Really Simple Animatronic Tail

Created by Phillip Burgess

https://learn.adafruit.com/really-simple-animatronic-tail

Last updated on 2023-08-29 02:49:08 PM EDT

Table of Contents

Introduction

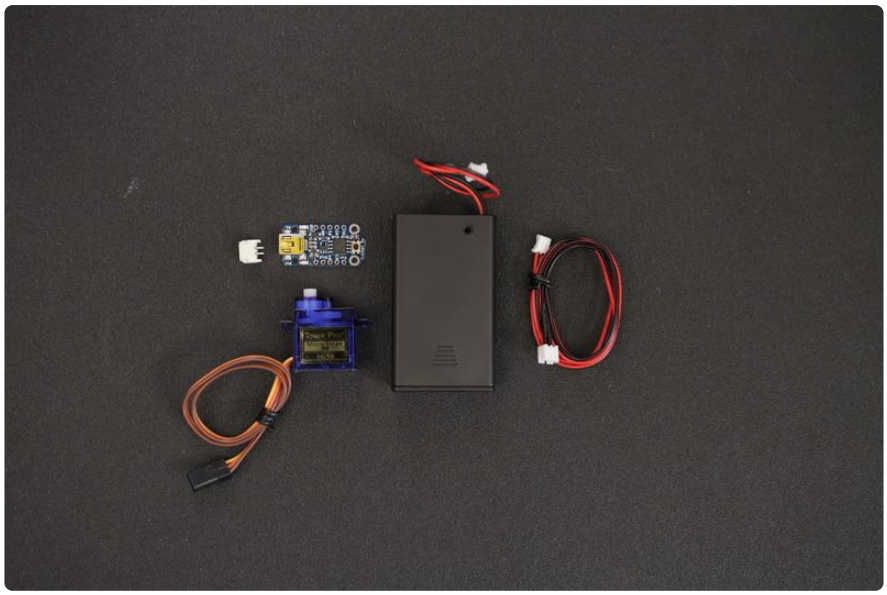
What kid hasn't imagined what it's like having a tail? Let's make it real!

Animatronics is tricky. In addition to the software and electrical engineering of most of our projects, it requires mechanical engineering. Normally when I talk with cosplayers just getting started with electronics, I point them toward a light () or sound () project, to limit the number of variables.

This project reduces animatronics to its simplest case. Mechanical tails are complex stuff, often involving armatures for bones and cable mechanisms () to simulate muscles and tendons. Rather that fighting against gravity, we'll instead make it our ally. Thinking of the tail as a pendulum, a single small servo and a little math is all it takes.

> "The cheapest, fastest, and most reliable components are those that aren't there."
>
> — Gordon Bell

It's not the most realistic, but that's not the goal. We're learning...the project is modest in scope, with just a few inexpensive components and minimal soldering.



Parts and tools needed:

- 5V Trinket microcontroller (http://adafru.it/1501) (not the Pro Trinket...just the basic tiny one!)
- Micro servo (). (You can use a servo you already have, but our 3D-printed enclosure is designed around this one () specifically.)
- Power source. Either:
    - 3x AAA battery holder w/switch (http://adafru.it/727) (and some AAA batteries (http://adafru.it/617))
    - JST connector ()
    - Extension cable ()
- or:
    - USB battery pack (http://adafru.it/1959)
    - USB cable
- A tail! You can either use a natural raccoon tail (vendors often sell these at anime conventions and Renaissance faires), or sew a simple tube from synthetic craft fur. Depending how flopsy your tail is, you may need to beef it up a little with some plastic aquarium tubing or similar. It can't be a huge, stuffed plush tail though...the servo we're using is very small.
- 3D printer (but see below).
- #4-40 and #2-56 machine screws and nuts for assembling the enclosure.
- Cable "zip tie" for joining tail to servo clip.

You will also need basic soldering tools and paraphernalia.

But I don't have a 3D printer!

Fear not! With some crafting ingenuity, you can work around this...perhaps a cheap cell phone holster can provide a belt clip. Mint tins make great electronics enclosures (just make sure there's no electrical contact). Parts such as servos can be held at different angles using materials like Shapelock plastic (aka Friendly Plastic, Instamorph, etc.).

You could also use a 3D printing service such as Shapeways ().

Wiring

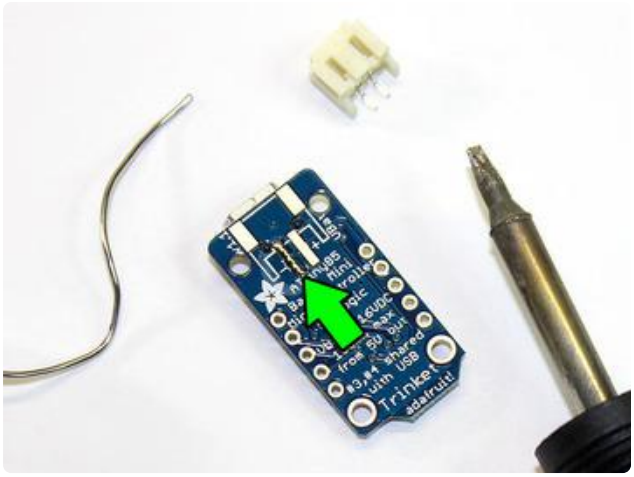
There are a couple of different ways to power the project: either with a 3x AAA battery holder, or with a USB battery pack. Which power source you choose determines how things should be connected...

Wiring for 3X AAA Battery Holder

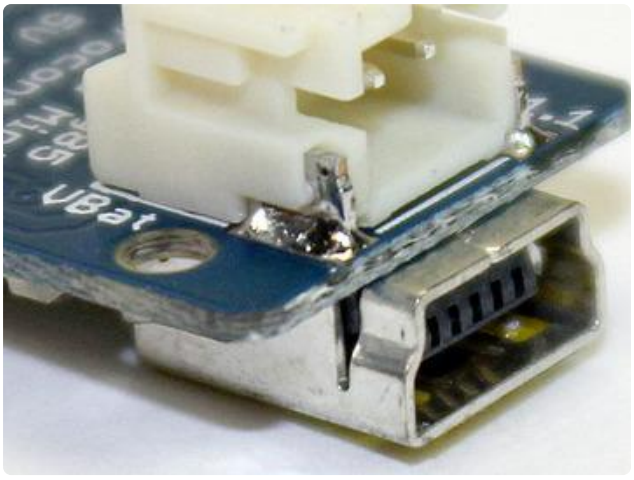
Advantages of using a 3x AAA battery holder are that it's inexpensive and the connection is robust; it's not likely to fall out. You can also get replacement batteries just about anywhere. And the power wire is more discreet.

Alkaline batteries are recommended for this configuration; rechargeable NiMH cells have a lower voltage and won't drive the servo with sufficient force. This is also why we're not using a 3.7V LiPoly battery.

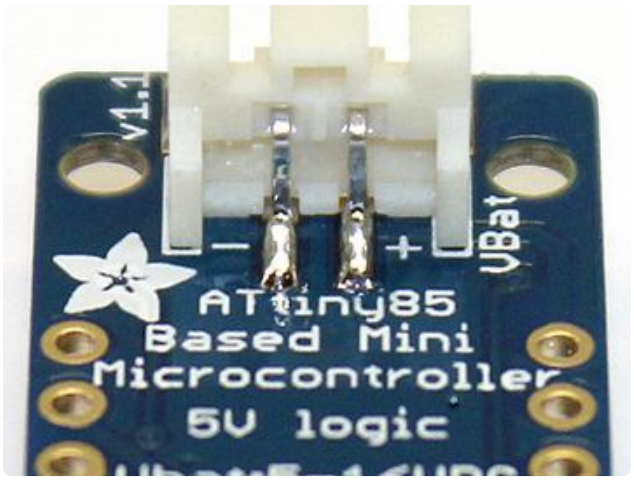
You'll need to add a JST connector to the Trinket, and a JST extension cable. The battery pack can then be carried in a pocket.



Start by “tinning” one of the JST pads on the back of the Trinket...heat the pad and apply solder so the whole surface is covered.

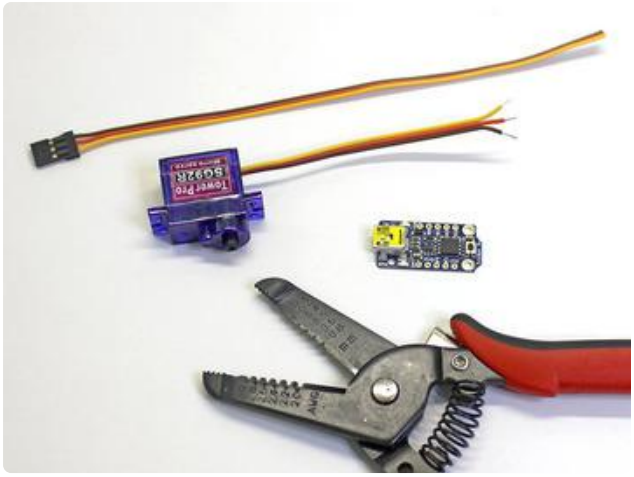


Hold the JST socket in place (tweezers recommended) and re-melt the solder, allowing the part to sink into position.



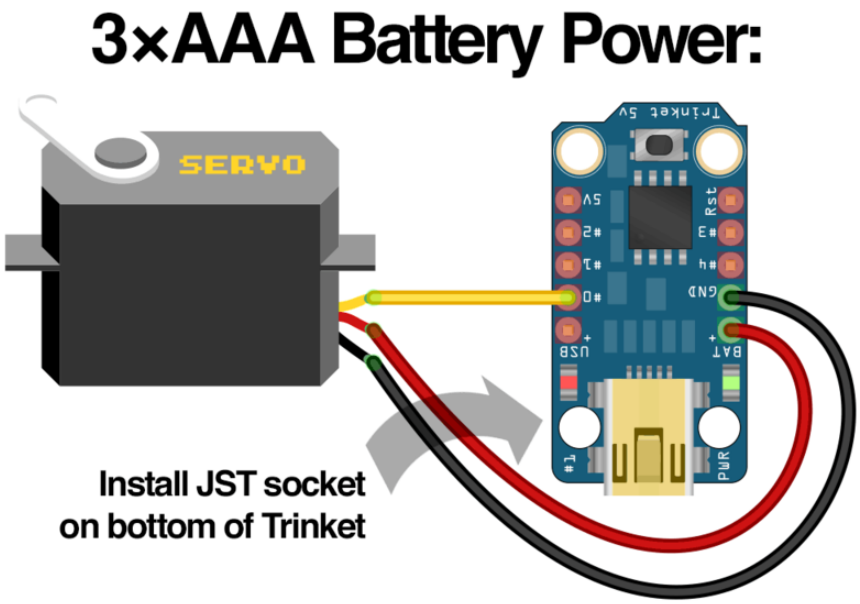
Once this first pin is tacked down, the rest are easy. Remember to heat the parts, then apply solder...do not melt solder on the iron and “wipe” it on the parts...that makes a weak cold solder joint. Properly done, the connections should be shiny and smooth.

Congrats, you’ve done surface-mount soldering!



Clip off the connector from the servo, leaving about 3 inches of cable attached, then strip and tin the ends of the wires. Then solder the following connections:

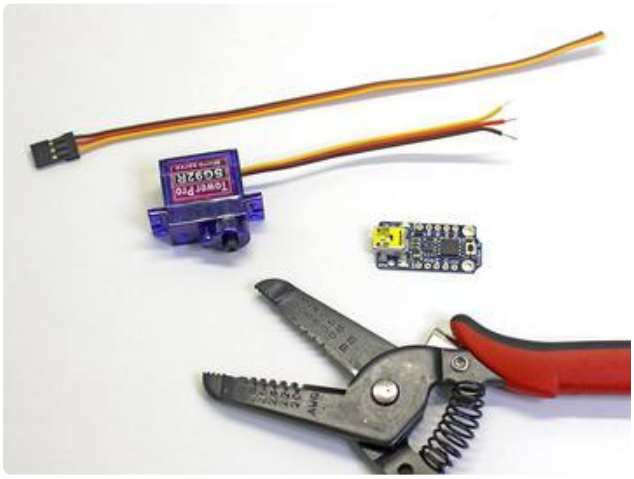
Servo	Trinket
Brown wire	GND
Red wire	BAT +
Orange wire	Pin #0



Wiring for USB Power

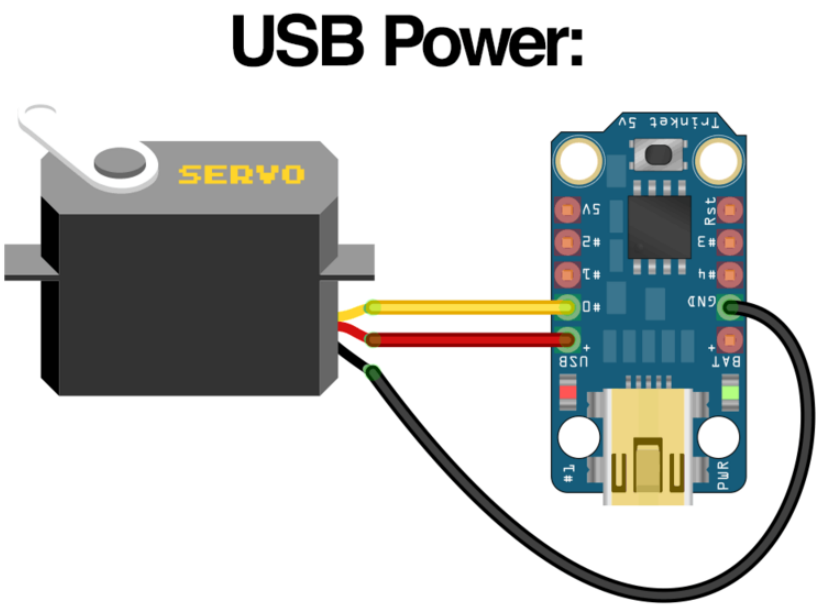
I'm not especially fond of USB batteries for wearable projects — USB plugs lack a latching connector and pull out too easily — but the fact remains that a lot of people already have these battery packs around for charging a cell phone, and may want to put it to other uses.

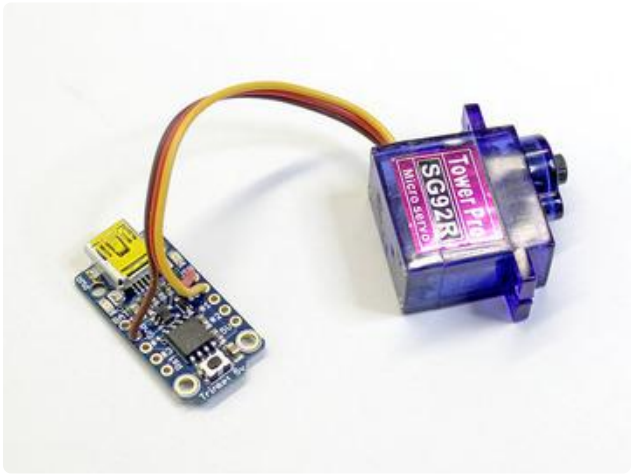
You don't need the JST socket or extension when using USB power, just a suitable USB cable. The battery can be carried in a pocket.



Clip off the connector from the servo, leaving about 3 inches of cable attached, then strip and tin the ends of the wires. Then solder the following connections:

Servo	Trinket
Brown wire	GND
Red wire	USB +
Orange wire	Pin #0





Just three connections! Simple and adorable.

Why the different connections for USB vs AAA?

Servos sometimes need a lot of current. Best way to ensure this is to wire directly to the power source. The Trinket's '5V' pin is only rated for 150 milliamps, but the servo may want as much as 350 mA at times. The 'USB+' pin goes directly to USB, and 'BAT+' directly to the battery pack.

In reality, we probably could get away with using the 5V pin...the servo only operates intermittently...but that would set a bad example. You might decide to adapt this project into new things that make more vigorous use of servos...wired wrong, either the servo response would be anemic or one would risk burning out the voltage regulator. So instead, there's different wiring for different power sources.

Can I use a 3V Trinket instead?

Sure! In the Arduino IDE, select “Trinket 16 MHz” from the Tools→Board menu regardless.

I’m using a different servo with different-colored wires. Will it work?

Most likely, yes. The +V wire is always red. Ground is usually black or sometimes brown. Signal may be orange, yellow, white or blue.

3D Printing and Assembly

Parts

This enclosure is four pieces that are fasten together with machine screws. These parts are optimized to print on FDM based machines with no support material.

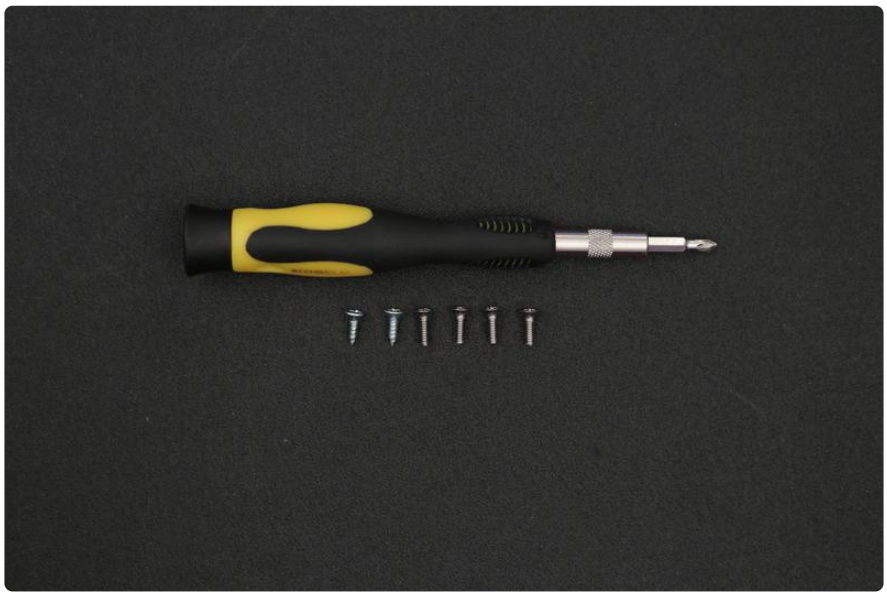
Download Files

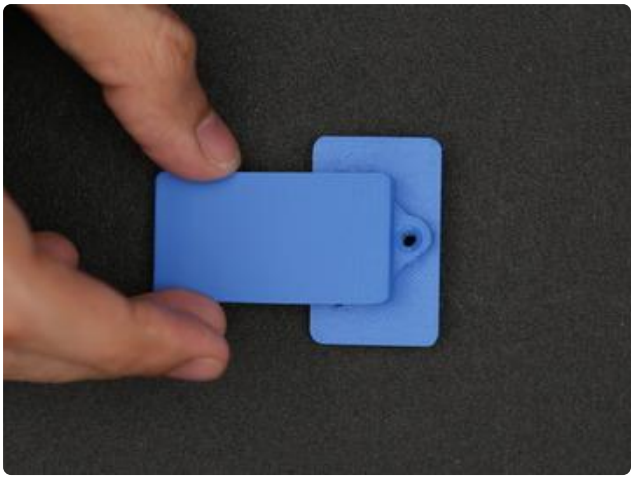
File	Settings	Print Time

servoTail-bot.stl	220c 2 shells 15% infill 0.2 layer height 40 speed	about 15 minutes
servoTail-box.stl	-	about 35 minutes
servoTail-clip.stl	-	about 25 minutes
servoTail-top.stl	-	about 10 minutes
servoTail-attachment.stl	-	about 5 minutes

Hardware

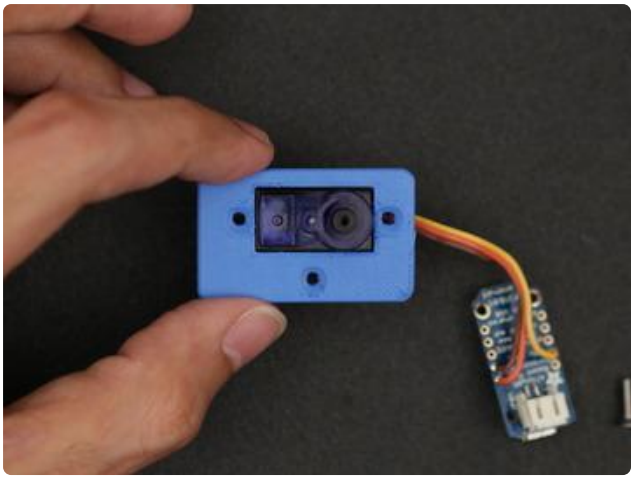
The holes in the parts are sized for #4-40 3/8 flat phillips machine screws. You'll need six screws for this project and a screw driver.

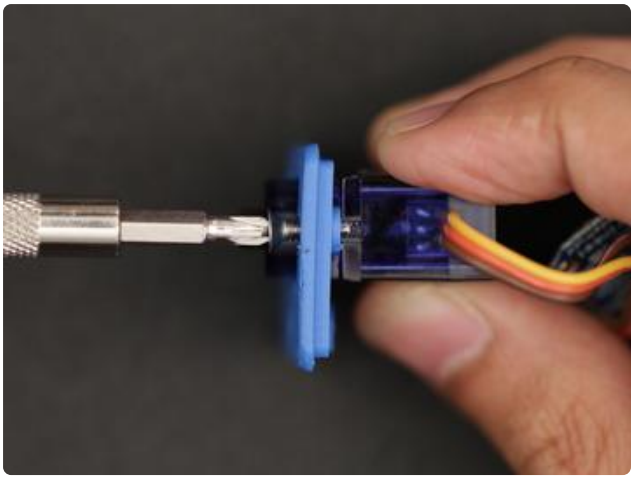




Secure Clip to Bot

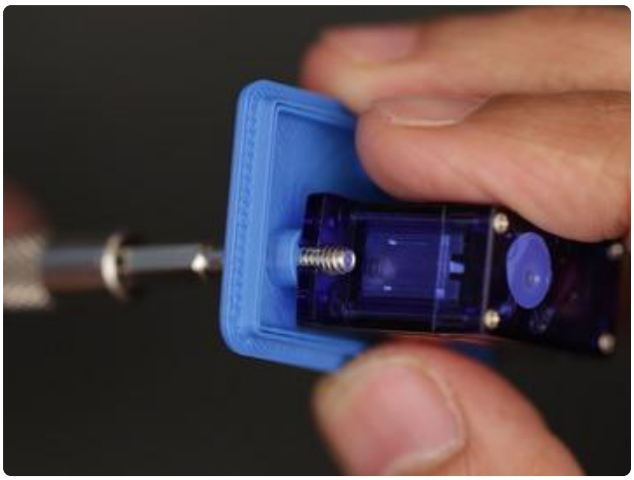
Insert the clip.stl part into the bot.stl part with the ledge part fitting into the slit. They should snap fit tolerance. Insert a single #4-40 screw into the tab so it threads through both parts. Fasten the screw only until the tip of the screw thread peeps through the other side.

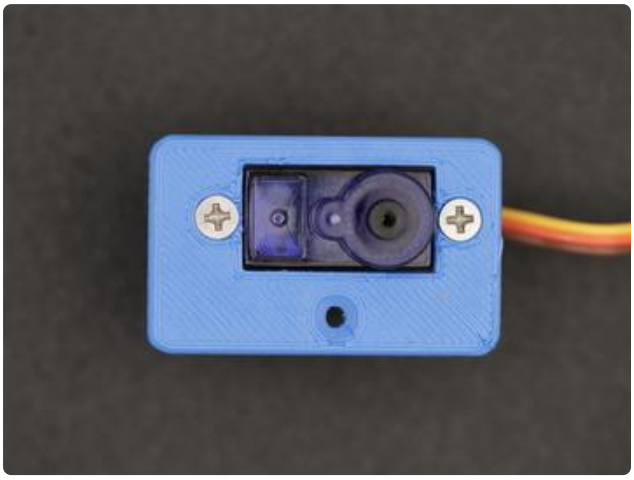


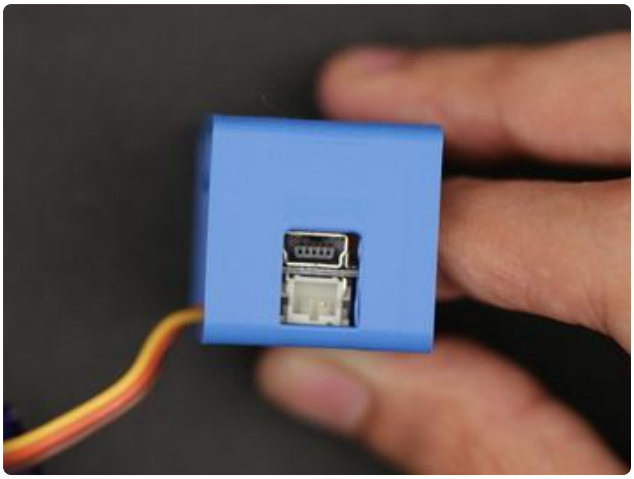


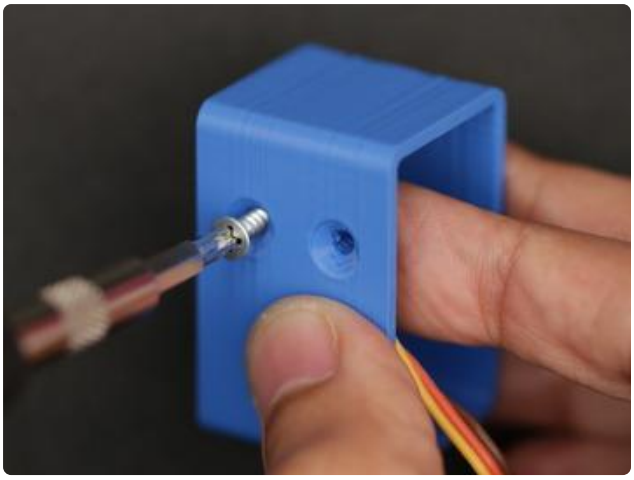
Mounting Servo

Position the top part over the micro servo and line up the standoffs on the cover with the holes in micro servos tabs. Insert and fasten two #4-40 machine screws into the top part. The screws and micro servo should be flush with the surface of the top enclosure part.



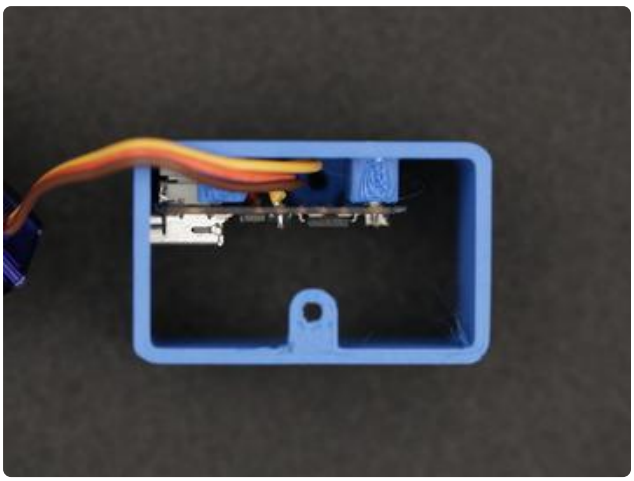


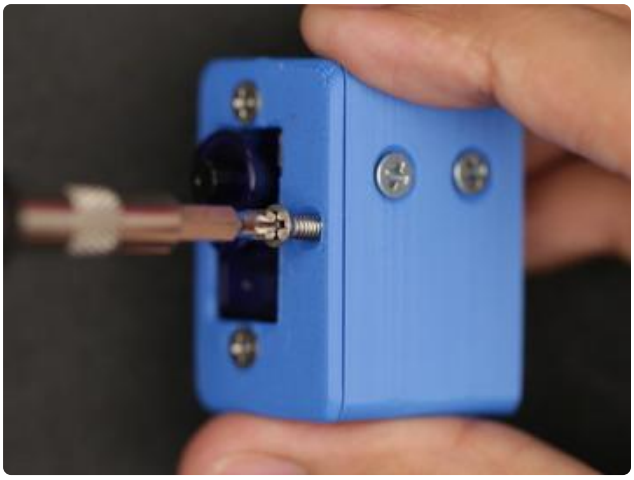




Mounting Trinket

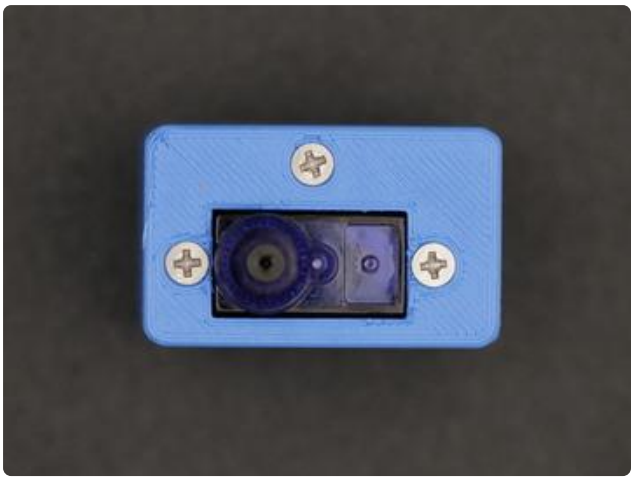
Insert the Trinket into the box.stl part with the USB and JST port facing the cut out. It should be oriented up right with the mounting holes over the standoffs. Insert and fasten two machine screws into the box.stl. The screw heads should be flush with the enclosure.





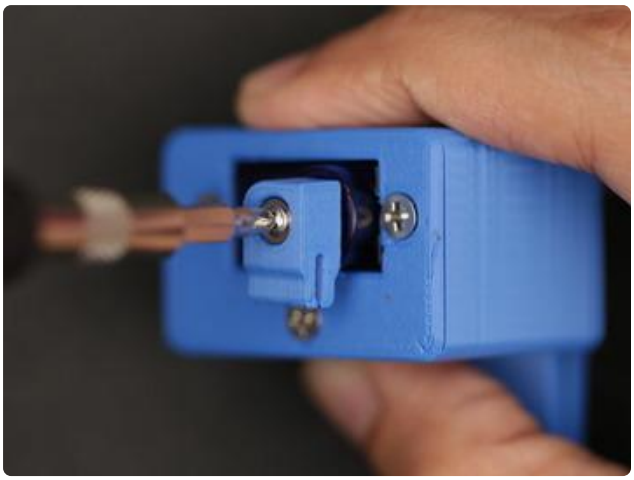
Installing Top

Position the top.stl part over the box.stl part where the lip inserts to the frame. Press it down into place. Insert and fasten a #4-40 machine screw into the mounting hole near the bottom center of the top.stl part.



Install Clip and Bot into Box

Place the bot.stl part over the box.stl part and press it into the part. Fasten the screw all the way through until the screw head is flush with the part.

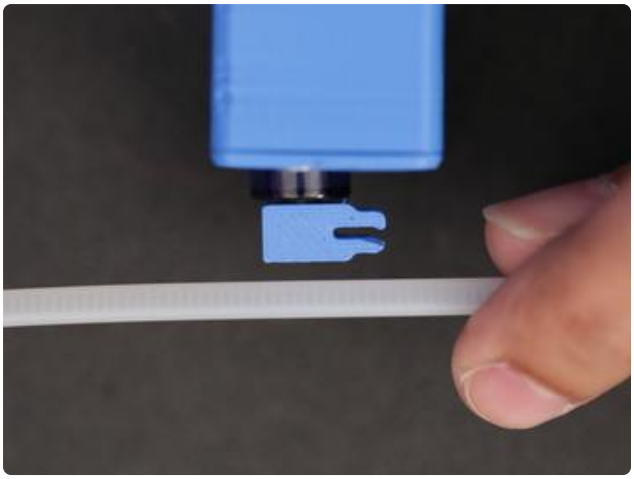


Secure servo horn to tail

The large hole in the attachment is meant to snap fit onto the nub of the servo. The part with the slit is suppose to clip onto the zip tie. You'll want to get this attachment secured to the zip tie before pressing it into the servo tooth. It's a tight fit so you'll need to do a bit of "hard but too hard" type of thing to clip it on.

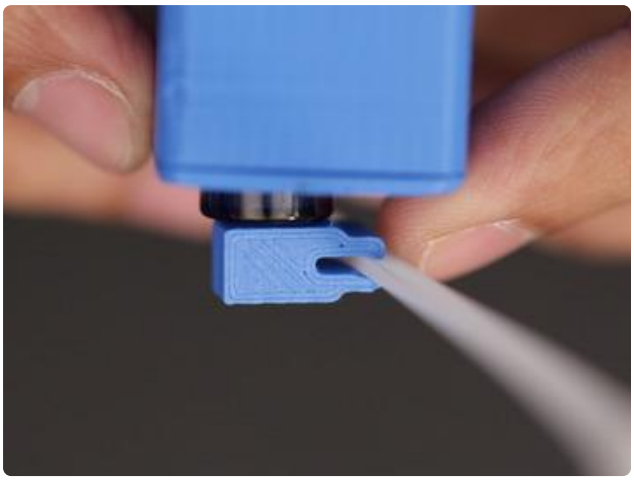
Install attachment to servo

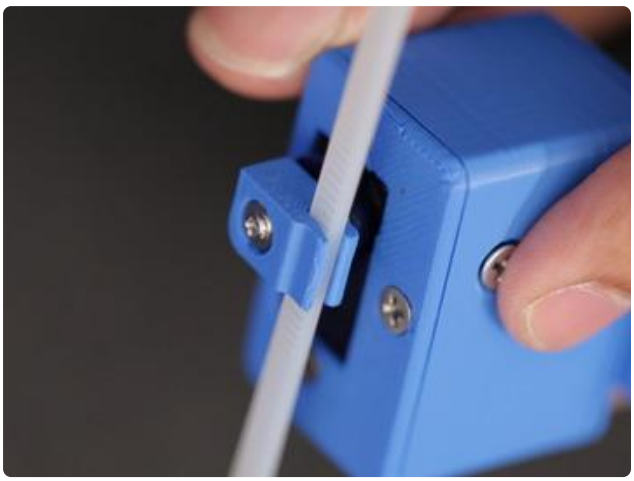
Get the servo teeth in the center with the orientation relative to your clips position. You'll want to test fit this to see which way you need to install the attachment. Once you figure out what's up and down, press the attachment into the servo tooth. Insert and fasten a #2-56 machine screw into the hole of the attachment to secure it to the servo.



Install Ziptie

The slit in the servo horn is sized for a ziptie that's 4.5mm wide by thick 1.4mm. Find the center of the ziptie and gently fit it inside the slit of the servo horn. The ziptie is held in place with friction - apply adhesives for a perminent hold.





Tie the Tail

The tail we used in this project has an hoop near the top. It came with a small bead chain that was threaded through the hoop. It's a great pace to thread the ziptie. String it through and tie it however tight you'd like.

When you run the software presented on the next page, you might find the tail is off-balance. Easily fixed! Just wait for the tail to come to rest, disconnect power, then unscrew the servo horn (the little piece to which our tail is tied) and reinstall it in a neutral (centered) position.

Code

You'll need to have the Arduino IDE software configured for use with the Adafruit Trinket. If you've not done that before, it's explained in this guide ().

Copy the code below and paste it into a new Arduino sketch.

(Instructions continue below, after the code.)

```
            /*
Simple servo tail wagger for Adafruit 5V Trinket (not Pro) microcontroller.
Uses servo on pin 0.  The tail has no 'tendons' -- it's a passive thing,
simply hanging off the servo -- though a weak 'spine' (such as aquarium tube)
adds just enough body to help.  Pendulum math is then used to induce a
reasonable wag effect.

To break up the repetition and appear a little more 'alive,' the speed,
magnitude and duration of the tail wag is randomized (within certain ranges),
and it periodically settles down and stops (adds variety and also saves some
battery).  There's an optimal period (single-swing time) for a given tail
length, but it may randomly go a little faster or slower than this to add
```

```
some 'english' to the wag.

You'll need to calibrate this, editing a few lines below.  TAIL_LENGTH is the
length of the tail in meters (e.g. a 40 cm tail is 0.4 meters); for inches,
multiply by 2.54 to get centimeters, then divide by 100 for meters.
SERVO_MIN and SERVO_MAX are the pulse times (in microseconds) for the leftmost
and rightmost servo positions; though nominally these are 1000 and 2000 usec
(1.0 to 2.0 milliseconds), every servo in reality is a little different, and
you'll need to tune these values for your actual desired swing range.
*/

#ifdef __AVR_ATtiny85__
 #include &lt;avr/power.h&gt;
#endif

// CONFIGURABLE STUFF --------------------------------------------------------

#define TAIL_LENGTH 0.4  // Nominal tail length (meters)
#define SERVO_PIN   0    // Servo is connected here
#define SERVO_MIN   500  // Servo pulse times
#define SERVO_MAX   1800 // in microseconds

// Tail cycles through four states: off, ramp up, steady wag, ramp down.
// Durations are semi-random; this table sets min &amp; max times for each.
static const uint8_t PROGMEM modeTime[4][2] = {
  { 4,  9 }, // 4 to 9 second off time
  { 3,  6 }, // 3 to 6 second ramp up
  { 4, 12 }, // 4 to 12 sec steady wag
  { 2,  5 }  // 2 to 5 sec ramp down
};

// SHOULDN'T NEED TO EDIT BELOW THIS LINE ------------------------------------

#define SERVO_RANGE    (SERVO_MAX - SERVO_MIN)
#define MODE_OFF       0
#define MODE_RAMP_UP   1
#define MODE_HOLD      2
#define MODE_RAMP_DOWN 3

uint8_t  tailMode      = MODE_OFF;
float    wagnitude     = 0.8, // Magnitude of current wag cycle
         period        = M_PI * 2.0 * sqrt(TAIL_LENGTH / 9.8);
uint32_t modeStartTime = 0,
         modeDuration  = 0,
         lastPulseTime = 0;

// SETUP just configures prescaler &amp; enables servo output ----------------------

void setup() {
#if defined(__AVR_ATtiny85__) &amp;&amp; (F_CPU == 16000000L)
  clock_prescale_set(clock_div_1); // 16 MHz Trinket (not Pro) requires this
#endif
  pinMode(SERVO_PIN, OUTPUT);
  randomSeed(analogRead(1));
}

// LOOP does all the tail-waggling math --------------------------------------

void loop() {
  uint32_t t = millis(); // Elapsed time, milliseconds

  // Compare time in current mode against planned duration
  if((t - modeStartTime) &gt; modeDuration) { // Time's up!
    for(;;) {
      if(++tailMode &gt; MODE_RAMP_DOWN)      // Cycle to next mode,
        tailMode = MODE_OFF;               // wrap if needed
      modeDuration = 1000 * random(        // Randomize mode duration
        pgm_read_byte(&amp;modeTime[tailMode][0]),
        pgm_read_byte(&amp;modeTime[tailMode][1]));
```

```
      if(tailMode != MODE_OFF) break;      // If 'off' mode...
      // Randomize magnitude of next wag cycle (70% to 100%)
      wagnitude = (float)random(7, 10) / 10.0;
      // Randomize tail length for next wag cycle (60% to 120%)
      float len = TAIL_LENGTH * (float)random(6, 12) / 10.0;
      // Solve for period (sec) from length (meters):
      period = M_PI * 2.0 * sqrt(len / 9.8);
      delay(modeDuration);                 //   Stop servo,
      t = millis();                        //   revise time and
    }                                      //   mode-cycle again
    modeStartTime = t;                     // Save mode start time
  }

  // Calc amplitude of wag at current time (ramping up/down/steady)
  float a = wagnitude; // Assume MODE_HOLD
  if(tailMode != MODE_HOLD) {
    a = wagnitude * (float)(t - modeStartTime) / (float)modeDuration;
    if(tailMode == MODE_RAMP_DOWN) a = wagnitude - a;
  }

  uint16_t servoPulseLength = SERVO_MIN + (int)((float)SERVO_RANGE *
    ((sin(
    ((float)t / 1000.0)   // Current time in seconds
    / period * M_PI * 2.0 // Seconds to wag cycles
    ) * a)                // Sine wave * amplitude (-1.0 to 1.0)
    + 1.0) * 0.5);        // Convert to integer servo usec pulse

  // Handle servo pulse @ 50 Hz...
  while(((t = micros()) - lastPulseTime) &lt; 20000); // Wait for it...
  digitalWrite(SERVO_PIN, HIGH);
  delayMicroseconds(servoPulseLength);
  digitalWrite(SERVO_PIN, LOW);
  lastPulseTime = t;
}
```

Measure the length of your tail and convert to meters. Perfectly fine to use round “ish” numbers…it’s not rocket science. For example, a 12 inch tail is about 30-ish centimeters, or 0.3 meters.

Look for this line in the Arduino code, and replace the number there with your measurement:

```
#define TAIL_LENGTH 0.4  // Nominal tail length (meters)
```

In the Tools→Board menu, select “Adafruit Trinket 16 MHz.” Connect a USB cable and click the Upload button.

Normally the reset button is used to start the Trinket bootloader and upload code…but inside the enclosure, this can’t be reached. Plugging in the USB cable has the same effect…there’s about a 10 second window to start uploading code to the board.

After uploading…if using the 3xAAA battery pack, unplug the USB cable and switch the battery pack on. The servo will not run off USB in this configuration. This is normal.

If all goes well, the tail will be quiet for a moment, then gradually start to wag. After a few seconds it'll settle down, then start up again, perhaps with a slightly different speed.

If the swinging is off-center, there's a fix. Look for these two lines in the code:

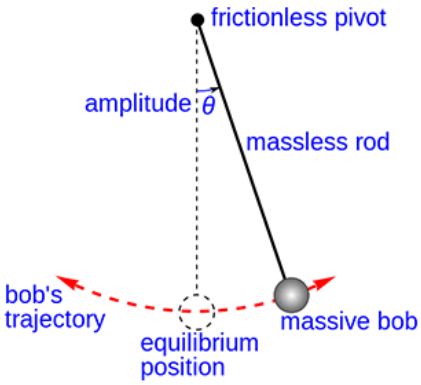
```
#define SERVO_MIN   500  // Servo pulse times
#define SERVO_MAX   1800 // in microseconds
```

Servos expect a continuous series of pulses (about 50 times per second...50 Hz). The "on" time of each pulse indicates the servo position. Nominally this time is said to be between 1 and 2 milliseconds (1,000 and 2,000 microseconds) to represent the full range of motion...but every servo's a little different, and the range of values could go higher or lower. So you may need to experiment with different values and upload the revised code to the board.

If the amount of rotation is acceptable, and it's not running up against the left or right limits of the servo...just off-center...an easier option is just to unscrew the servo horn (the bit to which the tail is tied) and re-attach it at the desired angle.

How does it work?

The code is based on a formula devised by Galileo Galilei () in the early 1600s...that the period (swing time) of a pendulum is much more dependent on its length than the amplitude of its swing...in fact, for small swing ranges the amplitude can be ignored.

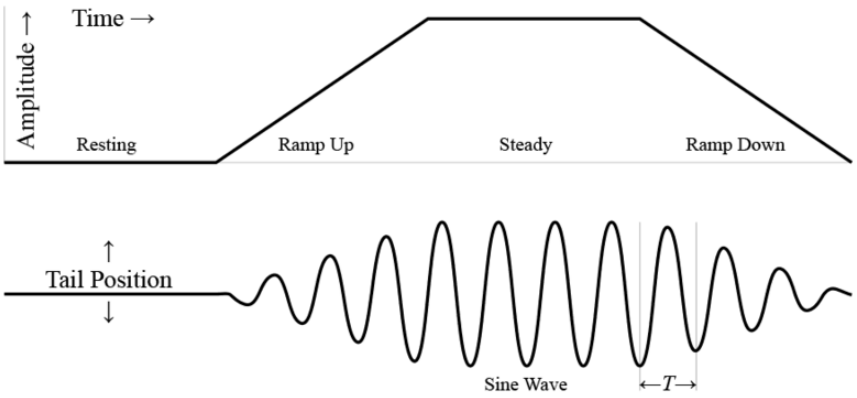


$$T \approx 2\pi\sqrt{\frac{L}{g}}$$

L is the length of the pendulum (our tail), in meters. g is acceleration due to gravity — about 9.8 meters/second^2 on Earth. T is the resulting approximate period in seconds. Then we just use a sine wave matching that period.

The code cycles through four states: resting (no wagging), ramp up, steady wag, and ramp down. During all but the resting phase, the period of the swing remains the

same (using the above formula)...only the amplitude changes. Gradually ramping up imparts just a little bit of extra energy on each cycle...a bit like kicking your legs on a swing. This is how we're able to use such a small and inexpensive servo.



The four states are slightly randomized...it may rest or wag a little more or less on some cycles, and may go a bit faster or swing higher at times. A little unpredictability like this helps make it a little more believable or "alive" — our brains pick up very quickly on obvious repeating patterns!