Animatronic Hand

Created by John Park

https://learn.adafruit.com/animatronic-hands

Last updated on 2023-08-29 03:42:58 PM EDT
Table of Contents

Overview 3
- Parts
- Materials & Supplies
- Tools

Build the Animatronic Hand 4
- Make the Hand
- Hinge the Joints
- Add Straw Channels
- Add Cords
- Rubber Band Return Tendons
- Mount the Hand
- Mount the Servo Motors
- Thread the Cords
- Run wires to CRICKIT

Code the CPX and CRICKIT for Puppeteering 30
- Setup the CPX and CRICKIT
- CircuitPython Code
- Library Imports
- Slide Switch
- Servo Setup
- Capacitive Touch Setup
- Main Loop
- Switch Check
- Read the Capacitive Pads, Move Servos

Puppeteer the Animatronic Hand 34
- Add Power
- Cap Touch

Audio-Animatronic Binary Counting 39
- CPX Setup
- Audio Files
- CircuitPython Code
- Code Explainer
- Servo Counting
- Counting Time
Overview

Build your own animatronic hand using cardboard, straws, some cord, and rubber bands. Pull the cords to actuator fingers -- or even better, let some servo motors do it for you!

CRICKIT is the perfect platform for building your animatronics projects -- it can take commands from your Circuit Playground Express and then power your servos. You can even add lights and sound later if you like.

Parts

1 x CRICKIT for Circuit Playground Express
Robot Friend

1 x Circuit Playground Express
Round, Awesome Microcontroller Board

4 x Standard servo
TowerPro SG-5010

1 x Waterproof 3xAA Battery Holder
with On/Off Switch

1 x Male DC Power adapter
2.1mm plug to screw terminal block

1 x Alkaline AA batteries
Materials & Supplies

In addition to the parts above, you'll need:

- Corrugated cardboard from a shipping box or two
- Drinking straws
- Cord, yarn, or string
- Rubber bands

Tools

- Hobby knife or scissors
- Hot melt glue gun and hot melt glue sticks

Build the Animatronic Hand
Make the Hand

Align a piece of cardboard so that the "grain" of the corrugated flutes is oriented vertically.

Use a pencil to trace your hand.

With a hobby knife, cut out the pattern of your hand.
Hinge the Joints

With a pencil, mark the locations of each finger joint
Gently score each mark with your knife to aid in bending
Carefully bend each joint to give the fingers a natural curl and to guide them later when they're pulled with a cord by the servos
Add Straw Channels

Cut small segments of drinking straws to length so that they will fit on each finger segments without colliding with their neighboring segments when bent
Using the hot melt glue gun, secure each segment in place
Add Cords

Cut four lengths of cord, string, or yarn to about 18" each
Thread each cord through the straw of its respective finger
Fold the tip over and glue it in place from the back
Once the glue has cooled sufficiently, test the pull of each string -- they should easily curl each finger
Rubber Band Return Tendons

Cut each rubber band into a single strip
Glue one end of each band to a finger tip on the back side of the finger
Using a pencil, poke a hole through the back of the hand at the base of each finger
Push the rubber band end through the hole, then pull it taut enough to give the finger a good return action, but not so tight that it can't be bent by pulling the cord
Tie off each band in the palm area
Mount the Hand

Position the hand at one edge of the box
Mark two spots to poke holes in the hand and the box side
Poke the holes through with the Make Do tool
Screw in two screws to hold the hand in place
Mount the Servo Motors

Lay out the servos so there will be enough clearance that the servo horns don't collide.
Mark positions for each servo -- keep the marks a bit smaller than the dimensions of the servo so the fit will be tight and hold them in place.
Cut the holes out with the hobby knife.
Feed the wires into the box, then press the servos in place.
Thread the Cords

Orient the shafts fully counter-clockwise
Lift of the servo horns and then reposition them on the shafts so the arm you’ll connect to the cord is at the one o’clock position
Thread a cord through the end hole on that arm for each servo
Pull the cord taut, but not to much -- there should be a slight bend to each finger
Wrap off the cords around the servo horns like you would a cleat hitch -- a figure eight pattern where the final loop wraps the tail underneath the cord so it doesn't come loose
Run wires to CRICKIT

Position the CRICKIT on the box so there will be enough clearance to use the capacitive touch pads.
Mark a position on the box near the servo connectors to run the wiring.
Cut out the hole, then pull the wires up through.
Plug in the servos -- the yellow signal wires should be at the outer edge of the board.
Code the CPX and CRICKIT for Puppeteering

Setup the CPX and CRICKIT

The Circuit Playground Express (CPX) paired with the CRICKIT is a powerful, yet simple to use combination for building animatronics! To get started, you'll want to set up the CPX for use with CircuitPython by following this guide. When you’re ready, and can upload code to the board return here.

To use the CRICKIT with the CPX, follow the steps listed here to install the special build of CircuitPython, as well as the latest library bundle.

CircuitPython Code

Puppeteering Mode

A really fun, satisfying way to code the animatronic hand is in an interactive, puppeteering mode. We'll set things up so when you touch any of the four capacitive touch pads on the CRICKIT, the associated servo will rotate and actuate its connected finger.

Adafruit suggests using the Mu editor to edit your code and have an interactive REPL in CircuitPython. You can learn about Mu and installation in this tutorial.
The full code is included below -- you can copy and paste that into Mu and save it to your CPX to get it all working. After the full code is an explanation of how the various code snippets work.

```python
# SPDX-FileCopyrightText: 2018 John Edgar Park for Adafruit Industries
#
# SPDX-License-Identifier: MIT

# Animatronic Hand
# CPX with CRICKIT and four servos
# touch four cap pads to close the fingers

import board
from digitalio import DigitalInOut, Direction, Pull
from adafruit_crickit import crickit

DOCUMENTATION

# CPX switch
# use the CPX onboard switch to turn on/off (helps calibrate)
switch = DigitalInOut(board.SLIDE_SWITCH)
switch.direction = Direction.INPUT
switch.pull = Pull.UP

# 4 Servos!
servos = (crickit.servo_1, crickit.servo_2, crickit.servo_3, crickit.servo_4)
for servo in servos:
    servo.angle = 180  # starting angle, open hand

# 4 Touch sensors!
touches = (crickit.touch_1, crickit.touch_2, crickit.touch_3, crickit.touch_4)
cap_state = [False, False, False, False]
cap_justtouched = [False, False, False, False]
cap_justreleased = [False, False, False, False]
curl_finger = [False, False, False, False]
finger_name = ['Index', 'Middle', 'Ring', 'Pinky']

while True:
    if not switch.value:  # the CPX switch is off, so do nothing
        continue
    # Check the cap touch sensors to see if they're being touched
    for i in range(4):
        cap_justtouched[i] = False
        cap_justreleased[i] = False
        if touches[i].value:
            #print("CT" + str(i + 1) + " touched!")
            if not cap_state[i]:
                cap_justtouched[i] = True
                print("%s finger bent." % finger_name[i])
                servos[i].angle = 0
                # store the fact that this pad is touched
                cap_state[i] = True
            else:
                if cap_state[i]:
                    cap_justreleased[i] = True
                    print("%s finger straightened." % finger_name[i])
                    servos[i].angle = 180
                    # print("CT" + str(i + 1) + " released!")
                    # store the fact that this pad is NOT touched
                    cap_state[i] = False
        if cap_justtouched[i]:
            curl_finger[i] = not curl_finger[i]
```

©Adafruit Industries
Here are the different sections of the code, and how they work.

Library Imports

First, we'll import a few libraries. `DigitalInOut`, `Direction`, and `Pull` are imported from `digitalio` in order to allow us to read the slide switch built into the CPX.

The `crickit` library is imported from `adafruit_crickit` to allow us to use the CRICKIT's onboard seesaw co-processor to read capacitive pads, to write out command signals to servos, and to communicate over the SDA and SCL pins between the CPX and the CRICKIT.

We also import the `board` library, which gives us convenient definitions for objects on the CPX that we can call.

```python
import board
from digitalio import DigitalInOut, Direction, Pull
from adafruit_crickit import crickit
```

Slide Switch

We'll set up the slide switch built onto the CPX, name it `switch`, setting its direction to `INPUT`, and using the built-in pullup resistor.

```python
switch = DigitalInOut(board.SLIDE SWITCH)
switch.direction = Direction.INPUT
switch.pull = Pull.UP
```

Servo Setup

We'll created a variable list called `servos` that sets up our four servos. These are called out by their CRICKIT servo port names `(1, 2, 3, 4)`. They are set to be controlled with pulse width modulation (PWM) with a PWM frequency of 50, and then moved to an angle of 180.

```python
servos = (crickit.servo_1, crickit.servo_2, crickit.servo_3, crickit.servo_4)
for servo in servos:
    servo.angle = 180 # starting angle, open hand
```
Capacitive Touch Setup

Next, you'll set up the capacitive touch pads. You'll create some variable lists to help with the logic of registering touches to the cap pads and moving the servos when pads are touched or released.

We also have a variable list with the friendly names of the fingers to use when debugging with print statements.

```python
touches = (crickit.touch_1, crickit.touch_2, crickit.touch_3, crickit.touch_4)
cap_state = [False, False, False, False]
cap_justtouched = [False, False, False, False]
cap_justreleased = [False, False, False, False]
curl_finger = [False, False, False, False]
finger_name = ['Index', 'Middle', 'Ring', 'Pinky']
```

Main Loop

With the setup complete, now we get into the bulk of the program, which runs over and over. This is everything that follows the `while True:` line.

Switch Check

The first thing that happens is we check the state of the CPX slide switch with the `if not switch.value:` line. In CircuitPython, this is a concise way of saying "if the switch value is 0 (which is the reading when the switch is to the right), do nothing and continue". When the switch is to the left, the value is 1 and we to proceed with the rest of the code.

Read the Capacitive Pads, Move Servos

Here we'll loop through four times, checking each capacitive pad to see if its value is above the touch threshold.

When the values read are greater than the threshold, we check to see if we have already been touching that pad by querying the value of the `cap_state[]` variable. By setting this state to change when the cap pads are released, we can have the servos curl the fingers when pads are first touched, hold them in a curl while being touched, and uncurl them when the pads are released.
# Check the cap touch sensors to see if they're being touched
for i in range(4):
    cap_justtouched[i] = False
    cap_justreleased[i] = False
    if touches[i].value:
        #print("CT" + str(i + 1) + " touched!")
        if not cap_state[i]:
            cap_justtouched[i] = True
            print("%s finger bent." % finger_name[i])
            servos[i].angle = 0
            # store the fact that this pad is touched
            cap_state[i] = True
        else:
            if cap_state[i]:
                cap_justreleased[i] = True
                print("%s finger straightened." % finger_name[i])
                servos[i].angle = 180
                #print("CT" + str(i + 1) + " released!")
                # store the fact that this pad is NOT touched
                cap_state[i] = False
                if cap_justtouched[i]:
                    curl_finger[i] = not curl_finger[i]

That's the full code breakdown for the interactive mode of the animatronic hand -- now we'll power up the system we can have fun playing with it!

Servo motors can draw a lot of current! It's possible that your CPX will restart if you draw too much current by jamming on all four cap touch pads at once, causing all four servos to move at the same moment.

Puppeteer the Animatronic Hand

Now, you're ready to use the hand in interactive mode by pressing the capacitive touch pads to actuate each finger.

Add Power

Plug in your power supply to the CRICKIT -- this needs to be a 5V DC source that can deliver from 2-10 amps, with a 2.1mm center positive barrel plug. The CPX will be powered by the CRICKIT, so no need to plug it in separately.
Cap Touch

Press the CRICKIT's capacitive touch pads to curl the fingers!
Audio-Animatronic Binary Counting

Here's another fun way to use your animatronic hand -- it can count up to the number 15 (and say the numbers out loud)! How's this possible with only four active fingers? By using a modified finger binary counting method.

Finger binary is a counting method where each finger is assigned a 0 (down) and 1 (up) state, each with a different power of two.

These will be the powers of two and values of each finger of our modified, four-finger binary counting system:

- Index is $2^0$, in binary 0001, or a value of 1
- Middle is $2^1$, in binary 0010, or a value of 2
- Ring is $2^2$, in binary 0100, or a value of 4
- Pinky is $2^3$, in binary 1000, or a value of 8

By holding up the Index (1) and Middle (2) fingers, the binary representation is 0011. We add their values, 1 + 2, therefore we indicate the number 3.

Here's another one: Pinky(8) + Ring(4) is 1100 in binary, or a value of 12.
So, here's the pattern for counting from 1 to 15 on four fingers:

- 1 = Index 0001
- 2 = Middle 0010
- 3 = Index, Middle 0011
- 4 = Ring 0100
- 5 = Ring, Index 0101
- 6 = Ring, Middle 0110
- 7 = Ring, Middle, Index 0111
- 8 = Pinky 1000
- 9 = Pinky, Index 1001
- 10 = Pinky, Middle 1010
- 11 = Pinky, Middle, Index 1011
- 12 = Pinky, Ring 1100
- 13 = Pinky, Ring, Index 1101
- 14 = Pinky, Ring, Middle 1110
- 15 = Pinky, Ring, Middle, Index 1111

**CPX Setup**

Below is the CircuitPython code we'll use to count in binary on the fingers.

You'll still want all 4 servos attached. But now you'll also want a speaker on the speaker port!

![CircuitPython setup diagram]

CRICKIT_servos_speaker.fzz
Audio Files

The CRICKIT is a terrific board for animatronics projects, particularly audio-animatronics projects, because it can play back audio as well as control servos and motors. Download this .zip file and uncompress it to get a collection of audio .wav files we'll use during number counting.

We'll use an external speaker to hear things loud an clear. Screw the leads of your $4\Omega$ or $8\Omega$ speaker to the two speaker terminals on the CRICKIT -- orientation does not matter, so either wire can go to either terminal.

[Image]

Once you've uncompresses the .zip file, drag the .wav files onto your CIRCUITPY drive. The files can live at the root level of the CPX.

For more on audio playback with CRICKIT, check out this guide.

CircuitPython Code

Download this code and paste it into Mu. Then, save the file as main.py or code.py to your Circuit Playground Express. Make sure the CPX switch is to the left and sit back to watch and listen as your CRICKIT-powered animatronic hand counts in binary!

```
# SPDX-FileCopyrightText: 2018 John Edgar Park for Adafruit Industries
#```
Animatronic Hand
Binary Counting on four fingers up to 15

```python
import time
from digitalio import DigitalInOut, Direction, Pull
import audiioio
import audiocore
import board
from adafruit_crickit import crickit

------------- CPX switch
# use the CPX onboard switch to turn on/off (helps calibrate)
switch = DigitalInOut(board.SLIDE_SWITCH)
switch.direction = Direction.INPUT
switch.pull = Pull.UP

------------- Audio setup
print("Let's count in binary.")
introfile = "intro.wav"

cpx_audio = audiioio.AudioOut(board.A0)
def play_file(wavfile):
    with open(wavfile, "rb") as f:
        wav = audiocore.WaveFile(f)
        cpx_audio.play(wav)
        while cpx_audio.playing:
            pass

------------- 4 Servos!
servos = (crickit.servo_1, crickit.servo_2, crickit.servo_3, crickit.servo_4)
for servo in servos:  # close the fist
    servo.angle = 0  # close the fingers
    print("Servo %d angle = 0" % (servos.index(servo)+1) )
    time.sleep(.2)

# Which servos to actuate for each number
counting = ( [3],
             [2],
             [3, 2],
             [1],
             [1, 3],
             [1, 2],
             [3, 2, 1],
             [0],
             [0, 3],
             [0, 2],
             [0, 3, 2],
             [0, 1],
             [0, 3, 1],
             [0, 2, 1],
             [0, 3, 2, 1]
           )

play_file(introfile)
while True:
    if not switch.value:
        continue
    # the CPX switch is on, so do things
    for servo in servos:  # close the fist
        servo.angle = 0  # close the fingers
        print("Servo %d angle = 0" % (servos.index(servo)+1) )
        time.sleep(.2)

    time.sleep(1)  # pause a moment
```

©Adafruit Industries
for i in range(len(counting)):
    # close all the counting fingers between numbers
    for servo in servos:
        servo.angle = 0  # close
        print("\t\t\tServo #%d angle 0" % (servos.index(servo)+1))
        time.sleep(0.3)
    print("Number #%d \tfingers: %s % (i+1, counting[i])")
    # open just the indicated fingers when counting
    for finger in counting[i]:
        servos[finger].angle = 180  # open
        print("\t\t\tServo #%d angle 180" % (finger+1))
        time.sleep(0.3)
    # say it!
    play_file(wavfiles[i])
    # hold for a bit of time
    time.sleep(0.3)
    print("...")

Code Explainer

Take a look at the code to get a sense of how it works. After importing libraries and setting up the CPX switch, we have an audio setup section.

First, we create a variable list called `wavfiles` that stores the names of all the counting audio files we'll use. We also make a variable called `introfile` that has the filename of the intro audio wav file.

Then, we instantiate the `audioio` command and define a process called `play_file` to open, play, and close the audio files specified.

What follows are partial code snippets from the full program above, you cannot run they by themselves, they're here just for explanation

```python
### Audio setup
print("Let's count in binary.")
wavfiles = ["one.wav", "two.wav", "three.wav", "four.wav", "five.wav", "six.wav", "seven.wav", "eight.wav", "nine.wav", "ten.wav", "eleven.wav", "twelve.wav", "thirteen.wav", "fourteen.wav", "fifteen.wav"]
introfile = "intro.wav"
cpx_audio = audioio.AudioOut(board.A0)
def play_file(wavfile):
    with open(wavfile, "rb") as f:
        wav = audiocore.WaveFile(f)
        cpx_audio.play(wav)
        while cpx_audio.playing:
            pass
```
Servo Counting

Next, we set up the four servos, and create a variable list of lists called `counting`. This specifies which finger or set of fingers is to be raised up for each number that we're counting. You can see, for example, that the number 1 raises the index finger only, while 15 raises all four fingers.

```
############### 4 Servos
servos = (crickit.servo_1, crickit.servo_2, crickit.servo_3, crickit.servo_4)
for servo in servos:
    servo.angle = 180 # starting angle, open hand

# Which servos to actuate for each number
counting = [
    [3],
    [2],
    [3, 2],
    [1],
    [1, 3],
    [1, 2],
    [3, 2, 1],
    [0],
    [0, 3],
    [0, 2],
    [0, 3, 2],
    [0, 1],
    [0, 3, 1],
    [0, 2, 1],
    [0, 3, 2, 1]
]
```

Counting Time

The final bit of setup is to play the intro wav file and then get into the main loop. Here we check the CPX switch and only proceed if it is set to the left position.

Then, we close all the fingers down into a fist and pause a moment.

```
play_file(introfile)

while True:
    if not switch.value:
        continue

    # the CPX switch is on, so do things
    for servo in servos:  # close the fist
        servo.angle = 0  # close the fingers
        print("Servo %d angle = 0" % (servos.index(servo)+1) )
        time.sleep(.2)
    time.sleep(1)  # pause a moment
```

Finally, we run through the `counting` list one item at a time and open only the fingers specified per entry in that list.
After all of the fingers needed for a particular number have been raised, we play back the appropriate wav file so the audio-animatronic can speak!

```python
for i in range(len(counting)):
    # close all the counting fingers between numbers
    for servo in servos:
        servo.angle = 0  # close
        print("\t\tServo #%d angle 0" % (servos.index(servo)+1))
    time.sleep(0.3)
    print("Number #%d \tfingers: %s" % (i+1, counting[i]))

    # open just the indicated fingers when counting
    for finger in counting[i]:
        servos[finger].angle = 180  # open
        print("\t\tServo #%d angle 180" % (finger+1))
    time.sleep(0.3)
    # say it!
    play_file(wavfiles[i])
    # hold for a bit of time
    time.sleep(0.3)
    print("...")
```