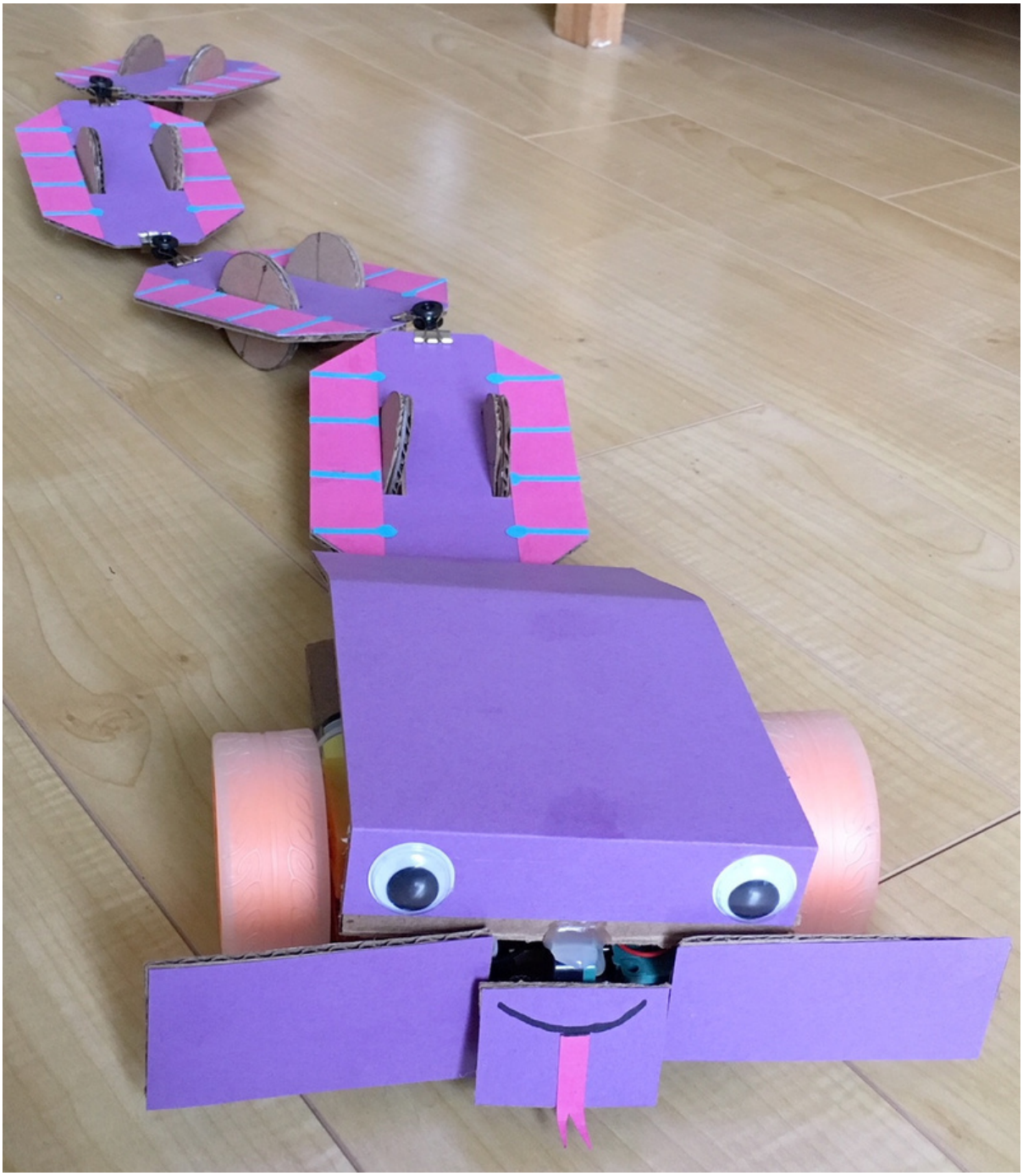
CRICKIT Snake Bot

Created by Dave Astels



https://learn.adafruit.com/crickit-snake-bot

Last updated on 2024-03-08 03:06:25 PM EST

Table of Contents

Overview 5

- Parts
- Tools and Supplies

Basic Design 8

- Code

Adding Bumpers 13

- Parts Required
- Wiring Additions
- The Whiskers
- Code
- Tack Movement
- Counting Attempts
- Handling 'nose' Bumps

Adding a Warning Buzzer 21

- Parts Required
- New Wiring Additions
- Code

A Bigger, Better Power Supply 25

- Code

Putting it All Together 28

Snake? What Snake? 33

- Cutting the pieces
- Marking the Pieces
- Assembling the Body Sections
- Blinka-fying

Overview

This project guides you through building a robot that can wander around your floor. This riffs on John Park's Crickit Carnival Bumper Bot (https://adafru.it/BXS). The difference is that the project progresses through several levels, starting with a very basic bot and adding capabilities.

Parts

Basic Design

You won't need the battery holders if you decide at the end to use a LiPo battery.

1 x Circuit Playground Express
https://www.adafruit.com/product/3333
ATSAMD21 based educational board, that's loaded with I/O devices and CircuitPython capable.

1 x CRICKIT for Circuit Playground Express
https://www.adafruit.com/product/3093
Robot building addon board.

2 x TT Motor
https://www.adafruit.com/product/3777
DC gearbox motor - 200RPM, 3 to 6 vdc

2 x Orange and Clear TT Motor Wheel
https://www.adafruit.com/product/3766
Wheels for the TT motor. This is one of the available options which provides good traction.

Use this if you want to use NiCd batteries.

1 x 4xAA battery holder — https://www.adafruit.com/product/3788

Use this if you want to use NiCd batteries.

---

1 x 3xAA battery holder — https://www.adafruit.com/product/3842

Use this if you want to use Alkaline batteries. This is one 3xAA holder. there are others and depending on which you have you may also need the DC Power adapter. This wasn't available at the time this build was done so a different holder was used.

---

1 x Male DC Power adapter — https://www.adafruit.com/product/369

2.1mm plug to screw terminal block, if your chosen battery pack doesn't have a 2.1mm plug.

---

1 x 5v power supply — https://www.adafruit.com/product/276

Handy power supply to have while working on a CRICKIT project at your desk, or to power a stationary project.

---

1 x Cardboard Pop Rivets — https://www.adafruit.com/product/3822

Small plastic rivets that are great for joining pieces of cardboard ot other thin things. We'll use them to connect body segments.

With bumpers

3 x Micro switch with lever — https://www.adafruit.com/product/818

Basic level micro switch, 2 terminals

---

1 x 20 6" F/M Jumper wires — https://www.adafruit.com/product/1954

Great to have around, and used here to connect the micro switches to the CRICKIT

Adding an alert buzzer

1 x 5v buzzer — https://www.adafruit.com/product/1536

Apply power and it makes a 2KHz tone. The simplest way to make a beep.

---

1 x Stacking header strip — https://www.adafruit.com/product/3366

Female header with extra long pins

A bigger, better power system

If you do this, you'll also need the power adapter and a F/M jumper wire listed above.

1 x 6600mAh 3.7v Lipo Battery — https://www.adafruit.com/product/353
This is a big LiPo battery that works great for power hungry motors.

---

1 x PowerBoost 1A LiPo recharger and boost — https://www.adafruit.com/product/2465
For charging the big LiPo as well as boosting its 3.7v to 5v to power the CircuitPlayground Express and CRICKIT.

---

1 x SPDT slide switch — https://www.adafruit.com/product/805
Simple slide switch with pins on 0.1" spacing for use with the LiPo boost

---

Tools and Supplies

- Corrugated cardboard for chassis, bumpers, body pieces and wheels
- hot glue gun and glue rods
- craft knife for cutting cardboard
- double-sided tape
- 1 large binder clip to serve as a rear slide for the bit
- several binder clips to join body pieces together, twice the number of body segments you want
- thin bamboo skewers
- colored construction paper and a gluestick if you wish to give your snake bot a little more character

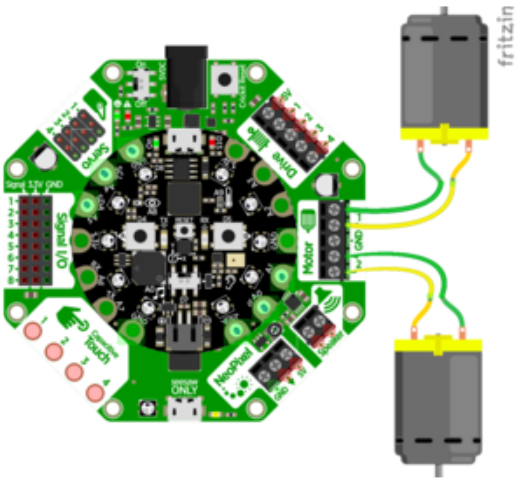
Basic Design

We'll be using CircuitPython for this project. Are you new to using CircuitPython? No worries, there is a full getting started guide here (https://adafru.it/cpy-welcome).

Adafruit suggests using the Mu editor to edit your code and have an interactive REPL in CircuitPython. You can learn about Mu and its installation in this tutorial (https://adafru.it/ANO).

We'll be making use of the CRICKIT robot add on board. New to CRICKIT, we have a guide to get introduce you to its capabilities (https://adafru.it/BD7). There are several other guides that deal with specific aspect of CRICKIT and a growing number of projects. Just search for CRICKIT on learn.adafruit.com (https://adafru.it/BXT).

Start by mounting your Circuit Playground Express on the CRICKIT, and making sure your CRICKIT has up-to-date firmware. See the CRICKIT guide (https://adafru.it/BD7) for details.



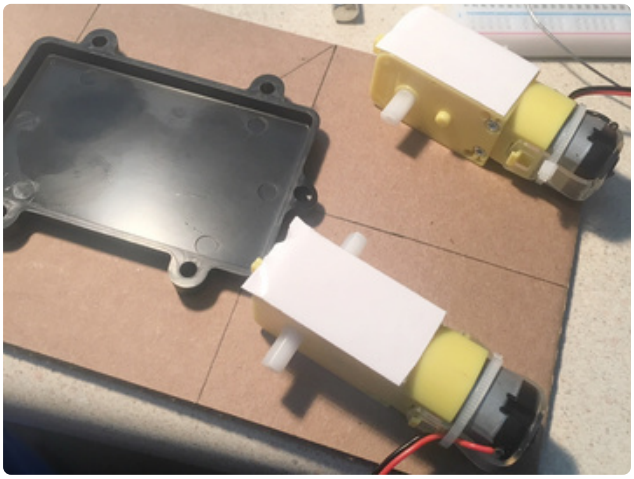
The wiring is simple for this build: connect a couple motors to the CRICKIT. Do notice that they are wired in opposite polarity to each other. That let's us use a positive throttle value for both to get forward motion (and both negative for reverse).

The bot starts with a simple rectangular piece of cardboard for the chassis/base. It has to be wide enough to contain the battery pack and the two TT motors. If you use the yellow TT motors (as opposed to the blue ones), remember that they have a drive shaft on both sides which needs to be considered when deciding how much room is needed. Make it longer than you think you'll need for now; it can always be trimmed as desired. You'll want to leave about 4cm in front on the motors.

I decided to mount everything on the top of the chassis. This lets the chassis ride closer to the ground which makes attaching the tail and bumpers somewhat easier. However, whatever method you use to attach the motors will be holding up the weight of everything else (the batteries being the heaviest item) so use something strong.

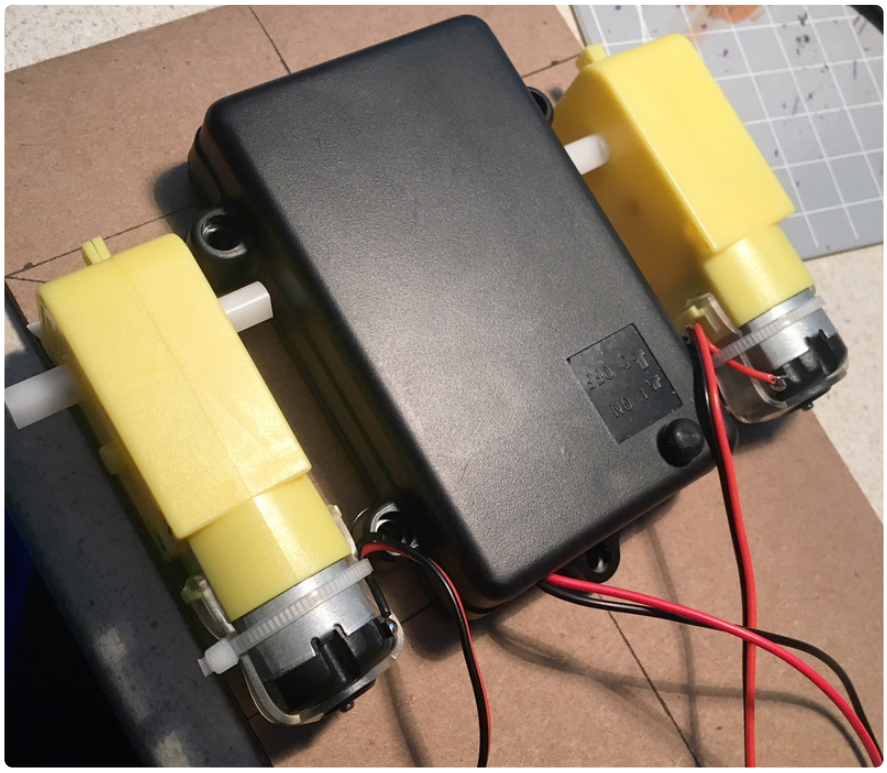
Here is it with the 3xAA weatherproof battery holder.

The battery holder can be secured to the cardboard with some double sided tape, as can the motors. If you use this battery holder, slip the hex nuts in place before taping the holder cover in place. That way you can remove the cover to replace the batteries.

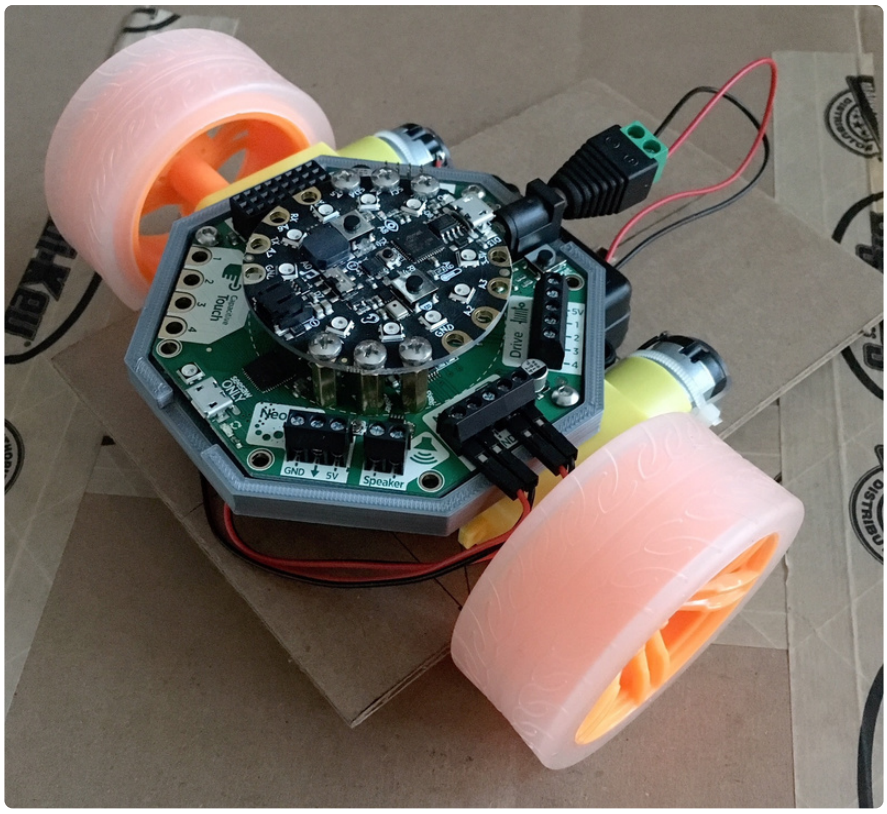


Place tape on the bottom side of the two motors and finalize there positions. You'll probably want to position them with the wires on the inside.

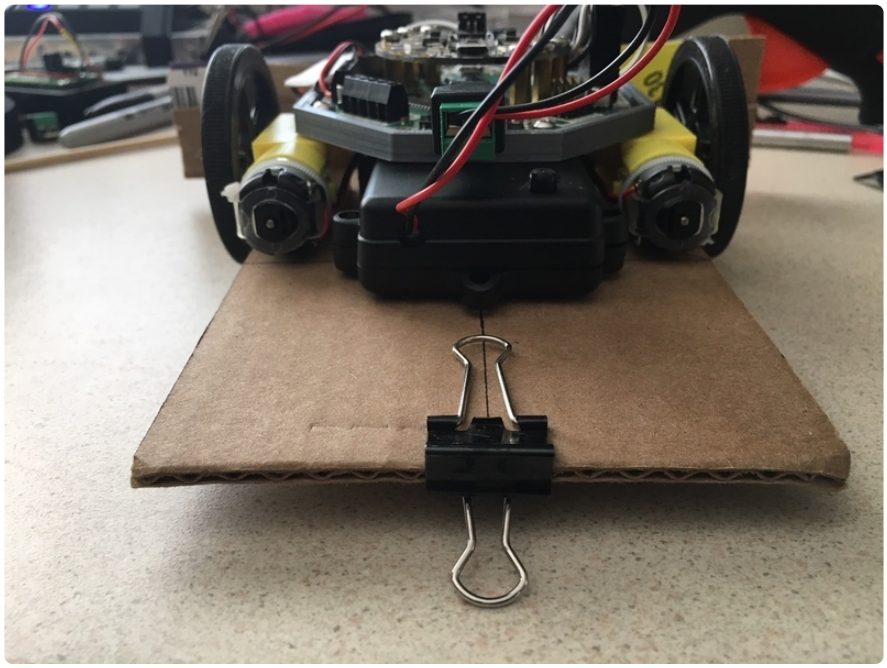
Carefully position and secure the motors. Be sure to get them as straight as possible. You'll want them along the outside edges of the base anyway, so it can be used to align the sides of the motors.



Now use double-sided tape to secure the Circuit Playground Express to the top of the motors. If you have foam double-sided tape you can use it to accommodate the uneven underside of the Circuit Playground Express. To use thin tape, consider using a 3D printed CRICKIT mount (https://adafru.it/BXU). Not only does it provide a smooth bottom surface, but it adds a bit of width that helps secure it to the motors.



To keep the back end from dragging, you can simply add a binder clip to give it something smooth to glide on.



Code

The code for this build is simple. Set up the motors then run one fast and the other slow, repeat, alternating which runs faster.

There's one wrinkle in this. Because motors seldom behave identically, we need to compensate in order to maintain a roughly straight path. Having the bot tack back and forth rather than try to move in a straight line helps disguise the difference but if one motor runs consistently faster than the other the overall course will veer to one side.

One way to deal with this is to monitor the rotation of each motor and adjust the throttle to make their rotations match. We're not doing that in this build, and will use a far simpler, yet less dynamic, solution. That's what the `set_left/right` functions are for: they incorporate a factor into the throttles for each wheel. You can figure out the factors by uncommenting the lines below them to run the bot straight. Adjust the factors until a straight path is achieved. Then remove or comment out those lines.

Here's the code. Save it as **code.py** on your Circuit Playground Express.

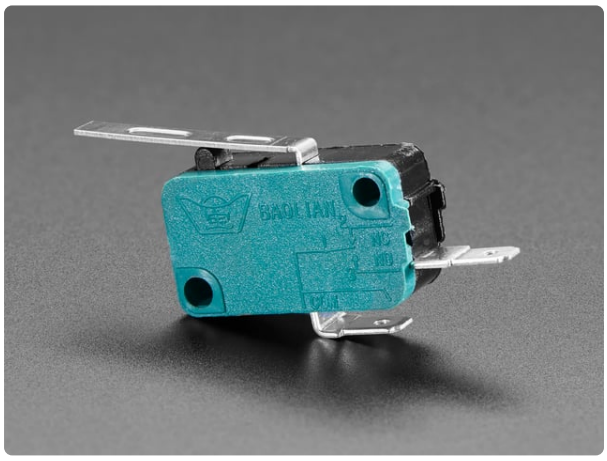
```
# SPDX-FileCopyrightText: 2018 Dave Astels for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import time
from adafruit_crickit import crickit

ss = crickit.seesaw

left_wheel = crickit.dc_motor_1
right_wheel = crickit.dc_motor_2


# These allow easy correction for motor speed variation.
# Factors are determined by observation and fiddling.
# Start with both having a factor of 1.0 (i.e. none) and
# adjust until the bot goes more or less straight
def set_right(speed):
    right_wheel.throttle = speed * 0.9

def set_left(speed):
    left_wheel.throttle = speed


# Uncomment this to find the above factors
# set_right(1.0)
# set_left(1.0)
# while True:
#     pass

while True:
    # tack left
    set_left(0.25)
    set_right(1.0)
    time.sleep(0.75)

    # tack right
    set_left(1.0)
    set_right(0.25)
    time.sleep(0.75)
```

Adding Bumpers

JP's bot (https://adafru.it/BNJ) detects collisions by using the Circuit Playground Express's on board accelerometer. It's a simple, elegant solution (not to mention not requiring any additional parts or construction). However it has some shortcomings: a) collisions have to be at fairly high speed to register, b) it requires a CircuitPlayground Express, so you can't use a Featherwing CRICKIT and a Feather, and c) without more processing you can't determine in what direction the collision occurred.

This project ops for using mechanical switch based bumpers. A drawback is requiring additional parts and construction, but an advantage is that it makes use of more of the CRICKIT's capabilities.

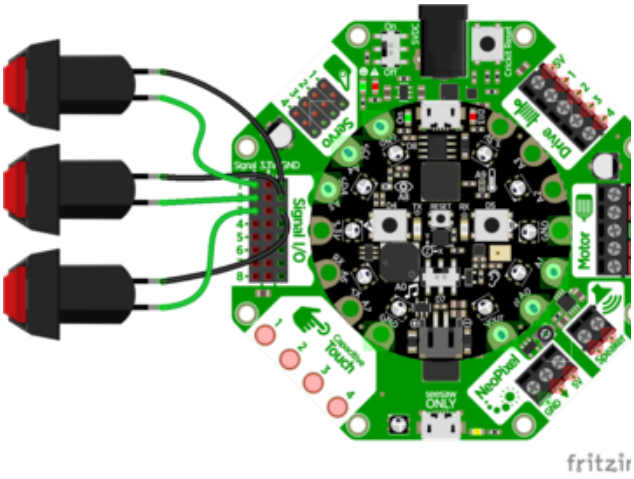
Parts Required



Micro Switch w/Lever - 2 Terminal
Micro-switches are often found in arcade buttons and joysticks but they're also really handy in any kind of mechatronics project or when you need a basic sensor. They are always...
https://www.adafruit.com/product/818

Premium Female/Male 'Extension' Jumper Wires - 20 x 6"
These Female/Male Extension jumper wires are handy for making wire harnesses or jumpering between headers on PCB's. These premium jumper wires are 6" (150mm) long and come in...
https://www.adafruit.com/product/1954

Wiring Additions



The wiring is shown here. Note that push-button switches are shown, but lever switches are used.

The Whiskers

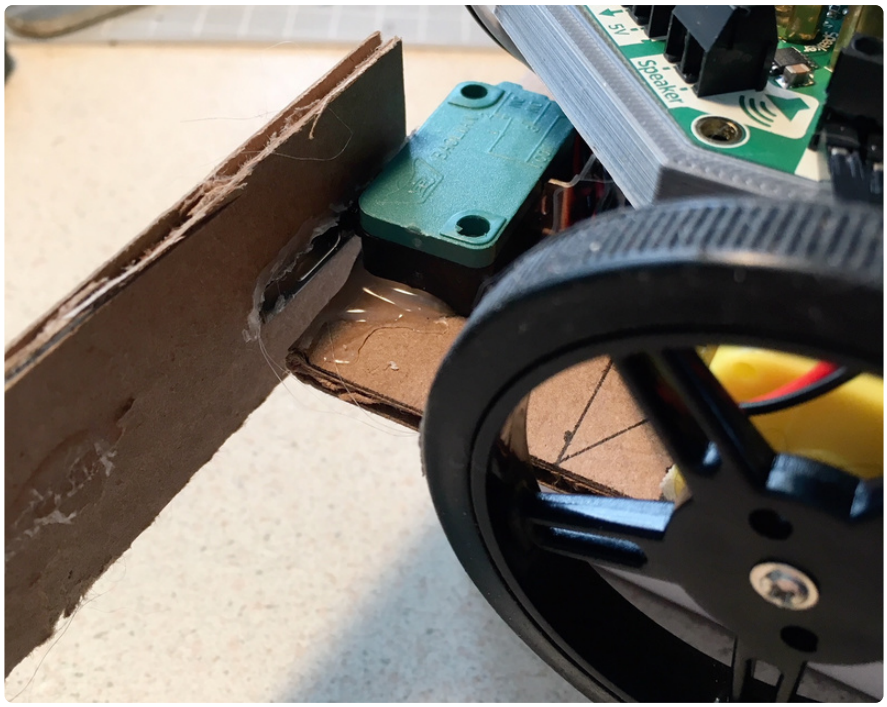
I know... snakes don't have whiskers. All I can say is that this one does.

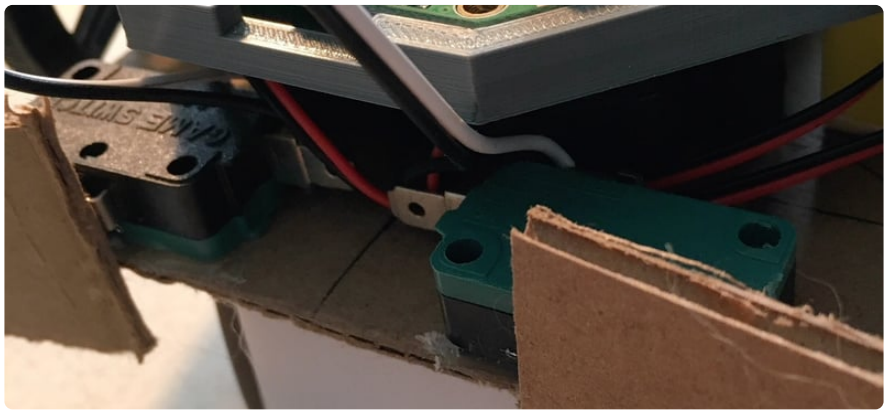
We'll add some whiskers to the bot so that it can know with which side it ran into something. The lever switches are perfect for this.

Start by separating two pairs of jumper wires. Remove the female ends. Strip and tin the fresh cut ends and solder them to the two contacts of the switches, one pair per switch

Then hot glue the switches to the front of the bot so that the levers point to the outside. Their edge should follow the front edge of the bot and they should be fairly close together. You won't be able to get the overly close, though, due to the way the contacts stick out.

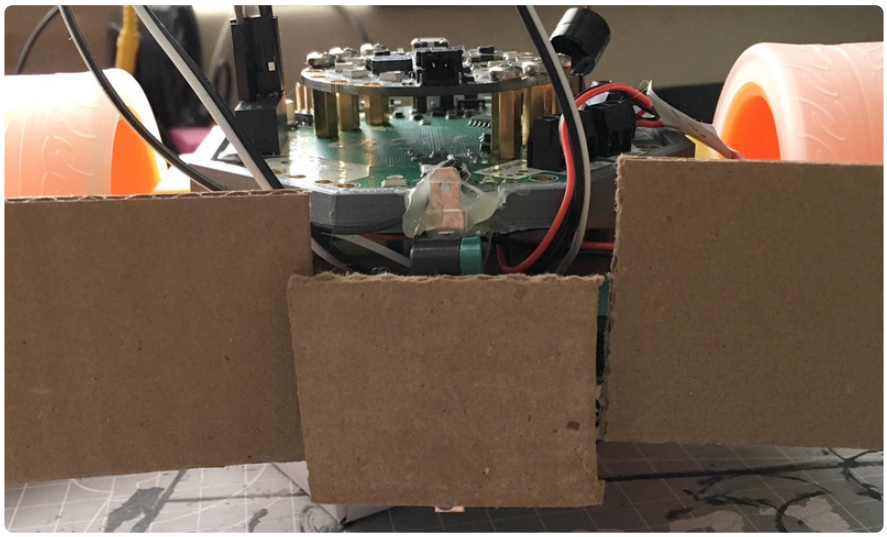
Next hot glue two pieces of cardboard to the levers. They should be around 8cm long by 4cm tall. Position them so that they are approximately parallel to the ground. This will likely result in them not being parallel with the levers. The switch levers usually have some give to them and the weight of the whisker will pull them down somewhat. But don't worry too much about it; as long as they don't drag, you're ok. And you can always trim them if they do.



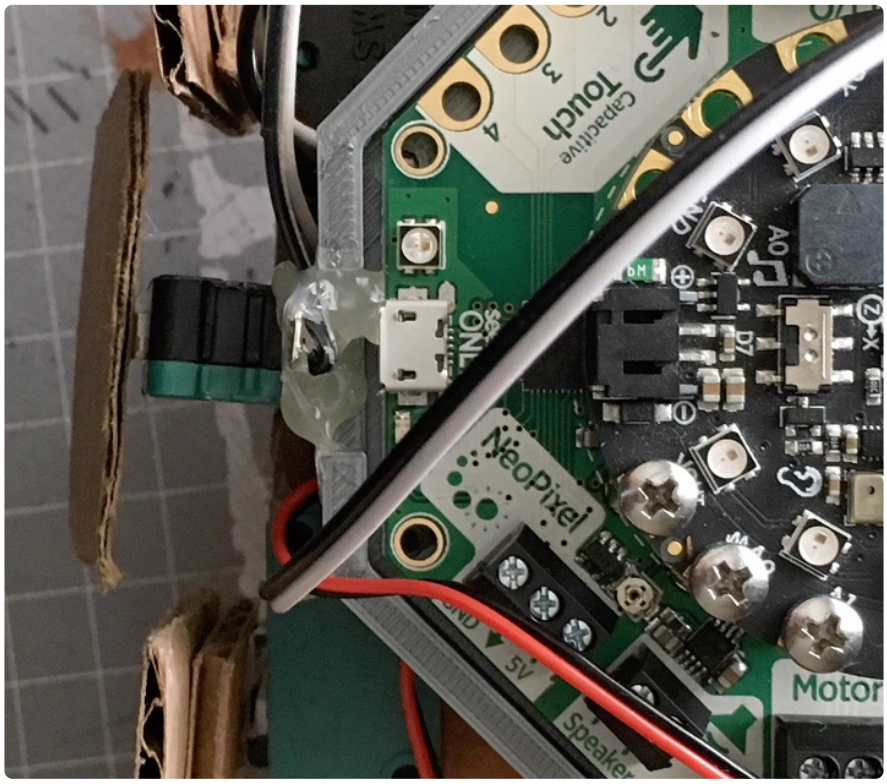


Notice the empty space between the whiskers. Right now it's a blind spot for your bot. It needs a nose to bump into things directly in front of it.

Thankfully, snakes do have noses.



The way to put this in place is to mount a switch vertically in the center of the front of the bot base, and glue a piece of cardboard to the switch lever that is wide enough to fill the gap between the whiskers.



The switch needs to be mounted so that the bottom of the nose sticks out at the bottom. This way it is sure to actuate the lever to press the switch when something is bumped.

Hot gluing the switch to the bot base and the front of the Circuit Playground Express (or it's mount) works well enough, but note that it does need to be angled slightly.

Code

The main addition in this version of the code is the `react_to_bumpers()` function.

```
def react_to_bumpers():
    attempt_count = 0
    # keep trying to back away and turn until we're free
    while True:

        # give up after 3 tries
       if attempt_count == 3:
          return True

       bumped_left = not ss.digital_read(LEFT_BUMPER)
       bumped_right = not ss.digital_read(RIGHT_BUMPER)
       bumped_center = not ss.digital_read(CENTER_BUMPER)

       # Didn't bump into anything, we're done here
       if not bumped_left and not bumped_right and not bumped_center:
           return False

       # If the middle bumper was triggered, randomly pick a way to turn
       if bumped_center:
          bumped_left = random.choice([False, True])
           bumped_right = not bumped_left

       # Back away a bit
       set_left(-0.5)
       set_right(-0.5)
       time.sleep(0.5)
```

```
        # If we bumped on the left, turn to the right
        if bumped_left:
            set_left(1.0)
            set_right(0.0)

        # If we bumped on the right, turn left
        elif bumped_right:
           set_left(0.0)
           set_right(1.0)

        # time to turn for
        time.sleep(random.choice([0.2, 0.3, 0.4]))
        attempt_count += 1
```

It checks the state of the three bumper input signals and tries to deal with the situation. Note that when a switch is closed due to bumping into something, that input signal is False.

In all cases where at least one bumper switch is activated it will back up a bit, then turn away from the bumper that detected a collision. How far does it turn? The code randomly turns for 200, 300, or 400 milliseconds.

There are a few other things going on in this code.

Counting Attempts

It counts how many times the loop executes and eventually gives up, recognizing that it's stuck. In that case the function returns `True`. After it backs away and turns if it finds that none of the bumpers detect a collision, it assumes the way ahead is clear and returns `False`.

Handling 'nose' Bumps

The other thing is the handling of the center "nose" bumper. This code randomly picks right or left and carries on as if that bumper has detected a collision. Another approach to dealing with the nose bumper would be to turn 180 degrees and go back the way it came. To do that we'd need someway to measure how far it's turning. Rotation encoders would let us do that, but we're not using them. Turning 180 degrees would also interact badly if you add the tail.

Tack Movement

Another change is moving tacking to a separate function: `tack(direction, duration)`:

```
LEFT = False
RIGHT = True

def tack(direction, duration):
    target_time = time.monotonic() + duration
    if direction == LEFT:
       set_left(0.25)
       set_right(1.0)
    else:
       set_left(1.0)
       set_right(0.25)
    while time.monotonic() &lt; target_time:
       if not(ss.digital_read(LEFT_BUMPER) and
                ss.digital_read(RIGHT_BUMPER) and
                ss.digital_read(CENTER_BUMPER)):
          return react_to_bumpers()
    return False
```

Tack starts the motors running to cause the bot to drift to the left or right, depending on the `direction` parameter. It then loops for the requested amount of time. Each time through the loop, all three bumpers are checked. If any have collided with something (indicated by at least one input being False), `react_to_bumpers()` is called to get out of trouble. If there are no collisions during the requested time, `False` is returned top indicate that everything is good.

The main loop calls `tack()` for each zig and zag. If it returns `False`, the bot is moving freely and continues on it's way. However, if it returns `True` that means it got stuck. In that case the loop exits and stops the motors.

```
while True:
    if tack(LEFT, 0.75):
        break
    if tack(RIGHT, 0.75):
        break
```

Save the code below as code.py on the Circuit Playground Express.

```
# SPDX-FileCopyrightText: 2018 Dave Astels for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import time
import random
from adafruit_crickit import crickit

LEFT = False
RIGHT = True

random.seed(int(time.monotonic()))
ss = crickit.seesaw

left_wheel = crickit.dc_motor_1
right_wheel = crickit.dc_motor_2

RIGHT_BUMPER = crickit.SIGNAL1
LEFT_BUMPER = crickit.SIGNAL2
CENTER_BUMPER = crickit.SIGNAL3
```

```
ss.pin_mode(RIGHT_BUMPER, ss.INPUT_PULLUP)
ss.pin_mode(LEFT_BUMPER, ss.INPUT_PULLUP)
ss.pin_mode(CENTER_BUMPER, ss.INPUT_PULLUP)

# These allow easy correction for motor speed variation.
# Factors are determined by observation and fiddling.
# Start with both having a factor of 1.0 (i.e. none) and
# adjust until the bot goes more or less straight
def set_right(speed):
    right_wheel.throttle = speed * 0.9

def set_left(speed):
    left_wheel.throttle = speed


# Uncomment this to find the above factors
# set_right(1.0)
# set_left(1.0)
# while True:
#     pass

# Check for bumper activation and move away accordingly
# Returns False if we got clear, True if we gave up
def react_to_bumpers():
    attempt_count = 0
    # keep trying to back away and turn until we're free
    while True:

        # give up after 3 tries
        if attempt_count == 3:
            return True

        bumped_left = not ss.digital_read(LEFT_BUMPER)
        bumped_right = not ss.digital_read(RIGHT_BUMPER)
        bumped_center = not ss.digital_read(CENTER_BUMPER)

        # Didn't bump into anything, we're done here
        if not bumped_left and not bumped_right and not bumped_center:
            return False

        # If the middle bumper was triggered, randomly pick a way to turn
        if bumped_center:
            bumped_left = random.choice([False, True])
            bumped_right = not bumped_left

        # Back away a bit
        set_left(-0.5)
        set_right(-0.5)
        time.sleep(0.5)

        # If we bumped on the left, turn to the right
        if bumped_left:
            set_left(1.0)
            set_right(0.0)

            # If we bumped on the right, turn left
        elif bumped_right:
            set_left(0.0)
            set_right(1.0)

            # time to turn for
        time.sleep(random.choice([0.2, 0.3, 0.4]))
        attempt_count += 1


def tack(direction, duration):
    target_time = time.monotonic() + duration
    if direction == LEFT:
        set_left(0.25)
```

```
        set_right(1.0)
    else:
        set_left(1.0)
        set_right(0.25)
    while time.monotonic() < target_time:
        if not(ss.digital_read(LEFT_BUMPER) and
               ss.digital_read(RIGHT_BUMPER) and
               ss.digital_read(CENTER_BUMPER)):
            return react_to_bumpers()
    return False


while True:
    if tack(LEFT, 0.75):
        break
    if tack(RIGHT, 0.75):
        break

set_left(0)
set_right(0)

while True:
    pass
```

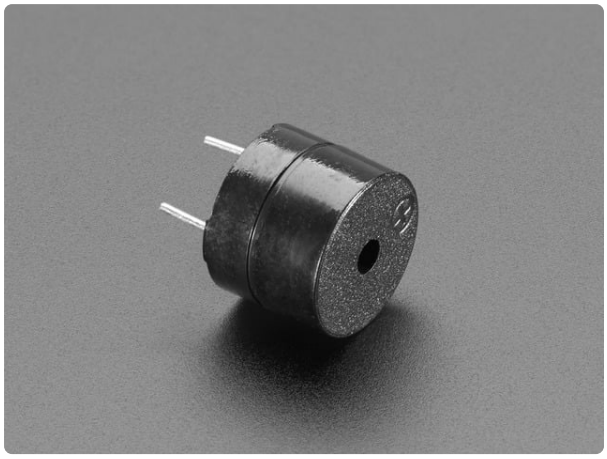
Adding a Warning Buzzer

What happens if your bot wanders away and gets stuck under the couch? Will it sit there patiently waiting for someone to notice and rescue it? Let's give it a voice so it can at least call for help.

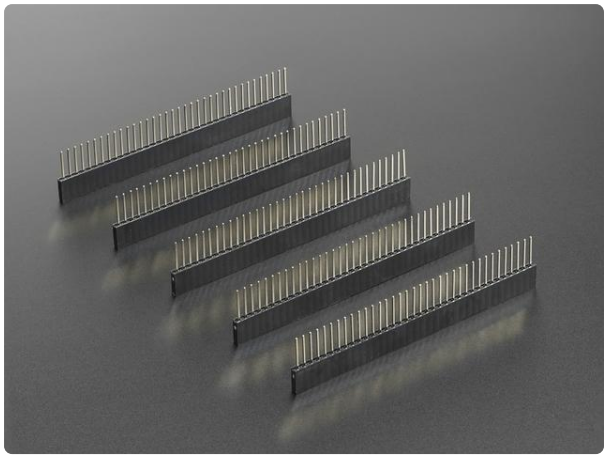
We could use the Circuit Playground Express's on board speaker to play a sound file or a tone. But that takes up a fair bit of memory, and is Circuit Playground Express specific. Part of the goal with this project is to focus on the CRICKIT, regardless of which model it is.

We could use the speaker output, but again that consumes memory for the waveform to play. The simplest solution that will address the problem (giving the bot a voice) is to use a drive output to switch on and off a buzzer. The one listed in the parts list is perfect for this: apply 5v and it outputs a steady, piercing 2kHz tone. The CRICKIT's drive outputs work at 5v.

Parts Required



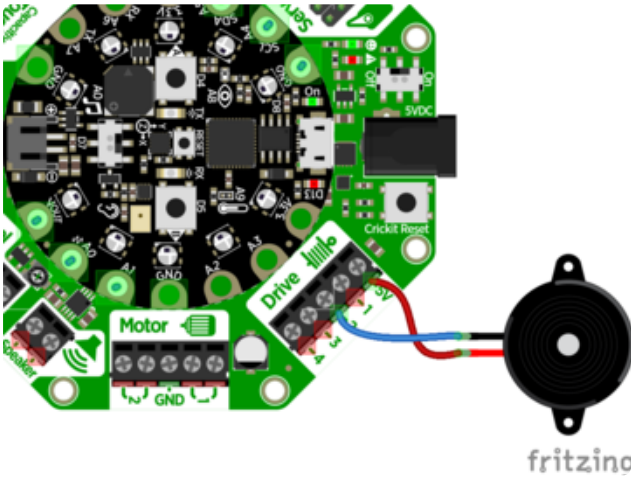
Buzzer 5V - Breadboard friendly
Hey want to hear a really loud noise? Apply 3V to 5V to this buzzer module and you'll be rewarded with a loud 2KHz BEEEEEEEEEEEEEEEEP. Unlike a
https://www.adafruit.com/product/1536



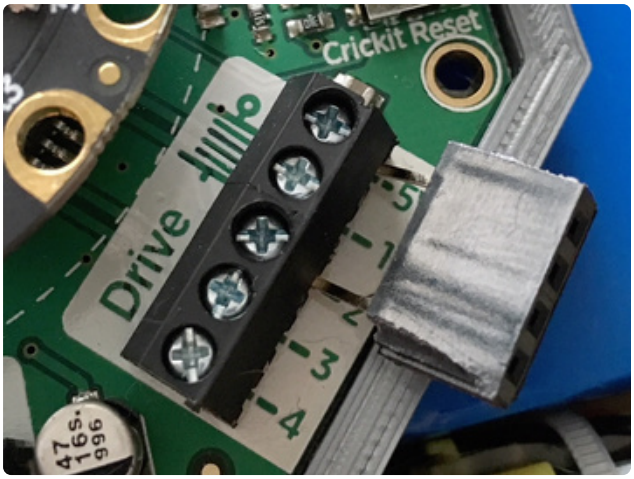
36-pin Stacking header - pack of 5!
Headers are like the duct tape of electronics. They're great for connecting things together, soldering to perf-boards, sockets for wires or break-away header, etc. We go...
https://www.adafruit.com/product/3366

You can also solder some wires to the buzzer to make the pins longer instead of stacking header pins.

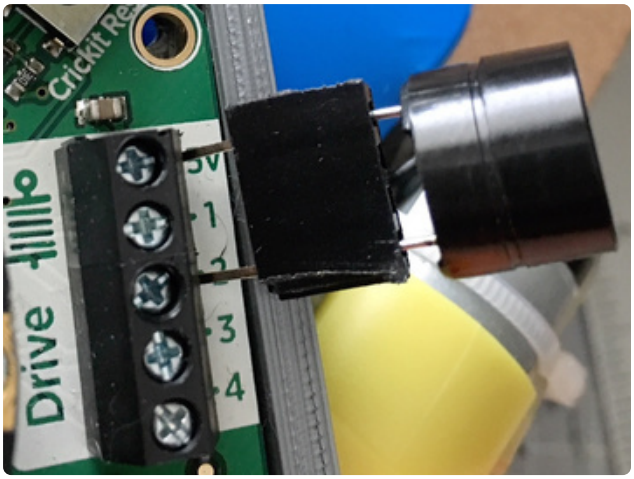
New Wiring Additions



The positive lead of the buzzer connects to the 5v connection in the drive section of the CRICKIT, and the other buzzer lead connects to one of the drive connections.



The leads on the buzzer line up nicely with the 5v and Drive 2 connectors, but are too short to reach. To solve this, you can use a 4-position long piece of stacking header. Pull out the middle 2 pins, leaving the outer two. This is just the right width for the buzzer leads. Bend the header pins as needed and secure them in the drive screw terminals as shown.



Now it's just a matter of inserting the buzzer pins into the header. Note the `+` pin on the buzzer. It has to be at the 5v connection on the CRICKIT.

Code

Adding the buzzer enables the bot to scream for help. The only additional code is at the very end. After exiting the loop and turning off the motors, the buzzer lets the bot call for help.

```
while True:
    for _ in range(3):
        crickit.drive_2.fraction = 1.0
        time.sleep(0.1)
        crickit.drive_2.fraction = 0.0
        time.sleep(.2)
    time.sleep(10.0)
```

Setting drive 2's fraction property to 1.0 turns the buzzer on, and setting it to 0.0 turns it off. This code sounds three short beeps every ten seconds. Very simple, yet very effective.

Save the code below as code.py on the Circuit Playground Express.

```
# SPDX-FileCopyrightText: 2018 Dave Astels for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import time
import random
from adafruit_crickit import crickit

LEFT = False
RIGHT = True

random.seed(int(time.monotonic()))
ss = crickit.seesaw

left_wheel = crickit.dc_motor_1
right_wheel = crickit.dc_motor_2

RIGHT_BUMPER = crickit.SIGNAL1
LEFT_BUMPER = crickit.SIGNAL2
CENTER_BUMPER = crickit.SIGNAL3

ss.pin_mode(RIGHT_BUMPER, ss.INPUT_PULLUP)
ss.pin_mode(LEFT_BUMPER, ss.INPUT_PULLUP)
ss.pin_mode(CENTER_BUMPER, ss.INPUT_PULLUP)

# These allow easy correction for motor speed variation.
# Factors are determined by observation and fiddling.
# Start with both having a factor of 1.0 (i.e. none) and
# adjust until the bot goes more or less straight
def set_right(speed):
    right_wheel.throttle = speed * 0.9

def set_left(speed):
    left_wheel.throttle = speed


# Uncomment this to find the above factors
# set_right(1.0)
# set_left(1.0)
# while True:
#     pass

# Check for bumper activation and move away accordingly
# Returns False if we got clear, True if we gave up
def react_to_bumpers():
    attempt_count = 0
    # keep trying to back away and turn until we're free
    while True:

        # give up after 3 tries
        if attempt_count == 3:
            return True

        bumped_left = not ss.digital_read(LEFT_BUMPER)
        bumped_right = not ss.digital_read(RIGHT_BUMPER)
        bumped_center = not ss.digital_read(CENTER_BUMPER)

        # Didn't bump into anything, we're done here
        if not bumped_left and not bumped_right and not bumped_center:
            return False

        # If the middle bumper was triggered, randomly pick a way to turn
        if bumped_center:
            bumped_left |= random.randrange(10) < 5
            bumped_right = not bumped_left

        # Back away a bit
        set_left(-0.5)
```

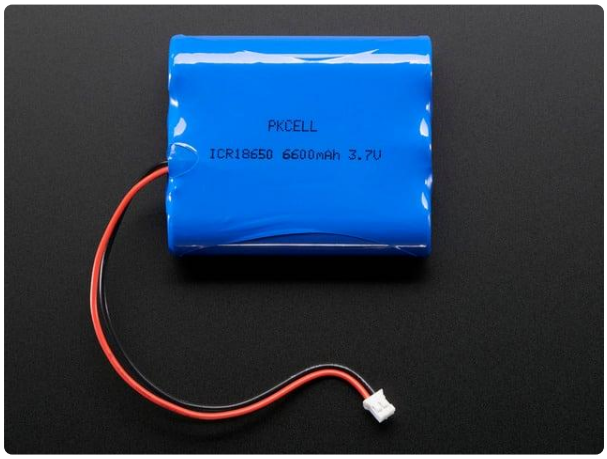
```
        set_right(-0.5)
        time.sleep(0.5)

        # If we bumped on the left, turn to the right
        if bumped_left:
            set_left(1.0)
            set_right(0.0)

            # If we bumped on the right, turn left
        elif bumped_right:
            set_left(0.0)
            set_right(1.0)

            # time to turn for
        time.sleep(random.choice([0.2, 0.3, 0.4]))
        attempt_count += 1


def tack(direction, duration):
    target_time = time.monotonic() + duration
    if direction == LEFT:
        set_left(0.25)
        set_right(1.0)
    else:
        set_left(1.0)
        set_right(0.25)
    while time.monotonic() < target_time:
        if not(ss.digital_read(LEFT_BUMPER) and
               ss.digital_read(RIGHT_BUMPER) and
               ss.digital_read(CENTER_BUMPER)):
            return react_to_bumpers()
    return False


while True:
    if tack(LEFT, 0.75):
        break
    if tack(RIGHT, 0.75):
        break


set_left(0)
set_right(0)

while True:
    for _ in range(3):
        crickit.drive_2.fraction = 1.0
        time.sleep(0.1)
        crickit.drive_2.fraction = 0.0
        time.sleep(.2)
    time.sleep(10.0)
```

A Bigger, Better Power Supply

Motors are fairly power hungry. Especially if they can't spin when power is applied. This can cause at best noise, and at worst a voltage drop (like a brownout). Since the same supply is used for the CRICKIT and the connected motors as well as for the Circuit Playground Express (or other board, depending on your CRICKIT) this can cause some problems. MCU boards really like smooth steady power. Alkaline or NiMH AA batteries don't fair well in these situations. There are a couple options if this is a problem: you can use separate supplies for the motors and the controller board. This

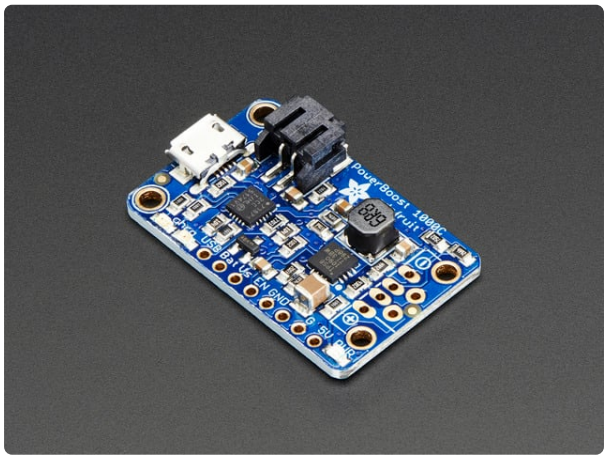
means more parts, more bulk, and the fact that the CRICKIT isn't designed for separate supplies. The other option is to use a beefier supply that can deal with the motor loads.

Beefing up the batteries is fairly simple. Adafruit stocks a massive 6600 mAh LiPo battery pack that provides far more power than Alkalines (up to 3000 mAh) or Ni-MH (about 2500 mAh). The problem is that LiPos are 3.7v and CRCKIT requires 5v.



Lithium Ion Battery Pack - 3.7V 6600mAh
Need a massive battery for your project? This lithium-ion pack is made of 3 balanced 2200mAh cells for a total of 6600mA capacity! The cells are connected in parallel and spot-welded...
https://www.adafruit.com/product/353

Fortunately, there is another product that can deal with that as well as provide several additional benefits: the PowerBoost 1000C (http://adafru.it/2465).



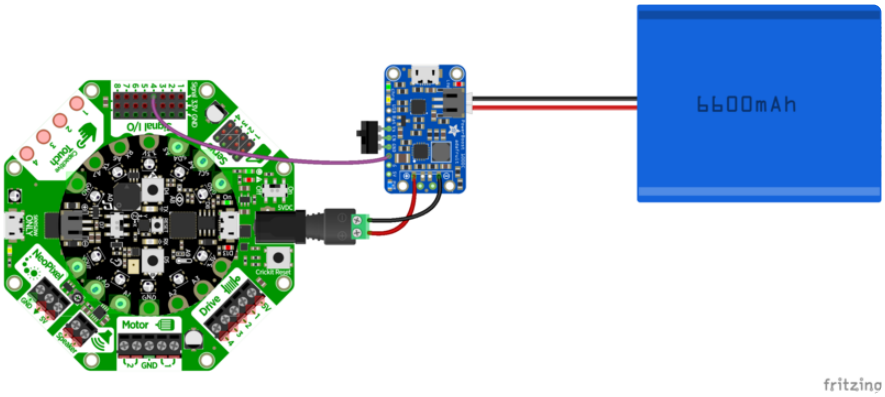
PowerBoost 1000 Charger - Rechargeable 5V Lipo USB Boost @ 1A
PowerBoost 1000C is the perfect power supply for your portable project! With a built-in load-sharing battery charger circuit, you'll be able to keep your power-hungry...
https://www.adafruit.com/product/2465

This board's core function is to convert (aka boost) the LiPo's 3.7v to 5v. It also provides a charger using micro USB as a power source, and can supply 5v while the LiPo is being charged. There are indicator LEDs to indicate charging/full, 5v power being supplied, and low battery output. The latter is also available as a high/low signal that can be read by the controller board.

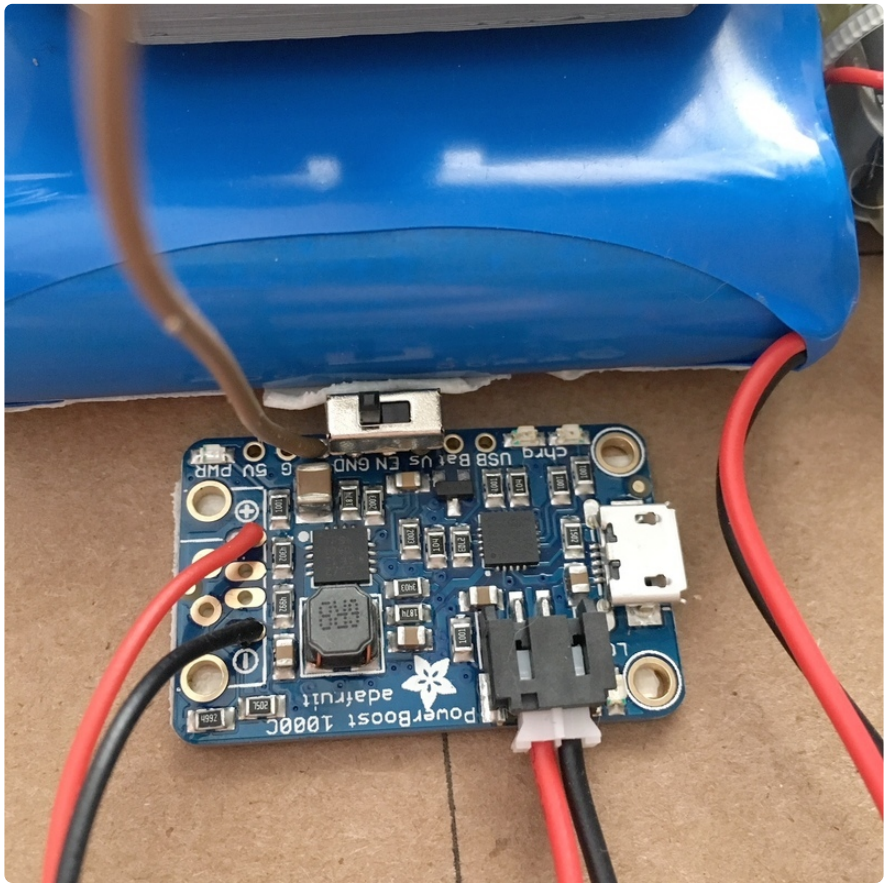
You can get power out of this board by soldering directly onto the + and - pads, soldering on the supplied USB-A socket (included), or soldering on a 2-pin terminal block (not included).

Finally, there is an enable input on that row of pins that will shut down the power booster (leaving the charger active). A truly nice piece of the design of this board is that the enable connection is flanked by ground and Vs. This lets you directly solder on a breadboard friendly slide switch.

The wiring is shown below. Note the switch that turns the power supply on/off as well as the LBO signal connected to Signal 4 of the CRICKIT.



Either double-sided tape or hot glue can be used to mount the board. Double-sided tape was used to secure the battery pack in place.



Code

This is even less of a change to the code than adding the buzzer was. In fact you don't really need any changes to the code if you don't want to monitor the Low Battery Level signal from the PowerBoost1000 board. Assuming you do, it's just the matter of setting up the input pin:

```
LBO = crickit.SIGNAL4
ss.pin_mode(LBO, ss.INPUT_PULLUP)
```

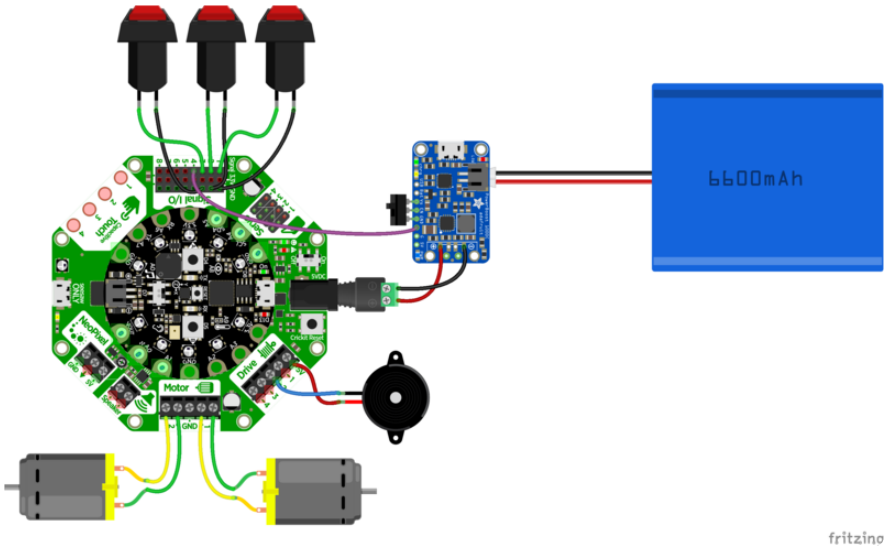
and checking it at the top of the main loop:

```
if not ss.digital_read(LBO):
    break
```

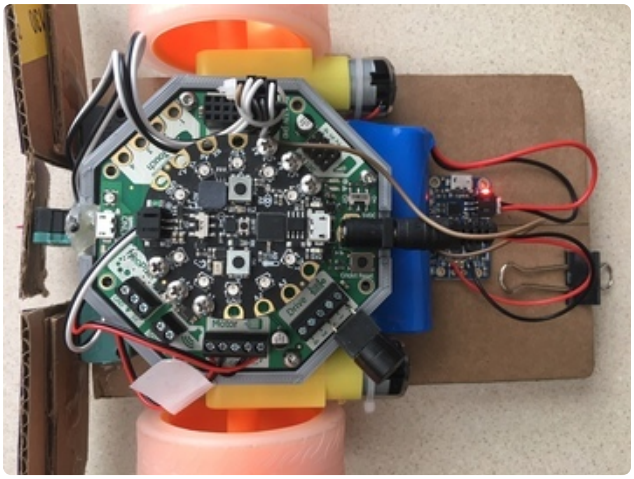
Like the case of getting stuck, this will turn off the motors and start calling for help (assuming you installed the buzzer).

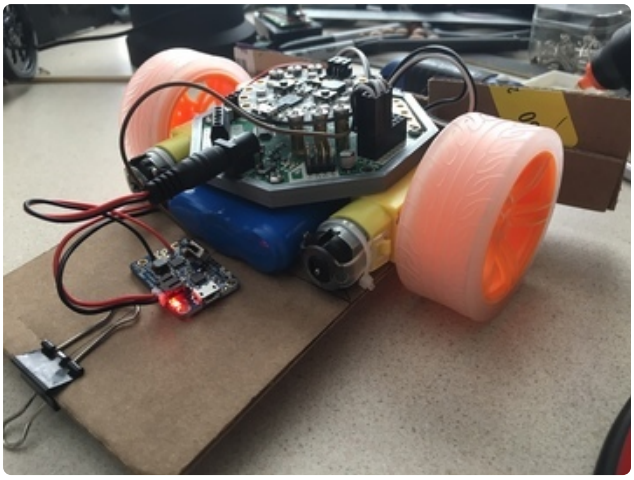
Putting it All Together

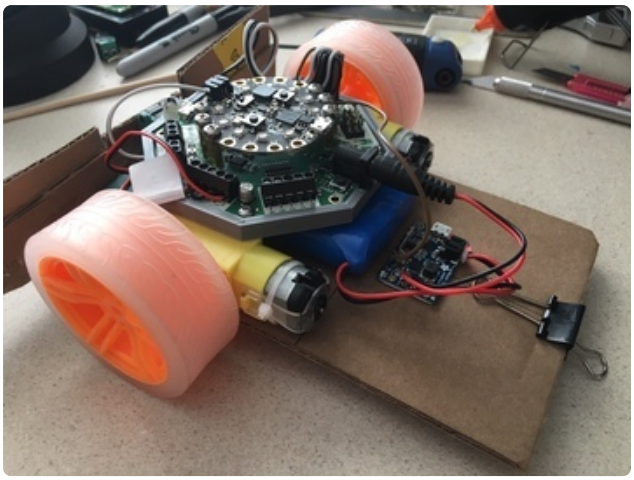
With all the hardware in place: motors, bumpers, buzzer, and enhanced power system, the wiring looks like this:



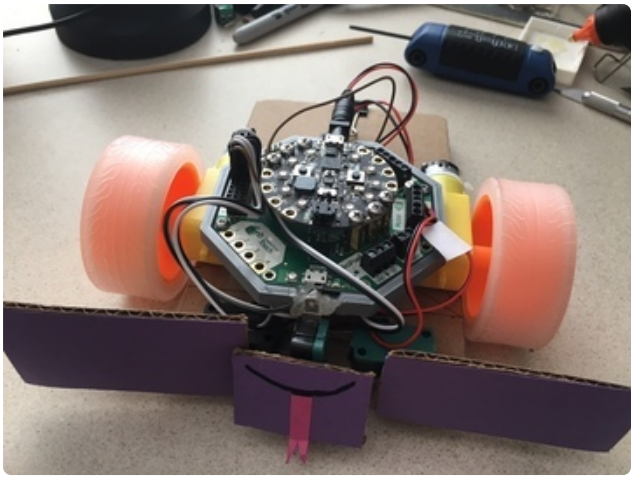
And are some shots of the final build.

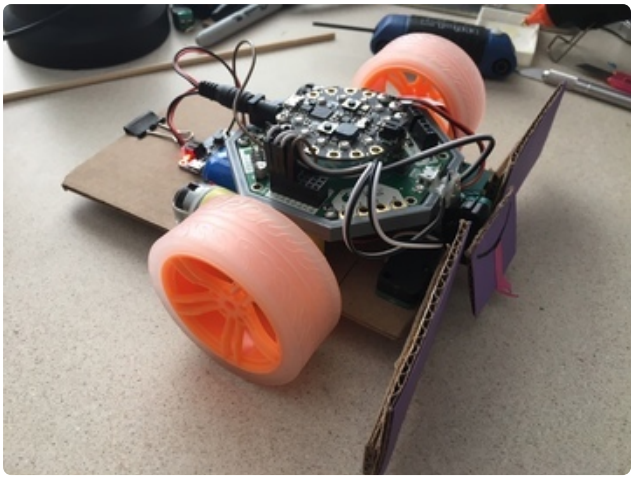


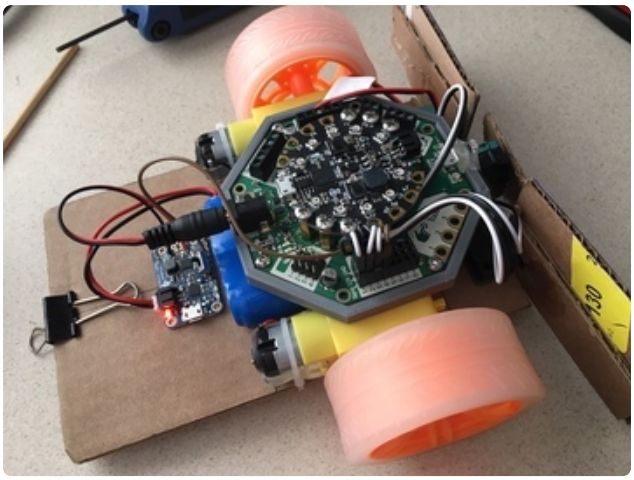


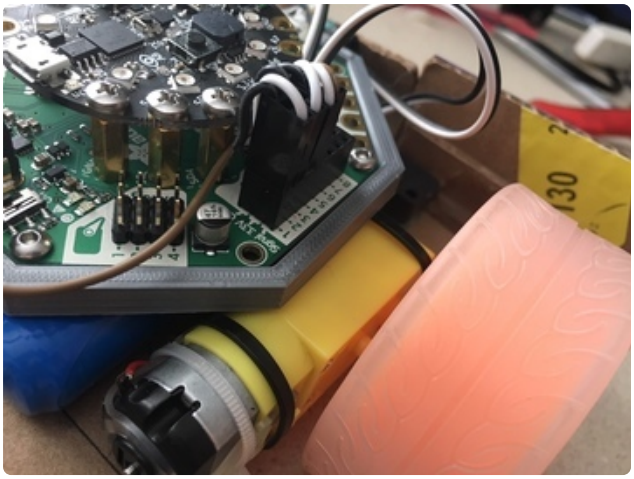


Here are some shots of the final build.









While mounting the motors on the top of the chassis makes it easier to attach the bumpers and tail, it does mean that whatever you attached them with has to bear the full weight of the bot.

If you have trouble with them coming unstuck you can add a couple zip ties on each motor to take that stress off of the adhesive. You still want to stick them in place so they don't wiggle around.

The code in it's entirety is below, copy it to you Circuit Playground Express and off you go.

```
import time
import random
from adafruit_crickit import crickit

LEFT = False
RIGHT = True

random.seed(int(time.monotonic()))
ss = crickit.seesaw

left_wheel = crickit.dc_motor_1
right_wheel = crickit.dc_motor_2

RIGHT_BUMPER = crickit.SIGNAL1
LEFT_BUMPER = crickit.SIGNAL2
CENTER_BUMPER = crickit.SIGNAL3
LBO = crickit.SIGNAL4

ss.pin_mode(RIGHT_BUMPER, ss.INPUT_PULLUP)
ss.pin_mode(LEFT_BUMPER, ss.INPUT_PULLUP)
ss.pin_mode(CENTER_BUMPER, ss.INPUT_PULLUP)
ss.pin_mode(LBO, ss.INPUT_PULLUP)

# These allow easy correction for motor speed variation.
# Factors are determined by observation and fiddling.
# Start with both having a factor of 1.0 (i.e. none) and
# adjust until the bot goes more or less straight
def set_right(speed):
```

```
    right_wheel.throttle = speed * 0.9

def set_left(speed):
    left_wheel.throttle = speed


# Uncomment this to find the above factors
# set_right(1.0)
# set_left(1.0)
# while True:
#     pass


# Check for bumper activation and move away accordingly
# Returns False if we got clear, True if we gave up
def react_to_bumpers():
    set_left(0.0)
    set_right(0.0)
    attempt_count = 0

    # keep trying to back away and turn until we're free
    while True:

        # give up after 3 tries
        if attempt_count == 3:
            return True

        bumped_left = not ss.digital_read(LEFT_BUMPER)
        bumped_right = not ss.digital_read(RIGHT_BUMPER)
        bumped_center = not ss.digital_read(CENTER_BUMPER)

        # Didn't bump into anything, we're done here
        if not bumped_left and not bumped_right and not bumped_center:
            return False

        # If the middle bumper was triggered, randomly pick a way to turn
        if bumped_center:
            bumped_left |= random.randrange(10) < 5
            bumped_right = not bumped_left

        # Back away a bit
        set_left(-0.5)
        set_right(-0.5)
        time.sleep(0.5)

        # If we bumped on the left, turn to the right
        if bumped_left:
            set_left(1.0)
            set_right(0.0)

            # If we bumped on the right, turn left
        elif bumped_right:
            set_left(0.0)
            set_right(1.0)

            # time to turn for
        time.sleep(random.choice([0.2, 0.3, 0.4]))
        attempt_count += 1


def tack(direction, duration):
    target_time = time.monotonic() + duration
    if direction == LEFT:
        set_left(0.25)
        set_right(1.0)
    else:
        set_left(1.0)
        set_right(0.25)
    while time.monotonic() < target_time:
```

```
        if not(ss.digital_read(LEFT_BUMPER) and
               ss.digital_read(RIGHT_BUMPER) and
               ss.digital_read(CENTER_BUMPER)):
            return react_to_bumpers()
    return False


while True:
    # check for low voltage/stall
    if not ss.digital_read(LBO):
        break
    if tack(LEFT, 0.75):
        break
    if tack(RIGHT, 0.75):
        break

set_left(0)
set_right(0)

while True:
    for _ in range(3):
        crickit.drive_2.fraction = 1.0
        time.sleep(0.1)
        crickit.drive_2.fraction = 0.0
        time.sleep(.2)
    time.sleep(10.0)
```

Snake? What Snake?

Now we have a bot that can wander about, bouncing off walls.

That's the head of the snake. Adding a body is a nice craft project. No additional electronics (although I can imagine adding some NeoPixels (https://adafru.it/BZ0) along the spine) and no additional code (again, unless you add something more to it).

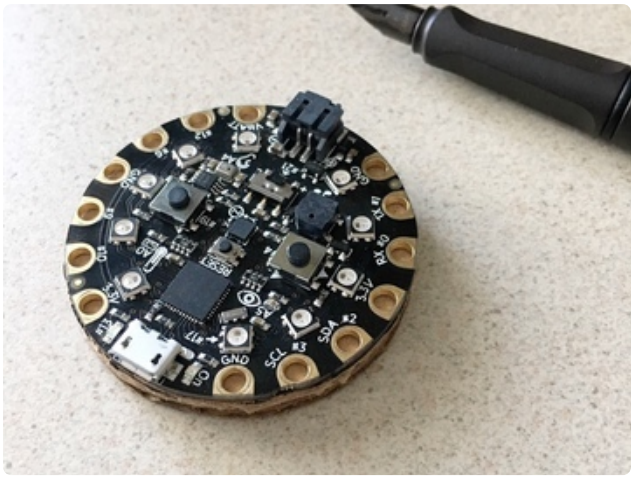
Cutting the pieces

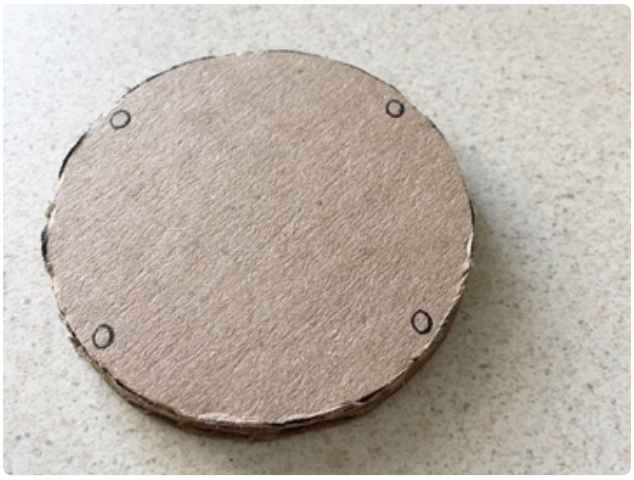
Start by cutting several rectangular pieces of cardboard 10cm x 16cm. Box flaps work well for this. Cut one for each section of body you want.

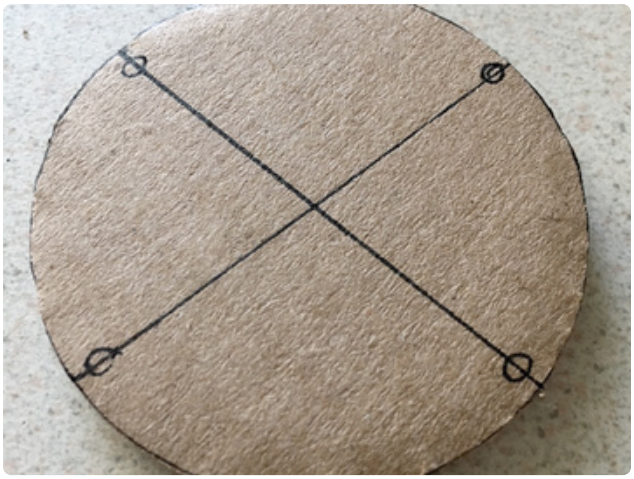
Next take more pieces of cardboard and cut disks out of it for wheels. Tracing a Circuit Playground Express works well and is a good size. Circles can be tricky to cut out well with a craft knife, but scissors work nicely. Cut 4 disks for each body piece.

Glue pairs of disks together with the corrugations at 90 degrees to each other. This strengthens them as well as alleviating problems from individual disks not being perfectly round.

Marking the Pieces

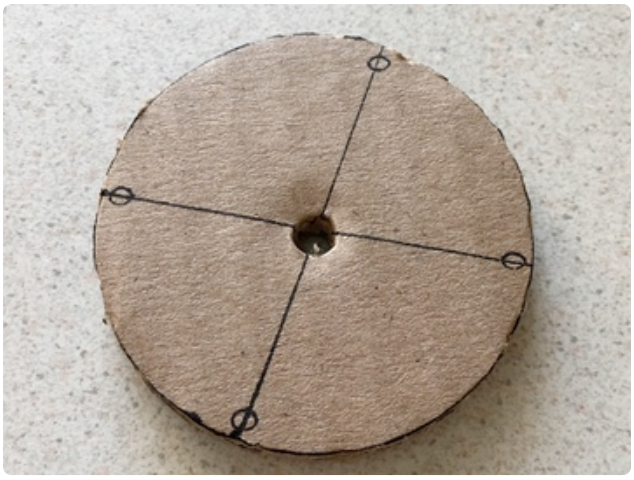




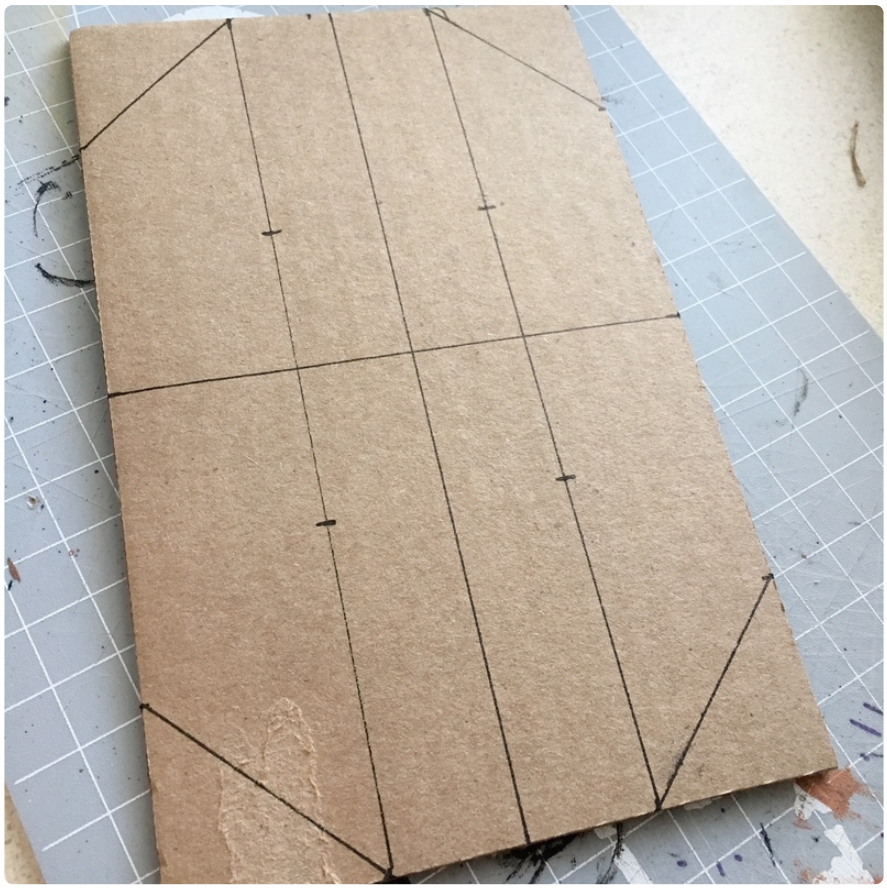


Take the Circuit Playground Express again and mark 2 pairs of opposite holes. Draw a line connecting opposite pairs to get the center point of the disks. At the center point, put a hole. It should be slightly bigger than the round skewers you are using for axles.

Poke a starting hole with the tip of a craft knife, then enlarge and round it with a knitting needle, Scru tool, or even a golf tee. When poking through cardboard, be extremely careful of your fingers. Even something dull like the Scru tool can hurt if pushed with enough force.

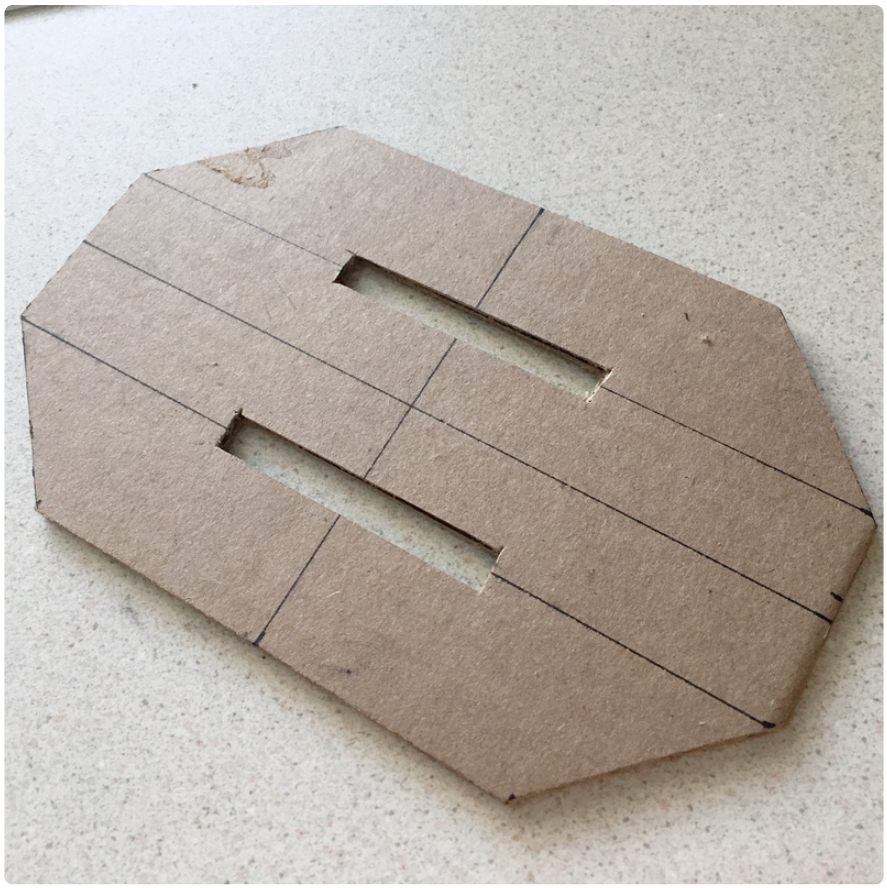


With each rectangular body piece, mark center lines in each dimension, as well as a line along the length of each piece 2cm on each side of the center line. Those last two lines should be 3cm from the long edges. Mark 3cm along the long edge from each corner and draw a line between each pair of corner marks. This will give you a line to cut along to remove the corners. Finally, mark 2.75cm from the center mark on each of the lines 2cm from the long center line. You should end up with something that looks like:

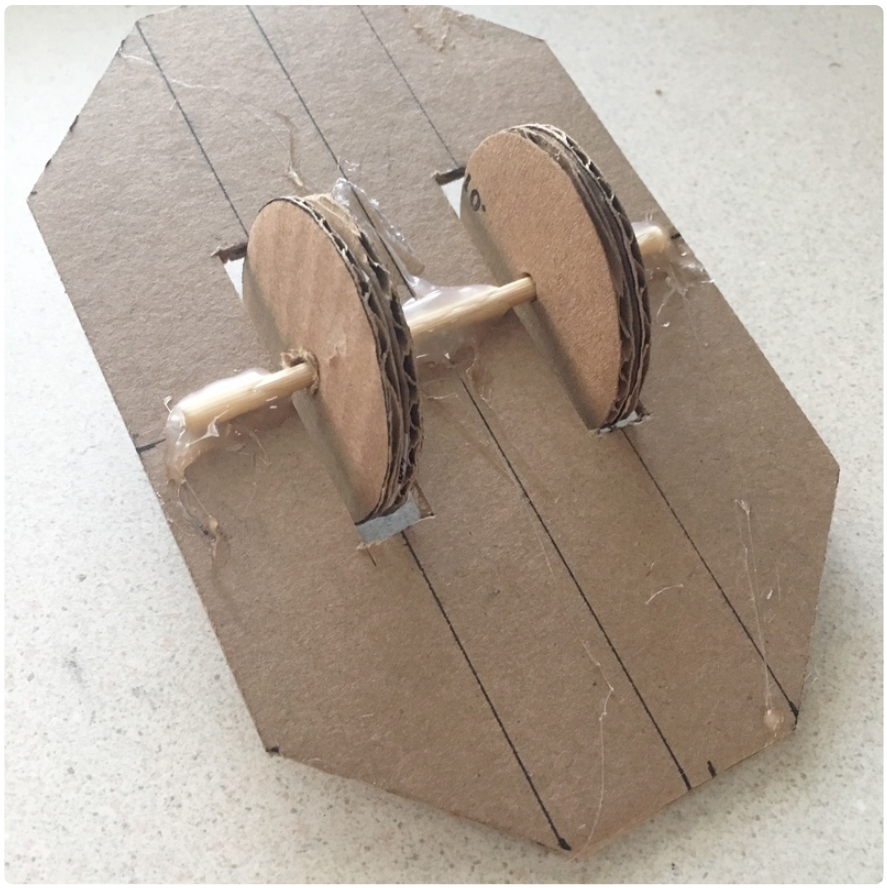


Assembling the Body Sections

Now we need to trim off the corners and cut slots for the wheels. Make the slots wide enough that your wheel disks have enough room to spin freely.



Cut pieces of bamboo skewers a bit sorter than the width of the body pieces, one per body section. These will be the axles. Slide each one through two wheel disks. Position the wheels in their slots, and hot glue the axles to the body pieces, along the the half-way line. Just be careful not to get any glue where the wheels meet the axle. This will be the underside of the body piece, partly because of how messy hot glue can be, and partly to give a smooth top surface for later decorating.



There are any number of ways to connect the body pieces together. A quick, cheap, and easy way is to use small binder clips and cardboard rivets.

Place a binder clip on the ends of each body piece except the final end. Place one in the middle of the head piece, sliding the existing one to the side. You'll still need that original clip for the back of the bot to slide on.

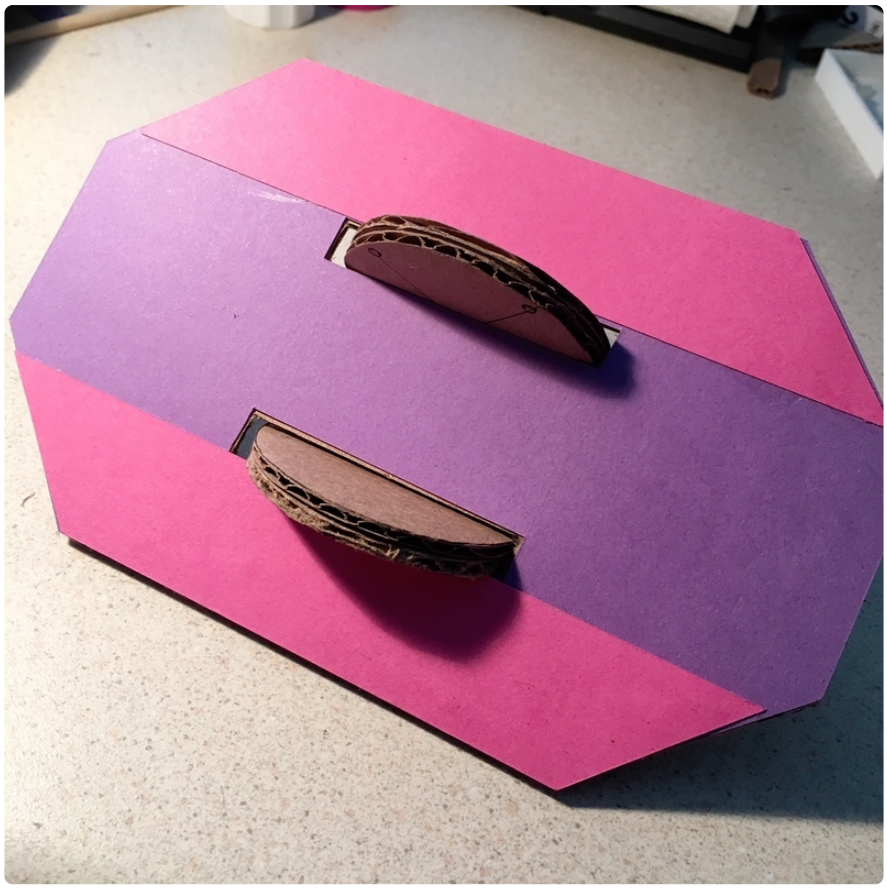
Flip the handle parts on the underside of the clips so they lie against the body pieces. place the extending clip handles on two body pieces together so that they overlap and slip a rivet through them, securing it with the flat part of the rivet. Do this with front and rear clips of subsequent body pieces to assemble a chain that will be the body of the snake.

Blinka-fying

What's a CircuitPython controlled snake bot with some Blinka trappings?

Now's where those scissors and gluestick skills pay off. Get some construction paper in appropriate colors, a couple googly eyes, and a glue stick.

Start by cutting pieces of purple construction paper to match the shape of the body pieces, complete with wheel cutouts. Glue those onto the top of the body pieces. Next cut pieces of pink to fit on the edges. Two are needed per body section. They'll be almost 16cm long, a bit less than 3cm wide (just wide enough to reach the wheel cut-out), with a corner cut off each end to match the body piece. Glue one of those on each side of the body pieces. For both of these shapes, making a cardboard template is useful.



Next cut "lollipops" out of some light blue paper, about 3cm long. The round heads will be halfway over the purple/blue line. You'll have to trim the tops off the lollipops where they are against the wheels.

The head can be simpler, a purple cover will do.

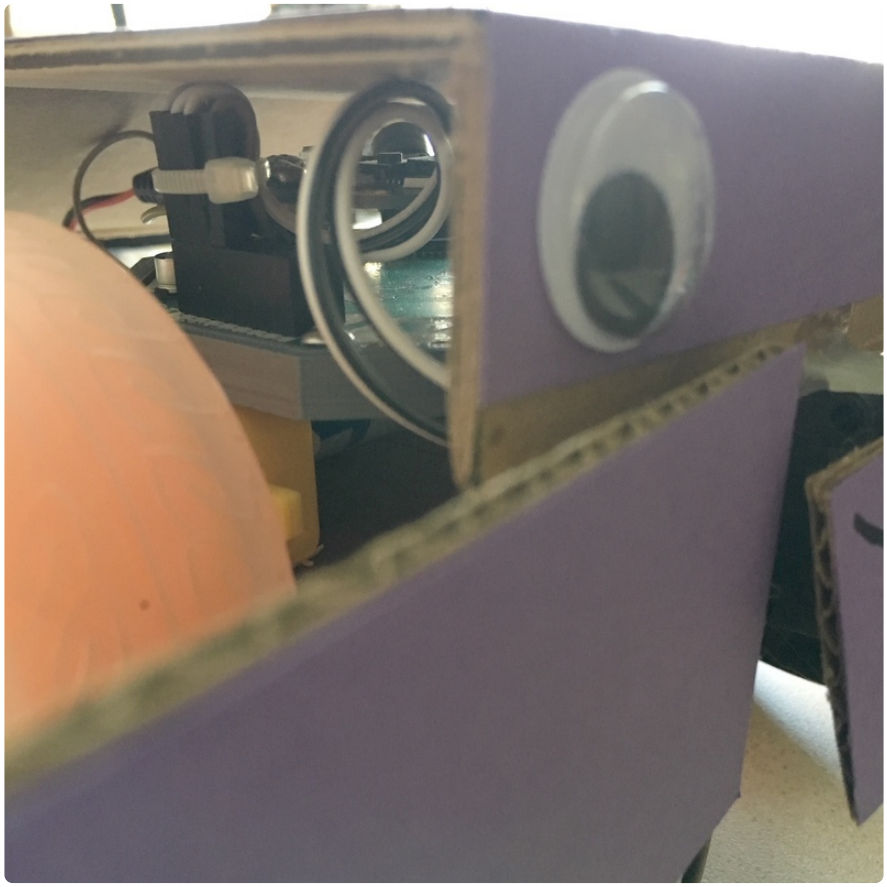
Start by cutting a piece of corrugated cardboard the same width as the base piece of the bot head. Start with it quite a bit longer than the head. Measure how high it has to be above the base piece and mark across the cover piece where it needs to be bent to make a front flap. Depending on how you built the bumpers, this will be long enough of a flap to reach something to glue to. If you added the nose/center bumper, the top of that switch ia a good choice. Now you will need to cut partially along that line, only through the top layer and into the corrugations. Cut a couple millimeters to either side of that line as well. The goal is to remove one outside layer and the corrugation layer, while leaving the other outside layer untouched.

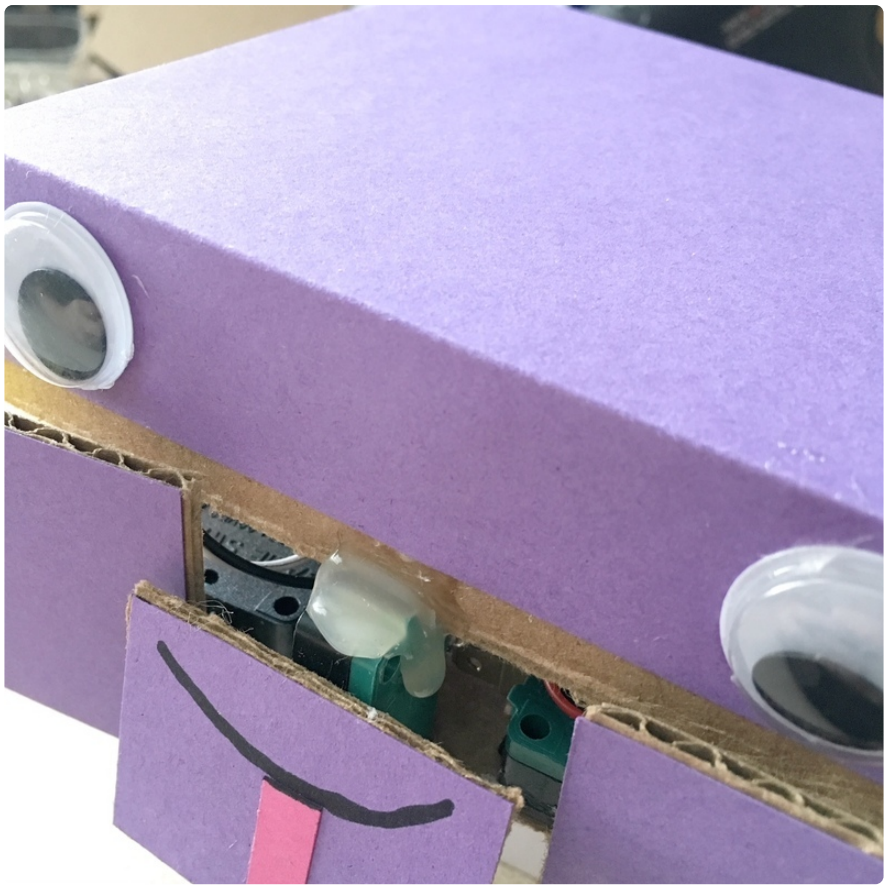
This channel will allow you to make a clean 90 degree bend. Fill the channel with hot glue and fold it to 90 degrees. Hold it there while the glue cools. To reinforce it, you can run a heavy bead of hot glue along the inside of the corner.

A gentle bend to have the back of the cover slope down toward the back end of the base piece completes the head cover. [I was fortunate enough to find a piece of Digikey box that was prescored for some reason in exactly the right place.]

Now it simply has to be mounted on the head. Use some zip ties to get wires out of the way as required and glue the cover in place, holding it until the glue cools.

Finish the effect by gluestick-ing an appropriately cut piece of purple paper over the cover and decorate as desired.





It's interesting how fairly complex looking behavior can emerge unplanned from very simple rules. Westworld is probably not far off when they say that human behavior can be codified in ~10,000 lines of code.

Here's our snake bot spontaneously demonstrating that most dog-like of behaviors.