the train coming and save themselves.

Working on an interdisciplinary project gave me the opportunity to develop new skills. In helping to complete the project, I learned to make videogame objects, rig and animate human figures, and assemble individual models into the Unity game engine.





HORSESHOE CRAB ROBOT

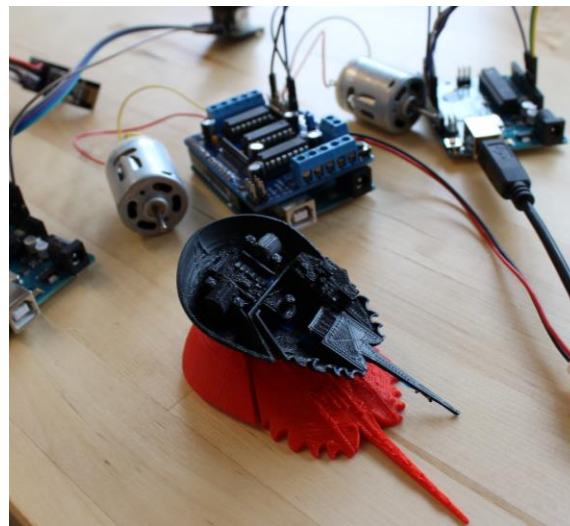
Fab Lab Intern
Dassault Systemes
(Spring 2018)

Project Overview

My Horseshoe Crab Robot was a project to create a remote-controlled robot that could flip itself over should it be inverted. I designed the shell to look like a horseshoe crab, because a horseshoe crab uses its tail to flip itself over.

I decided to place a 'soft robot' actuator in a cavity between the tail and the body of the robot to empower the robot to raise its tail. A soft robot is an inflatable silicone actuator that can direct actuation either laterally or angularly, depending on the shape of the actuator and the thickness of its walls.





Shell

The most challenging part of designing the shell was accomodating all the hardware components inside of it. Achieving this required several different design iterations and modeling all the components in SolidWorks to ensure a good fit.

Designing the moving joints was also a challenge, this being my first attempt at designing a hinge.

```

motorshieldcode | Arduino 1.8.5
File Edit Sketch Tools Help

motorshieldcode §

/*-----( Import needed libraries )-----*/
#include <AFMotor.h>

AF_DCMotor motor1(3);
AF_DCMotor motor2(4);

byte XPWM_PIN = A0;

int Xpwm_value;

int XoutputValue = 0;
int XsensorValue = 0;
int XpinValue = 0;

byte YPWM_PIN = A1;

int Ypwm_value;

int YoutputValue = 0;
int YsensorValue = 0;
int YpinValue = 0;

int wheelSpeed = 0; // variable to store the servo position
int backwheelSpeed = 0;

void setup()

```

Motor Shield Code

```

Radio_Joystick_Transmit | Arduino 1.8.5
File Edit Sketch Tools Help

Radio_Joystick_Transmit

/*-----( Import needed libraries )-----*/
#include <RF24.h> // Comes with Arduino IDE
#include "RF24.h" // Download and Install (See above)
#include "printf.h" // Needed for "printDetails" Takes up some memory
/*-----( Declare Constants and Pin Numbers )-----*/
#define CE_PIN 7 // The pin to be used for CE and SN
#define CSN_PIN 8

#define JOYSTICK_X_A0 // The Joystick potentiometers connected to Arduino
#define JOYSTICK_Y_A1 // The Joystick push-down switch, will be used as
#define JOYSTICK_SW_A2

/*-----( Declare Objects )-----*/
// Hardware configurations set up an RF24(RF) radio on SPI bus pins (usually)
RF24 radio(CE_PIN, CSN_PIN);

/*-----( Declare Variables )-----*/
byte addresses[16] = {"1Node", "2Node"}; // These will be the names of the
unsigned long timeout; // Used to grab the current time, calculate delays
unsigned long started_waiting_at;
boolean timeout; // Timeout? True or False

// Allow testing of radio and code without Joystick hardware. Set 'true'
//boolean hasHardware = false;
boolean hasHardware = true;

```

Joystick Radio Transmitter

```

recievetest | Arduino 1.8.5
File Edit Sketch Tools Help

recievetest

/* radio reciever code
1 - GND
2 - 5V
3 - CE to Arduino pin 7
4 - CSN to Arduino pin 8
5 - SCK to Arduino pin 13
6 - MISO to Arduino pin 11
7 - MISO to Arduino pin 12
8 - UNUSED

- V2.12 02/08/2016
- Uses the RF24 Library by TMRH20 and Maniacbug: https://github.com/TMRh
Questions: terry@yourduino.com */

/*-----( Import needed libraries )-----*/
#include <SPI.h> // Comes with Arduino IDE
#include "RF24.h" // Download and Install (See above)
#include "printf.h" // Needed for "printDetails" Takes up some memory
// NEED the SoftwareServo library installed
// http://playground.arduino.cc/uploads/ComponentLib/SoftwareServo.zip
#include <SoftwareServo.h> // Regular Servo library creates timer conflict
/*-----( Declare Constants and Pin Numbers )-----*/
#define CE_PIN 7 // The pins to be used for CE and SN
#define CSN_PIN 8

#define MY_SOFTSPI
#define MY_SOFT_SPI_SCK_PIN 13

```

Radio Receiver Code

Programming

The project necessitated three different Arduinos. There was one to transmit the radio signal and control the robot, one to receive the signal and one to control the motors. The two Arduinos within the robot were necessary because the pins that were used by the motor shield were also required for the radio receiver.

Part of the receiver code is dedicated to transmitting the analog joystick data to the motor shield through a PWM pin. This would not work until I tied together the ground pins for both Arduinos; this part of the project took a great deal of troubleshooting to accomplish.



Soft Robot Actuator

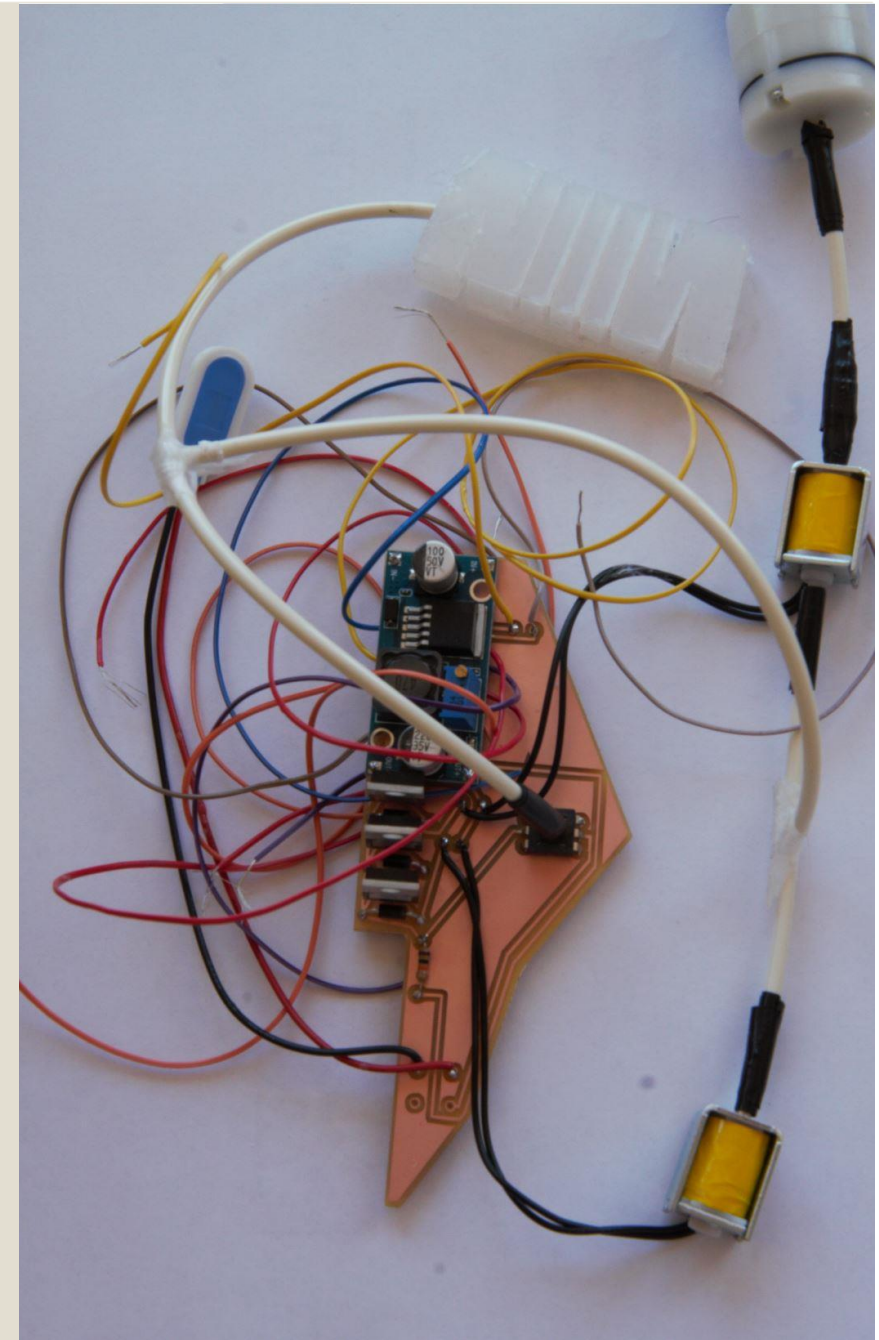
Pictured are three different iterations of my soft robotic actuator. I was able to download one 'zig-zag' and one 'curled' mold from the Soft Robotics Toolkit. When inflated, the 'zig-zag' model actuates linearly and the 'curled' model actuates angularly.

The iteration that worked best was a composite of both the linearly and angularly actuating models. In order to lift the tail, the actuator needed to expand linearly to fill the space freed by the lifting of the tail, and angularly around the axis of the hinge.

Electrical Design

I milled my PCB on a tabletop mill and used design tools in SolidWorks to cut the outside edges to fit within the amorphous shell. The hardware included 3 transistors to complete a circuit between the power supply and either the pneumatic pump or the two solenoid valves depending on whether it receives a digital signal from the Arduino controlling it.

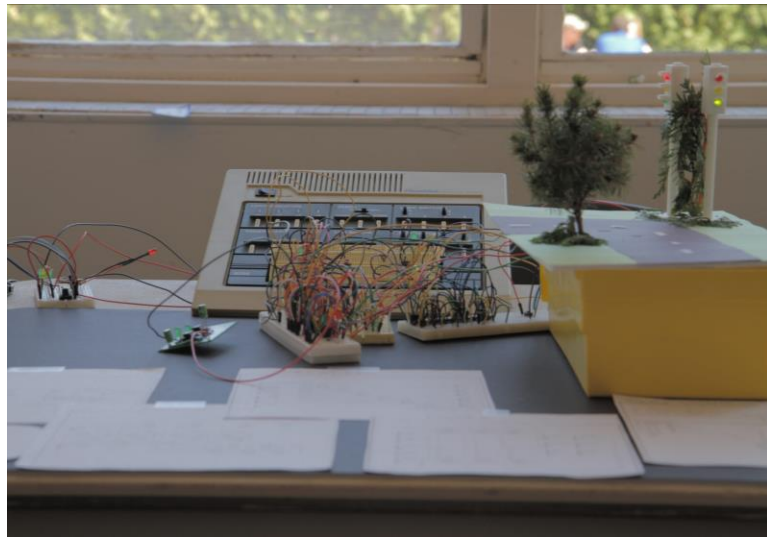
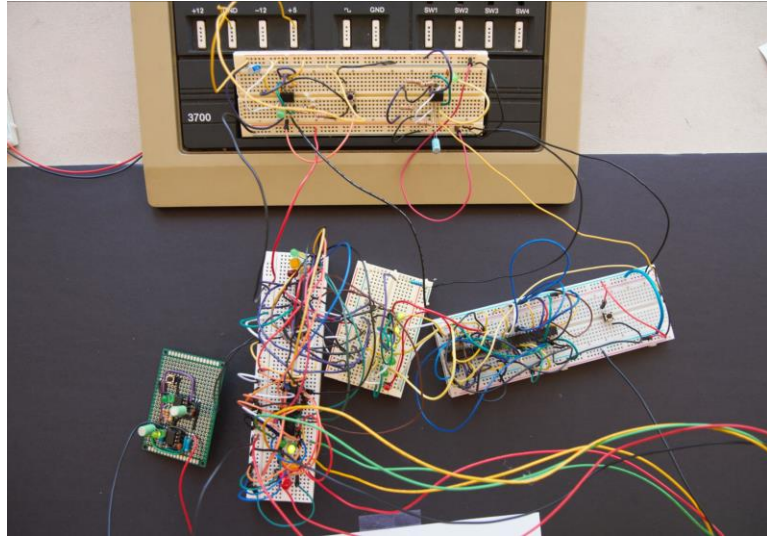
There was also a pressure sensor, which was used to gather air pressure data from inside the actuator, and either open the release valve to deflate the actuator or close it and activate the pneumatic pump to inflate the actuator. This way, the actuator would never become over-filled or under-filled.





STREETLIGHT

Digital Electronics Class Project
Massachusetts Bay Community College
(May 2018)



Project Overview

The goal of this project was to demonstrate a practical understanding of digital circuitry by making a model streetlight.

To simulate a car approaching a two-way intersection which would trigger the streetlight to change, we pressed a pushbutton. The green light would go to orange then red, while the red light went to green. The lights would then revert to their starting position.

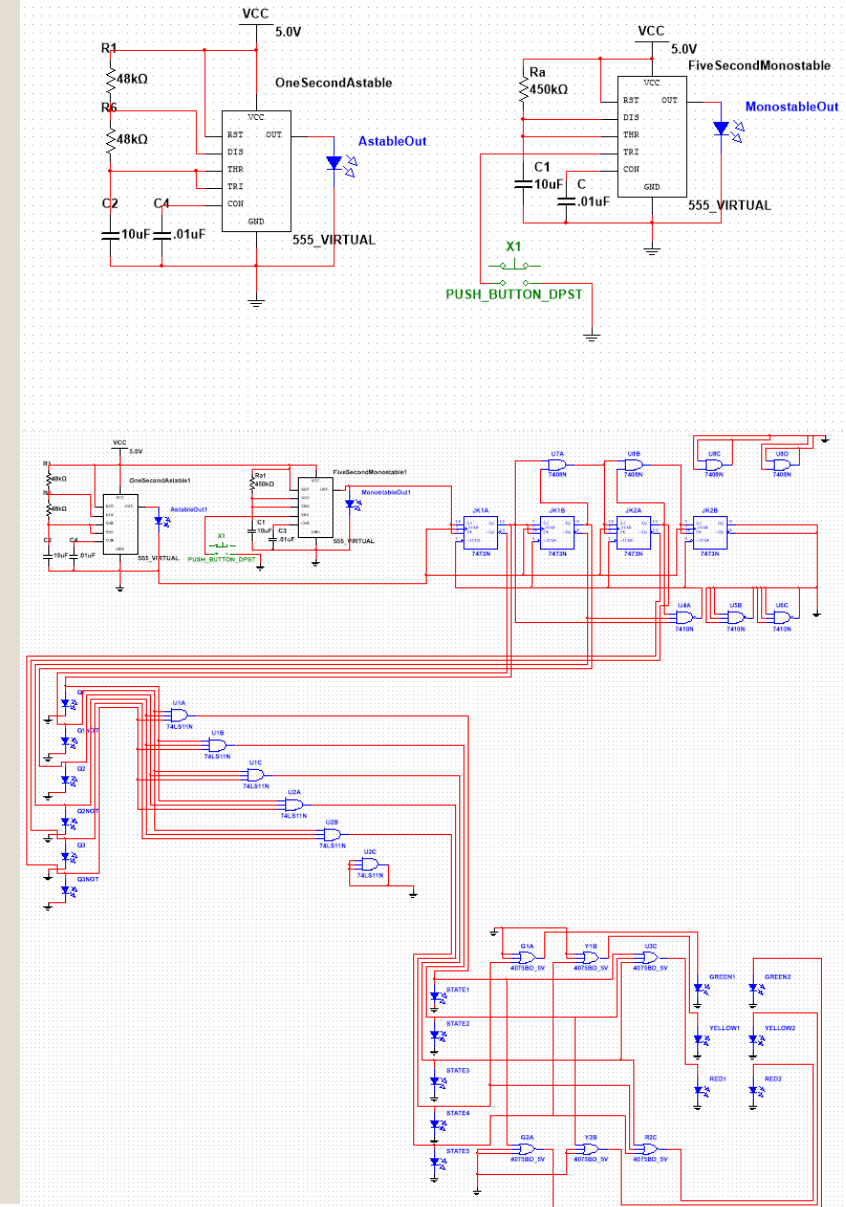
The circuit included both an astable and monostable 555 circuit, a JK Flip Flop binary counter, a decoder, and a series of OR gates allowing all the lights to change independently of one another.

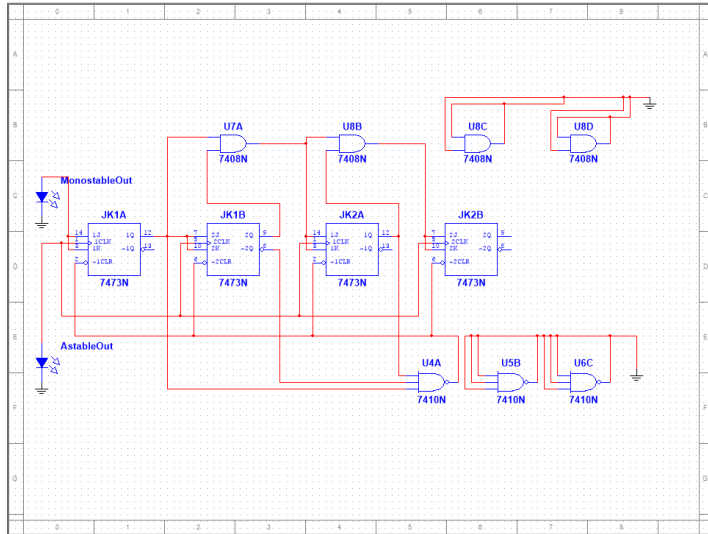
555 Timers

The first stage of the circuit are the 555 Timers. I wanted each state to last about one second, so I selected the resistor and capacitor values so that the timer had a period of one second.





Because there were 5 different states of excitation, the monostable timer was set to be high for a period of 5 seconds after the pushbutton had been pressed.

The output of the astable was then connected to the clock input of the JK Flip Flops, and the monostable output was connected to the J and K input of the first Flip Flop, enabling the counter to begin counting.





FUNCTION TABLE

INPUTS				OUTPUTS	
CLR	CLK	J	K	Q	\bar{Q}
L	X	X	X	L	H
H		L	L	Q_0	\bar{Q}_0
H		H	L	H	L
H		L	H	L	H
H		H	H	TOGGLE	

MOD 5 Binary Counter

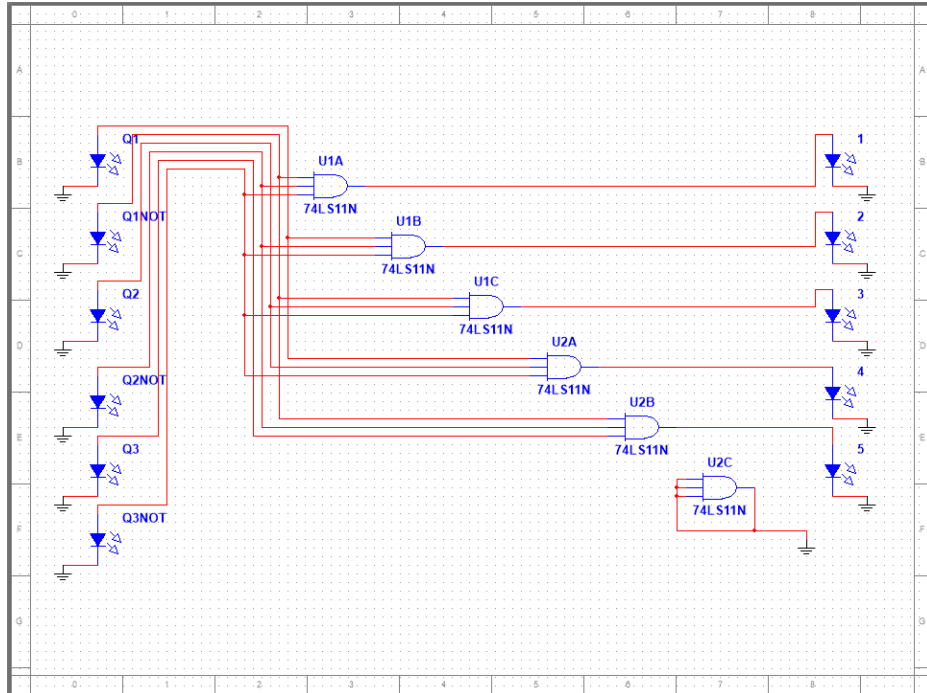
Because the CLR input needed to be high for the chip to operate, there is a NAND gate in place that resets the chip when the counter reaches 5.

The outputs of the first and second JK Flip Flops are ANDed together, so that the state after the first two Flip Flops are high will be a high third Flip Flop, and so on.

Decoder

This step is necessary to give one discrete high output at each of the five stages.

The decoder changes binary counting to single outputs that represent each stage.



OR Gates

This stage was important so that the LED lights could operate completely independent of each other.

Because at stage one, both a green and a red light were on, if this stage was not included, a stage where a yellow light and the aforementioned red light were high but the green light were low would be impossible.

The lights were mounted in small model streetlights that I designed and 3D printed myself.

