adafruit learning system

CircuitPython Rotary Trinkey Brightness Crank

Created by Tim C

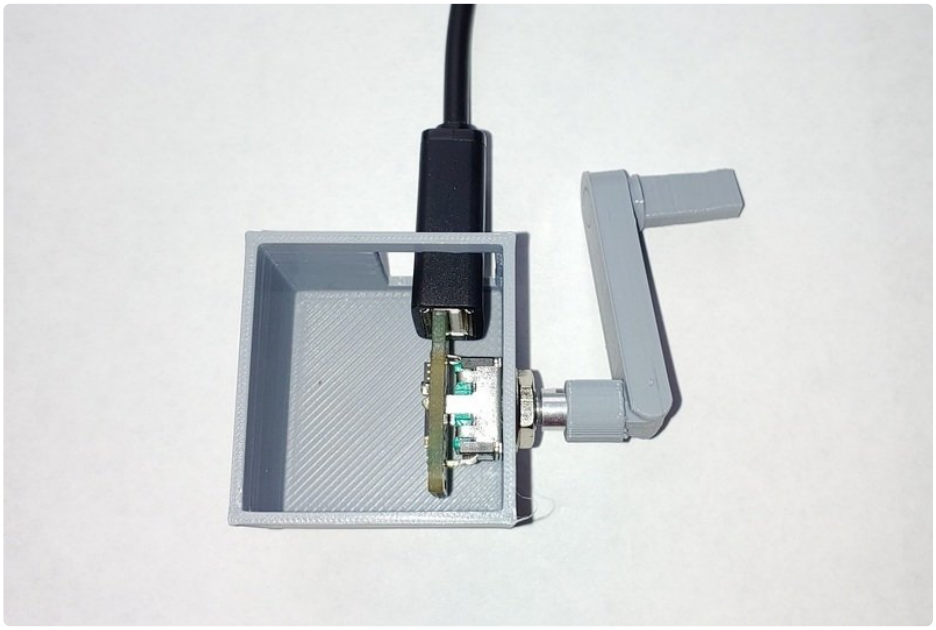
Last updated on 2021-06-30 12:51:41 PM EDT

Guide Contents

Overview

This device will help encourage you to spend less time with social media or tech devices. It will turn the brightness of your device down as long as you aren't cranking the rotary encoder arm. You must continue cranking in order to raise and keep the brightness level up.

This is a good project to show how you can use CircuitPython and HID to interact with mobile devices without having to write an app!



Parts

Your browser does not support the video tag.

Adafruit Rotary Trinkey - USB NeoPixel Rotary Encoder

It's half USB Key, half Adafruit Trinket, half rotary encoder...it's Rotary...

$6.95

In Stock

Add to Cart

Rotary Encoder + Extras

This rotary encoder is the best of the best, its a high quality 24-pulse encoder, with detents and a nice feel. It is panel mountable for placement in a box, or you can plug it into a...

Out of Stock

Out of Stock

---

iOS Lightning to USB OTG Cable

Your iOS phone or tablet may not have a USB port on the bottom but that doesn't mean you can't use it to connect USB devices. Secretly known as a 'Camera Connector' or...

$17.50

In Stock

Add to Cart

Or

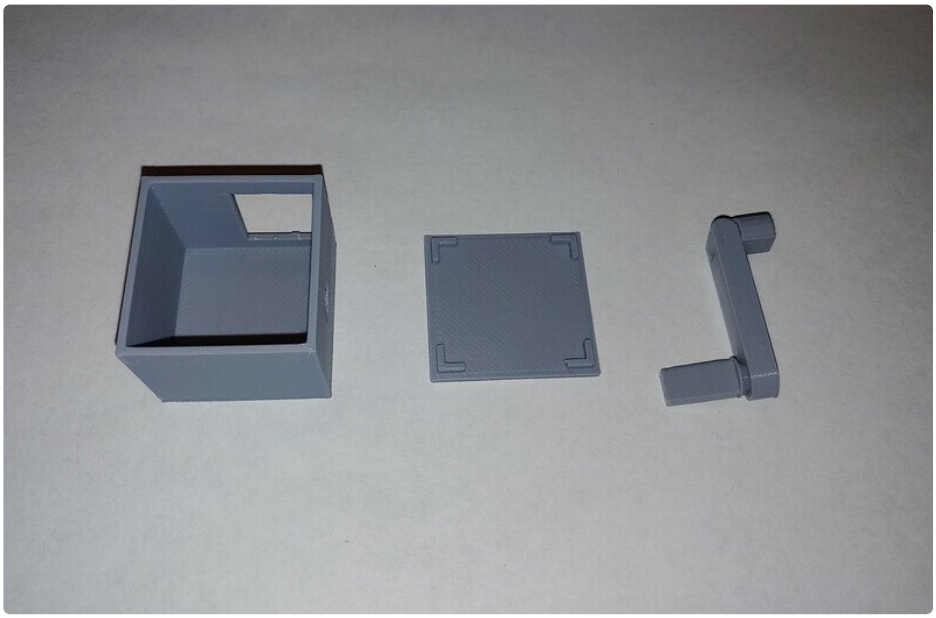
USB OTG Host Cable - MicroB OTG male to A female

This cable looks like a USB micro cable but it isn't! Instead of a USB A Plug, it has a USB A Socket on the end. This cable is designed for use with OTG (On the Go) host devices...

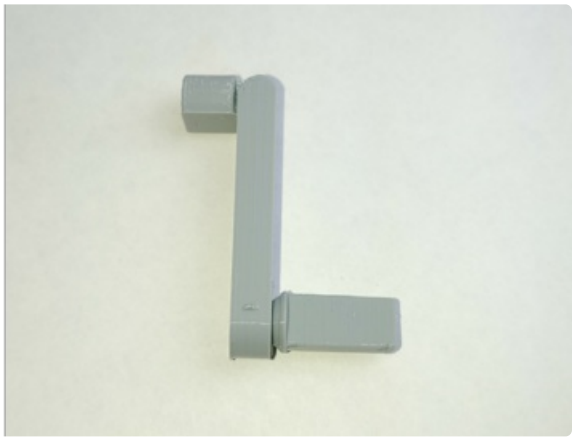
Out of Stock

Out of Stock

3D Printing



Crank Arm

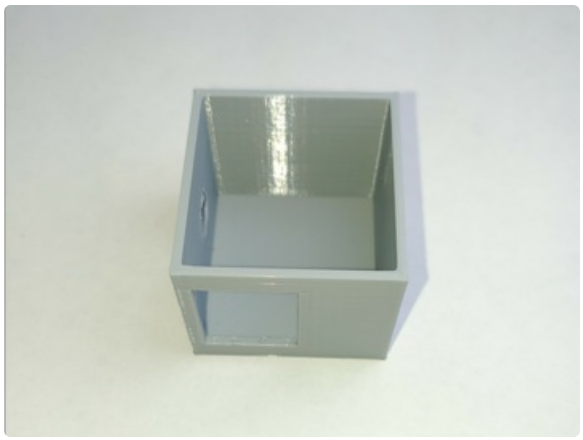


The rotary encoder crank arm comes from the USB HID Crank Controller (https://adafru.it/F12) guide.

https://adafru.it/TA2

https://adafru.it/TA2

Box

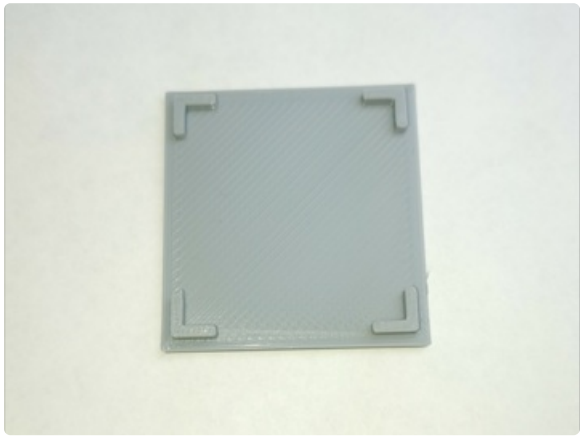


The box has a circle cut out on one side for the rotary encoder, and a rectangular cutout on an adjacent side for the USB cable to be plugged in. It was printed with supports enabled due to the overhang on the USB cable cutout.

https://adafru.it/TA3

https://adafru.it/TA3

Lid



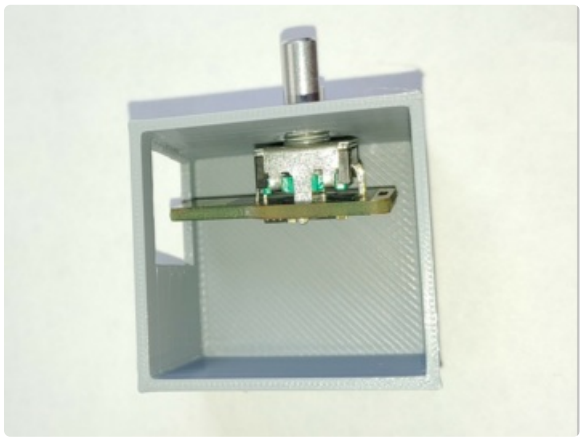
The lid is friction fit to the top of the box with the tabs in each corner. It's slightly rectangular so if it doesn't seem to fit try turning 90 degrees.

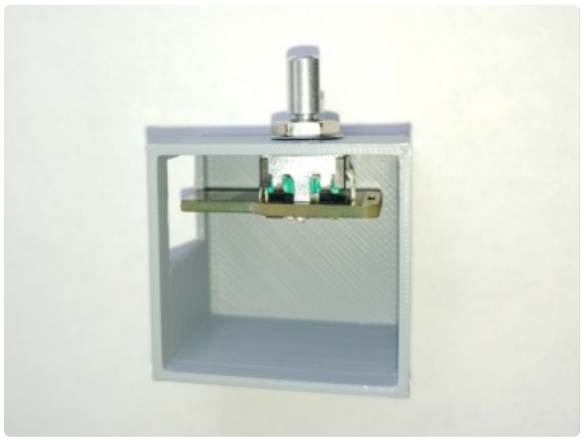
https://adafru.it/TA4

https://adafru.it/TA4

Assembly

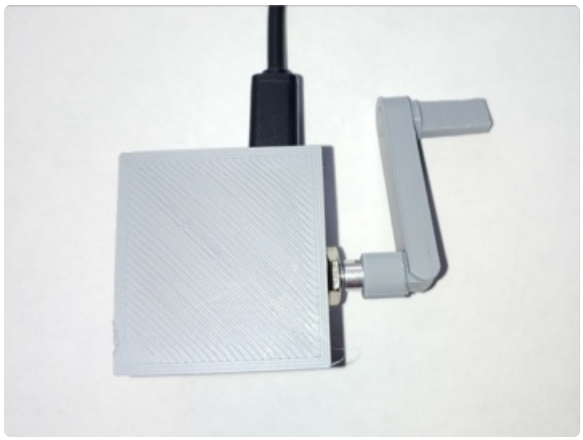
Attach Rotary Trinkey





Fit the rotary encoder through the hole in the side of the box and then tighten it on with the encoder's nut. The USB plug edge should face the cutout on the other side of the box.

Lid and Crank Arm



Push the lid onto the top. It only fits one way; if it's not fitting, try turning it 90 degrees.

Push the crank arm onto the encoder.

CircuitPython Setup

Download the project files with the button below. Unzip and copy/paste the **code.py** to your **CIRCUITPY** drive. The only libraries required by this project are included in the firmware so you don't need to copy them to the drive.

```
"""
Rotary Trinkey gadget that forces you to crank the handle in order
to keep the brightness up on your phone or tablet.
"""

import time
import math
import board
import digitalio
import rotaryio
import usb_hid
from adafruit_hid.consumer_control import ConsumerControl
from adafruit_hid.consumer_control_code import ConsumerControlCode

# how frequently we check the encoder value and apply brightness changes
ACTION_INTERVAL = 3  # seconds

# if encoder value increases at least this much
# then brightness stays the same
# if encoder value increases less than this
# then brightness goes down
STAY_EVEN_CHANGE_THRESHOLD = 60

# if encoder value increases at least this much
# then brightness goes up
INCREASE_CHANGE_THRESHOLD = 95

# timestamp of last time an action occurred
LAST_ACTION_TIME = 0

# encoder value variable
CUR_VALUE = 0

# pause state
PAUSED = False

cc = ConsumerControl(usb_hid.devices)

encoder = rotaryio.IncrementalEncoder(board.ROTA, board.ROTB)
switch = digitalio.DigitalInOut(board.SWITCH)
switch.switch_to_input(pull=digitalio.Pull.DOWN)

switch_state = None

# previous encoder position variable
last_position = encoder.position
```

```
# previous switch variable
prev_switch_value = False

while True:
    now = time.monotonic()

    if switch.value and not prev_switch_value:
        print("toggling pause")
        PAUSED = not PAUSED

        if not PAUSED:
            LAST_ACTION_TIME = now

    prev_switch_value = switch.value

    # read current encoder value
    current_position = encoder.position
    position_change = int(current_position - last_position)

    # positive change
    if position_change > 0:
        for _ in range(position_change):
            CUR_VALUE += position_change

    # negative change
    elif position_change < 0:
        for _ in range(-position_change):
            # use absolute value to convert to positive
            CUR_VALUE += int(math.fabs(position_change))

    last_position = current_position

    if not PAUSED:
        # is it time for an action?
        if now > LAST_ACTION_TIME + ACTION_INTERVAL:
            print(CUR_VALUE)

            # update previous time variable
            LAST_ACTION_TIME = now

            # less than stay even threshold
            if CUR_VALUE < STAY_EVEN_CHANGE_THRESHOLD:
                cc.send(ConsumerControlCode.BRIGHTNESS_DECREMENT)
                print("brightness down")

            # more than stay even threshold
            elif CUR_VALUE < INCREASE_CHANGE_THRESHOLD:
                print("stay even")

            # more than increase threshold
            else:
                cc.send(ConsumerControlCode.BRIGHTNESS_INCREMENT)
                print("brightness up")

            # reset encoder value
            CUR_VALUE = 0
```

Code Walk-Through

Brightness Consumer Codes

To change the brightness up and down we can send HID media consumer control codes for brightness up and down. The `adafruit_hid` library makes sending these easy.

Main Logic

Every few seconds, `ACTION_INTERVAL`, the program will check how many values the rotary encoder has changed by.

- If it is less than `STAY_EVEN_CHANGE_THRESHOLD`, we send the brightness down code.
- If it is above `STAY_EVEN_CHANGE_THRESHOLD`, it will remain steady not changing the brightness.
- If it's over `INCREASE_CHANGE_THRESHOLD`, we will send the brightness up code.

Different devices will have different brightness levels. You can change the threshold variables or `ACTION_INTERVAL` to suit your device or to make it easier or harder to keep the brightness up.

Alternate Code - Pausing

In the version below, pressing the rotary encoder button will pause the device so that it won't change the brightness until resumed by pressing the button again.

```
"""
Rotary Trinkey gadget that forces you to crank the handle in order
to keep the brightness up on your phone or tablet.
"""

import time
import math
import board
import digitalio
import rotaryio
import usb_hid
from adafruit_hid.consumer_control import ConsumerControl
from adafruit_hid.consumer_control_code import ConsumerControlCode

# how frequently we check the encoder value and apply brightness changes
ACTION_INTERVAL = 3  # seconds

# if encoder value increases at least this much
# then brightness stays the same
# if encoder value increases less than this
# then brightness goes down
STAY_EVEN_CHANGE_THRESHOLD = 60
```

```
# if encoder value increases at least this much
# then brightness goes up
INCREASE_CHANGE_THRESHOLD = 95

# timestamp of last time an action occurred
LAST_ACTION_TIME = 0

# encoder value variable
CUR_VALUE = 0

# pause state
PAUSED = False

cc = ConsumerControl(usb_hid.devices)

encoder = rotaryio.IncrementalEncoder(board.ROTA, board.ROTB)
switch = digitalio.DigitalInOut(board.SWITCH)
switch.switch_to_input(pull=digitalio.Pull.DOWN)

switch_state = None

# previous encoder position variable
last_position = encoder.position

# previous switch variable
prev_switch_value = False

while True:
    now = time.monotonic()

    if switch.value and not prev_switch_value:
        print("toggling pause")
        PAUSED = not PAUSED

        if not PAUSED:
            LAST_ACTION_TIME = now

    prev_switch_value = switch.value

    # read current encoder value
    current_position = encoder.position
    position_change = int(current_position - last_position)

    # positive change
    if position_change > 0:
        for _ in range(position_change):
            CUR_VALUE += position_change

    # negative change
    elif position_change < 0:
        for _ in range(-position_change):
            # use absolute value to convert to positive
            CUR_VALUE += int(math.fabs(position_change))

    last_position = current_position

    if not PAUSED:
```

```
    if not PAUSED:
        # is it time for an action?
        if now > LAST_ACTION_TIME + ACTION_INTERVAL:
            print(CUR_VALUE)

            # update previous time variable
            LAST_ACTION_TIME = now

            # less than stay even threshold
            if CUR_VALUE < STAY_EVEN_CHANGE_THRESHOLD:
                cc.send(ConsumerControlCode.BRIGHTNESS_DECREMENT)
                print("brightness down")

            # more than stay even threshold
            elif CUR_VALUE < INCREASE_CHANGE_THRESHOLD:
                print("stay even")

            # more than increase threshold
            else:
                cc.send(ConsumerControlCode.BRIGHTNESS_INCREMENT)
                print("brightness up")

            # reset encoder value
            CUR_VALUE = 0
```