PowerWash Simulator Nozzle Controller

Created by John Park

https://learn.adafruit.com/powerwash-simulator-nozzle-controller

Last updated on 2023-11-08 04:08:38 PM EST
## Table of Contents

**Overview**  
- Parts  
- PC and Game  

**Assemble the Circuit**  
- Piezo Element  
- JST Connection  
- Wiichuck Adapter  
- 9-DOF Sensor  

**CircuitPython**  
- CircuitPython Quickstart  
- Safe Mode  
- Flash Resetting UF2  

**Code the Controller**  
- Text Editor  
- Sensor Calibration  
- Download the Project Bundle  
- How it Works  

**Build the Controller**  
- Disassembly  
- Fit the Parts  
- Wire Strain Relief  
- Piezo Driver Mount  
- Piezo Element Mount  
- BNO055 Mount  
- Fasten the Nozzle Halves  

**Blast Away Some Grime!**
Overview

If you love the satisfaction of pressure washing dirt off of driveways, house siding, benches, and more, but want to do so virtually, than you may want to keep the party going with the [PowerWash Simulator](https://www.adafruit.com/product/4900) game for Windows PCs. However, mouse and keyboard control are a bit of a letdown compared to the real thing. That's why we've created the PowerWash Simulator Nozzle Controller!

Heft a real pressure washer nozzle in your hands and aim at all that dirt and grime to blast it away. A CircuitPython-powered QT Py RP2040 with BNO055 9-DOF orientation sensor know where you're aiming! A connected Wiichuck provides extra controls for triggering the water, movement, and more.

Parts

Adafruit QT Py RP2040
What a cutie pie! Or is it... a QT Py? This diminutive dev board comes with one of our new favorite chip, the RP2040. It's been made famous in the new [https://www.adafruit.com/product/4900](https://www.adafruit.com/product/4900)
Adafruit 9-DOF Absolute Orientation IMU Fusion Breakout - BNO055
If you've ever ordered and wire up a 9-DOF sensor, chances are you've also realized the challenge of turning the sensor data from an accelerometer, gyroscope, and magnetometer...
https://www.adafruit.com/product/4646

Adafruit Wii Nunchuck Breakout Adapter
Dig out that old Wii controller and use it as a sleek controller for your next robot if you like. The Adafruit Adafruit Wii Nunchuck Breakout Adapter fits snugly into the Wii connector...
https://www.adafruit.com/product/4836

Adafruit STEMMA Piezo Driver Amp - PAM8904
Piezos make noise when you put an AC voltage across them: and the bigger the Vpp the louder they are. With your standard 3V logic microcontroller you can make 3Vpp with a PWM out, or...
https://www.adafruit.com/product/5791

Wii controller (Nunchuck / Wiichuck)
This is a generic Wii Nunchuck controller, we haven't tried it with a Wii but it does work great with the Video Game shield, and all the microcontroller code we tried. May come in...
https://www.adafruit.com/product/342
### Hardware

<table>
<thead>
<tr>
<th>Description</th>
<th>Product ID</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>14mm diameter</strong></td>
<td></td>
</tr>
<tr>
<td><strong>1 x STEMMA QT / Qwiic JST SH 4-Pin Cable</strong></td>
<td><a href="https://www.adafruit.com/product/1740">https://www.adafruit.com/product/1740</a></td>
</tr>
<tr>
<td>400mm long</td>
<td></td>
</tr>
<tr>
<td><strong>1 x STEMMA QT / Qwiic JST SH 4-pin Cable</strong></td>
<td><a href="https://www.adafruit.com/product/5385">https://www.adafruit.com/product/5385</a></td>
</tr>
<tr>
<td>100mm Long</td>
<td></td>
</tr>
<tr>
<td><strong>1 x STEMMA JST PH 2mm 3-Pin to Male Header</strong></td>
<td><a href="https://www.adafruit.com/product/3893">https://www.adafruit.com/product/3893</a></td>
</tr>
<tr>
<td>Cable 200mm</td>
<td></td>
</tr>
<tr>
<td><strong>1 x USB Type A to Type C Cable</strong></td>
<td><a href="https://www.adafruit.com/product/4474">https://www.adafruit.com/product/4474</a></td>
</tr>
<tr>
<td>approx 1 meter / 3 ft long</td>
<td></td>
</tr>
<tr>
<td><strong>1 x USB Extension Cable</strong></td>
<td><a href="https://www.adafruit.com/product/993">https://www.adafruit.com/product/993</a></td>
</tr>
<tr>
<td>3 meters / 10 ft long</td>
<td></td>
</tr>
<tr>
<td><strong>2 x Thick Double-Sided Rectangle Foam Tape</strong></td>
<td><a href="https://www.adafruit.com/product/5019">https://www.adafruit.com/product/5019</a></td>
</tr>
<tr>
<td>2 pieces</td>
<td></td>
</tr>
</tbody>
</table>

### PC and Game

You'll need a Windows PC and the **PowerWash Simulator** game as well. This is not tested with console versions of the game.
Assemble the Circuit

This is an almost entirely plug-and-play circuit. You'll use STEMMA QT cables to connect the BNO055 and Wiichuck adapter to the QT Py. The piezo element connects to the driver board with screw terminals. The only soldering to do is the 3-pin STEMMA JST-PH cable's bare end to the GND, 3V, and MO pins of the QT Py.
Piezo Element

Insert the wires of the piezo element each into a terminal on the piezo driver board as shown, then tighten the two screws to secure.
JST Connection
Use the 3-wire JST-PH cable to connect the QT Py to the piezo driver. Plug the connector into the driver board.

Cut the Dupont connectors off of the wires and strip a short bit of insulation from each wire.

Optionally, slide a bit of heat shrink tubing over the wires to keep them neat.

Solder the wires to the QT Py:
black to GND
red to 3V
white to MO
Wiichuck Adapter

Plug the Wiichuck adapter into the BNO055’s right side STEMMA QT port using the shorter STEMMA QT cable.

Plug the Wiichuck into the adapter, being mindful to plug it in "notch side up" as indicated on the board.
9-DOF Sensor
Plug the QT Py into the BNO055's left side STEMMA QT port using the longer STEMMA QT cable.

The circuit is all ready for coding!

CircuitPython

CircuitPython is a derivative of MicroPython designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files on the CIRCUITPY drive to iterate.

CircuitPython Quickstart

Follow this step-by-step to quickly get CircuitPython running on your board.
Download the latest version of CircuitPython for this board via circuitpython.org

Click the link above to download the latest CircuitPython UF2 file.

Save it wherever is convenient for you.

To enter the bootloader, hold down the BOOT/BOOTSEL button (highlighted in red above), and while continuing to hold it (don't let go!), press and release the reset button (highlighted in blue above). Continue to hold the BOOT/BOOTSEL button until the RPI-RP2 drive appears!

If the drive does not appear, release all the buttons, and then repeat the process above.

You can also start with your board unplugged from USB, press and hold the BOOTSEL button (highlighted in red above), continue to hold it while plugging it into USB, and wait for the drive to appear before releasing the button.
A lot of people end up using charge-only USB cables and it is very frustrating! Make sure you have a USB cable you know is good for data sync.

You will see a new disk drive appear called RPI-RP2.

Drag the adafruit_circuitpython_etc.uf2 file to RPI-RP2.

The RPI-RP2 drive will disappear and a new disk drive called CIRCUITPY will appear.

That's it, you're done! :)

Safe Mode

You want to edit your code.py or modify the files on your CIRCUITPY drive, but find that you can't. Perhaps your board has gotten into a state where CIRCUITPY is read-
only. You may have turned off the CIRCUITPY drive altogether. Whatever the reason, safe mode can help.

Safe mode in CircuitPython does not run any user code on startup, and disables auto-reload. This means a few things. First, safe mode bypasses any code in boot.py (where you can set CIRCUITPY read-only or turn it off completely). Second, it does not run the code in code.py. And finally, it does not automatically soft-reload when data is written to the CIRCUITPY drive.

Therefore, whatever you may have done to put your board in a non-interactive state, safe mode gives you the opportunity to correct it without losing all of the data on the CIRCUITPY drive.

**Entering Safe Mode**

To enter safe mode when using CircuitPython, plug in your board or hit reset (highlighted in red above). Immediately after the board starts up or resets, it waits 1000ms. On some boards, the onboard status LED (highlighted in green above) will blink yellow during that time. If you press reset during that 1000ms, the board will start up in safe mode. It can be difficult to react to the yellow LED, so you may want to think of it simply as a slow double click of the reset button. (Remember, a fast double click of reset enters the bootloader.)

**In Safe Mode**

If you successfully enter safe mode on CircuitPython, the LED will intermittently blink yellow three times.

If you connect to the serial console, you'll find the following message.

```
Auto-reload is off.
Running in safe mode! Not running saved code.

CircuitPython is in safe mode because you pressed the reset button during boot. Press again to exit safe mode.

Press any key to enter the REPL. Use CTRL-D to reload.
```

You can now edit the contents of the CIRCUITPY drive. Remember, your code will not run until you press the reset button, or unplug and plug in your board, to get out of safe mode.
Flash Resetting UF2

If your board ever gets into a really weird state and doesn't even show up as a disk drive when installing CircuitPython, try loading this 'nuke' UF2 which will do a 'deep clean' on your Flash Memory. You will lose all the files on the board, but at least you'll be able to revive it! After loading this UF2, follow the steps above to re-install CircuitPython.

Download flash erasing "nuke" UF2

Code the Controller

Text Editor

Adafruit recommends using the Mu editor for editing your CircuitPython code. You can get more info in this guide.

Alternatively, you can use any text editor that saves simple text files.

Sensor Calibration

The first step, before putting the final code on your board, is to calibrate the sensor for your location. Since the magnetic sensor orientation is based on the Earth's magnetic field, your location matters!

Follow the steps shown here to get your sensor offset values, which will be printed to the REPL for you. They'll looks something like this:

```
Offsets_Magnetometer: (263, 315, 146)
Offsets_Gyroscope: (-1, 0, -1)
Offsets_Accelerometer: (-48, -13, -23)
```

Copy those values, and then paste them into the main code.py file in the lines in the BNO055 offsets section, which looks like this:

```
sensor.offsets_magnetometer = (263, 315, 146)
sensor.offsets_gyroscope = (-1, 0, -1)
sensor.offsets_accelerometer = (-48, -13, -23)
printd(f"offsets_magnetometer set to:
```
Download the Project Bundle

Your project will use a specific set of CircuitPython libraries, and the code.py file. To get everything you need, click on the Download Project Bundle link below, and uncompress the .zip file.

Connect your computer to the board via a known good USB power+data cable. A new flash drive should show up as CIRCUITPY.

Drag the contents of the uncompressed bundle directory onto your board's CIRCUITPY drive, replacing any existing files or directories with the same names, and adding any new ones that are necessary.

```python
# SPDX-FileCopyrightText: 2023 JG for Cedar Grove Maker Studios
# SPDX-License-Identifier: MIT

"""bno055_calibrator.py"
================================================================================
A CircuitPython module for calibrating the BNo055 9-DoF sensor. After manually calibrating the sensor, the module produces calibration offset tuples for use in project code.

* Author(s): JG for Cedar Grove Maker Studios

Implementation Notes
---------------------
**Hardware:**
* Adafruit BNo055 9-DoF sensor
**Software and Dependencies:**
* Driver library for the sensor in the Adafruit CircuitPython Library Bundle
* Adafruit CircuitPython firmware for the supported boards: https://circuitpython.org/downloads

```
M4G_MODE = 0x0A
NDOF_FMC_OFF_MODE = 0x0B
NDOF_MODE = 0x0C

# Uncomment these lines for UART interface connection
# uart = board.UART()
# sensor = adafruit_bno055.BNO055_UART(uart)

# Instantiate I2C interface connection
# i2c = board.I2C()  # For board.SCL and board.SDA
i2c = board.STEMMA_I2C()  # For the built-in STEMMA QT connection
sensor = adafruit_bno055.BNO055_I2C(i2c)
sensor.mode = Mode.NDOF_MODE  # Set the sensor to NDOF_MODE

print("Magnetometer: Perform the figure-eight calibration dance.")
while not sensor.calibration_status[3] == 3:
    # Calibration Dance Step One: Magnetometer
    # Move sensor away from magnetic interference or shields
    # Perform the figure-eight until calibrated
    print(f"Mag Calib Status: {100 / 3 * sensor.calibration_status[3]:3.0f}%")
    time.sleep(1)
print("... CALIBRATED")
time.sleep(1)

print("Accelerometer: Perform the six-step calibration dance.")
while not sensor.calibration_status[2] == 3:
    # Calibration Dance Step Two: Accelerometer
    # Place sensor board into six stable positions for a few seconds each:
    # 1) x-axis right, y-axis up, z-axis away
    # 2) x-axis up, y-axis left, z-axis away
    # 3) x-axis left, y-axis down, z-axis away
    # 4) x-axis down, y-axis right, z-axis away
    # 5) x-axis left, y-axis right, z-axis up
    # 6) x-axis right, y-axis left, z-axis down
    # Repeat the steps until calibrated
    print(f"Accel Calib Status: {100 / 3 * sensor.calibration_status[2]:3.0f}%")
    time.sleep(1)
print("... CALIBRATED")
time.sleep(1)

print("Gyroscope: Perform the hold-in-place calibration dance.")
while not sensor.calibration_status[1] == 3:
    # Calibration Dance Step Three: Gyroscope
    # Place sensor in any stable position for a few seconds
    # (Accelerometer calibration may also calibrate the gyro)
    print(f"Gyro Calib Status: {100 / 3 * sensor.calibration_status[1]:3.0f}%")
    time.sleep(1)
print("... CALIBRATED")
time.sleep(1)

print("\nCALIBRATION COMPLETED")
print("Insert these preset offset values into project code:")
print(f" Offsets_Magnetometer:  {sensor.offsets_magnetometer}"
print(f" Offsets_Gyroscope:     {sensor.offsets_gyroscope}"
print(f" Offsets_Accelerometer: {sensor.offsets_accelerometer}"

How it Works

Here's how the code works.
The code imports necessary libraries for working with the hardware, including the BNO055 sensor, USB HID interfaces for mouse and keyboard, and the Wiichuck adapter.

```python
import time
import math
import board
from simpleio import map_range, tone
import adafruit_bno055
import usb_hid
from adafruit_hid.mouse import Mouse
from adafruit_hid.keycode import Keycode
from adafruit_hid.keyboard import Keyboard
from adafruit_hid.keyboard_layout_us import KeyboardLayoutUS
from adafruit_nunchuk import Nunchuk
```

Several constants are defined, including debug flags, rates for cursor movement, keycodes for buttons, sensitivity thresholds, and debounce times.

You can toggle the `DEBUG` variable to see more info print to the REPL during coding.

The `CURSOR` variable can be set to true while you're coding to prevent the cursor from flying around the screen!

```python
DEBUG = False
CURSOR = True  # use to toggle cursor movement during testing/use
SENSOR_PACKET_FACTOR = 10  # Ratio of BNO055 data packets per Wiichuck packet
HORIZONTAL_RATE = 127  # mouse x speed
VERTICAL_RATE = 63  # mouse y speed
WII_C_KEY_1 = Keycode.R  # rotate nozzle
WII_C_KEY_2 = Keycode.C  # aim mode
WII_PITCH_UP = 270  # value to trigger wiichuck up state
WII_PITCH_DOWN = 730  # value to trigger wiichuck down state
WII_ROLL_LEFT = 280  # value to trigger wiichuck left state
WII_ROLL_RIGHT = 740  # value to trigger wiichuck right state
TAP_THRESHOLD = 6  # Tap sensitivity threshold; depends on the physical sensor mount
TAP_DEBOUNCE = 0.3  # Time for accelerometer to settle after tap (seconds)
```

You'll instantiate I2C, set up HID mouse and keyboard, the Wiichuck, and the BNO055 sensor next.

```python
# Instantiate I2C interface connection
# i2c = board.I2C()  # For board.SCL and board.SDA
i2c = board.STEMMA_I2C()  # For the built-in STEMMA QT connection
```
# setup USB HID mouse and keyboard
mouse = Mouse(usb_hid.devices)
keyboard = Keyboard(usb_hid.devices)
layout = KeyboardLayoutUS(keyboard)

# wii nunchuk setup
wiichuk = Nunchuk(i2c)

# Instantiate the BNo055 sensor
sensor = adafruit_bno055.BNO055_I2C(i2c)
sensor.mode = 0x0C  # Set the sensor to NDOF_MODE

**beep()** and **printd()** Functions

You'll set up functions for making the piezo beep and for printing debug information based on the **DEBUG** flag.

```python
# beep function
def beep(freq=440, duration=0.2):
    """Play the piezo element for duration (sec) at freq (Hz). This is a blocking method."""
    tone(board.D10, freq, duration)

# debug print function
def printd(line):
    """Prints a string if DEBUG is True."""
    if DEBUG:
        print(line)
```

**euclidean_distance()** Function

This function calculates the Euclidean distance between two sets of data. It's used to detect single taps on the BNO055 sensor.

```python
# euclidean distance function
def euclidean_distance(reference, measured):
    """Calculate the Euclidean distance between reference and measured points in a universe. The point position tuples can be colors, compass, accelerometer, absolute position, or almost any other multiple value data set.
reference: A tuple or list of reference point position values.
measured: A tuple or list of measured point position values."""
    # Create list of deltas using list comprehension
    deltas = [(reference[idx] - count) for idx, count in enumerate(measured)]
    # Resolve squared deltas to a Euclidean difference and return the result
    return math.sqrt(sum([d ** 2 for d in deltas]))
```
9-DOF Sensor Offsets

This is where you enter your specific calibration offset values based on running the calibration described above.

```python
# BNO055 offsets
# Preset the sensor calibration offsets
# User sets this up once for geographic location using `bno055_calibrator.py` in library examples
sensor.offsets_magnetometer = (198, 238, 465)
sensor.offsets_gyroscope = (-2, 0, -1)
sensor.offsets_accelerometer = (-28, -5, -29)
print(f"offsets_magnetometer set to: {sensor.offsets_magnetometer}"
print(f"offsets_gyroscope set to: {sensor.offsets_gyroscope}"
print(f"offsets_accelerometer set to: {sensor.offsets_accelerometer}"
```

Controller States

The variables maintain states for the Wiichuck, including roll and pitch states, button states, and a sensor packet count for the BNO055 to limit the ratio of sensor data between the 9-DOF and the Wiichuck.

```python
wii_roll_state = 1  # roll left 0, center 1, roll right 2
wii_pitch_state = 1  # pitch down 0, center 1, pitch up 2
wii_last_roll_state = 1
wii_last_pitch_state = 1
f_button_state = False
z_button_state = False
sensor_packet_count = 0  # Initialize the BNO055 packet counter
```

Beeps and Offsets

At the end of setup the piezo beeps, this is when you can point the nozzle at your screen to record a target angle offset. This is something you can re-run later with a key combo, but it is essential to keeping the relative offset of the sensor when compared to magnetic north pointed at the screen.

```python
print("PowerWash controller ready, point at center of screen for initial offset:")
beep(400, 0.1)
beep(440, 0.2)
time.sleep(3)
# The target angle offset used to reorient the wand to point at the display
#pylint:disable=(unnecessary-comprehension)
target_angle_offset = [angle for angle in sensor.euler]
beep(220, 0.4)
print("......reoriented", target_angle_offset
```
Main Loop

In the main loop, the code does the following:

- Reads Euler angle data from the BNO055 sensor
- Calculates cursor movement based on the orientation of the sensor
- Reads inputs from the Wiichuck joystick and accelerometer
- Processes button presses and changes in sensor orientation
- Detects single taps on the BNO055 sensor and triggers actions

```python
# BN055
# Get the Euler angle values from the sensor
# The Euler angle limits are: +180 to -180 pitch, +360 to -360 heading, +90 to -90 roll
sensor_euler = sensor.euler
sensor_packet_count += 1  # Increment the BNO055 packet counter
# Adjust the Euler angle values with the target_position_offset
heading, roll, pitch = [position - target_angle_offset[idx] for idx, position in enumerate(sensor_euler)]
printd(f"heading {heading}, roll {roll}"
# Scale the heading for horizontal movement range
# horizontal_mov = map_range(heading, 220, 260, -30.0, 30.0)
horizontal_mov = int(map_range(heading, -16, 16, HORIZONTAL_RATE*-1, HORIZONTAL_RATE))
printd(f"mouse x: {horizontal_mov}"
# Scale the roll for vertical movement range
t = int(map_range(roll, 9, -9, VERTICAL_RATE*-1, VERTICAL_RATE))
printd(f"mouse y: {vertical_mov}"
if CURSOR:
    mouse.move(x=horizontal_mov)
    mouse.move(y=vertical_mov)

# ================================================================
# sensor packet ratio
# Read the wiichuck every "n" times the BNO055 is read
if sensor_packet_count >= SENSOR_PACKET_FACTOR:
    sensor_packet_count = 0  # Reset the BNO055 packet counter

# ==============================================================
# wiichuck joystick
joy_x, joy_y = wiichuk.joystick
printd(f"joystick = {wiichuk.joystick}"
if joy_x < 25:
    keyboard.press(Keycode.A)
else:
    keyboard.release(Keycode.A)
if joy_x > 225:
    keyboard.press(Keycode.D)
else:
    keyboard.release(Keycode.D)
if joy_y > 225:
    keyboard.press(Keycode.W)
else:
    keyboard.release(Keycode.W)
if joy_y < 25:
    ```
keyboard.press(Keycode.S)
else:
    keyboard.release(Keycode.S)

# =======================================================================================
# wiichuck accel
wii_roll, wii_pitch, wii_az = wiichuk.acceleration
printd(f"roll:, {wii_roll}, pitch:, {wii_pitch}"
if wii_roll <= WII_ROLL_LEFT:
    wii_roll_state = 0
    if wii_last_roll_state != 0:
        keyboard.press(Keycode.SPACE)  # jump
        wii_last_roll_state = 0
elif WII_ROLL_LEFT < wii_roll < WII_ROLL_RIGHT:  # centered
    wii_roll_state = 1
    if wii_last_roll_state != 1:
        keyboard.release(Keycode.LEFT_CONTROL)
        keyboard.release(Keycode.SPACE)
        wii_last_roll_state = 1
else:
    wii_roll_state = 2
    if wii_last_roll_state != 2:
        keyboard.release(Keycode.LEFT_CONTROL)  # change stance
        wii_last_roll_state = 2

if wii_pitch <= WII_PITCH_UP:  # up used as modifier
    wii_pitch_state = 0
    if wii_last_pitch_state != 0:
        beep(freq=660)
        wii_last_pitch_state = 0
elif WII_PITCH_UP < wii_pitch < WII_PITCH_DOWN:  # level
    wii_pitch_state = 1
    if wii_last_pitch_state != 1:
        wii_last_pitch_state = 1
else:
    wii_pitch_state = 2  # down sends command and is modifier
    if wii_last_pitch_state != 2:
        keyboard.send(Keycode.TAB)
        beep(freq=110)
        wii_last_pitch_state = 2

# =======================================================================================
# wiichuck buttons
if wii_pitch_state == 0:  # button use when wiichuck is held level
    if wiichuk.buttons.C and c_button_state is False:
        target_angle_offset = [angle for angle in sensor_euler]
        beep()
        beep()
        c_button_state = True
    if not wiichuk.buttons.C and c_button_state is True:
        c_button_state = False
elif wii_pitch_state == 1:  # level
    if wiichuk.buttons.C and c_button_state is False:
        keyboard.press(WII_C_KEY_1)
        c_button_state = True
    if not wiichuk.buttons.C and c_button_state is True:
        keyboard.release(WII_C_KEY_1)
        c_button_state = False
elif wii_pitch_state == 2:  # down
    if wiichuk.buttons.C and c_button_state is False:
        keyboard.press(WII_C_KEY_2)
        c_button_state = True
    if not wiichuk.buttons.C and c_button_state is True:
        keyboard.release(WII_C_KEY_2)
        c_button_state = False

if wiichuk.buttons.Z and z_button_state is False:
    keyboard.press(Keycode.Space)
    keyboard.release(Keycode.Space)
    z_button_state = True
    keyboard.send(Keycode.TAB)
    beep(freq=110)
    z_button_state = False

else:
    keyboard.release(Keycode.Space)
mouse.press(Mouse.LEFT_BUTTON)
z_button_state = True
if not wiichuk.buttons.Z and z_button_state is True:
    mouse.release(Mouse.LEFT_BUTTON)
z_button_state = False

# ===========================================
# BNO055 tap detection
# Detect a single tap on any axis of the BNO055 accelerometer
accel_sample_1 = sensor.acceleration  # Read one sample
accel_sample_2 = sensor.acceleration  # Read the next sample
if euclidean_distance(accel_sample_1, accel_sample_2) >= TAP_THRESHOLD:
    # The difference between two consecutive samples exceeded the threshold ()
    # (equivalent to a high-pass filter)
    mouse.move(wheel=1)
    printd("SINGLE tap detected")
    beep()
    time.sleep(TAP_DEBOUNCE)  # Debounce delay

Build the Controller

For that authentic pressure washer feel, build the circuit into an old pressure washer nozzle. (You could alternately use an old squirt gun, Nerf blaster, or etc.)
Dissassembly
Unscrew the screws holding the halves of the nozzle together and remove the functional parts.

Fit the Parts
Dry fit the parts to make sure everything will fit before using adhesive foam tape to secure them in place.

The QT Py is mounted in the handle with the USB cable coming out of the base where the water line would normally go.

The 9-DOF sensor is mounted at the base of the barrel, with the Wiichuck adapter and piezo driver and piezo element in the central housing.
Wire Strain Relief
Use a zip tie to connect the Wiichuck cable to the USB cable inside of the handle as shown.

Piezo Driver Mount
Secure the piezo driver board using a piece of double-stick foam tape.
Piezo Element Mount
Remove the protective backing from the small piezo element's adhesive and press it to the inside of the forward handle guard.

Wiichuck Adapter Mount
Connect the Wiichuck cable to the interior of the main housing using double-stick foam tape.
BNO055 Mount
Mount the 9-DOF sensor oriented as shown -- here some foam was used to surround and secure it. You could also use double-stick foam tape instead.

Fasten the Nozzle Halves
Finally, close up the nozzle and screw the halves together -- be sure no wires are in the way before tightening it up.
You're ready to play!

Blast Away Some Grime!

Plug in the USB cable to your PC and fire up the game! These are the inputs and actions you'll control:

- nozzle heading/roll (sensor is mounted "sideways" in washer handle) = mouse x/y
- nozzle tap/shake = next nozzle tip
- wii C button (while level) = rotate nozzle tip
- wii Z button = trigger water
- wii joystick = WASD
- wii roll right = change stance stand/crouch/prone
- wii roll left = jump
- wii pitch up + C button = set target angle offset
- wii pitch down = show dirt
- wii pitch down + C button = toggle aim mode

This demo video shows it in action: