In [42]:
```
from time import sleep
import serial
import random
```

Connect to the Arduino

This Arduino (Arduino Nano) always connects as '/dev/ttyUSB#'

In [27]:
```
ser = None
connected = False# Connect to the Arduino
This Arduino (Metro Mini) always connects as '/dev/ttyUSB#'
for port in range(8):
    try:
        ser = serial.Serial('/dev/ttyUSB%d' % port, 115200) # Establ
ish the connection on a specific port
        connected = True
        print("Connected to device at /dev/ttyUSB%d" % port)
        break
    except:
        continue
if not connected:
    print("Failed to connect")
```

```
Connected to device at /dev/ttyUSB0
```

Controlling the Vibration Motors

These are several high-level functions that can be used to send commands to the Arduino and control the vibration motors.

The Arduino code has support for analog motor control (0-99 values of power, 0 being fully off and 99 being on at full power), but the motors used in the test setup are digital, i.e. they only have two states, off and on at full power. Therefore, two functions, *set_vibration_analog* and *set_vibration_digital* are provided to control different types of vibration motors, but the Arduino code should be able to handle both.

stop_vibration is provided for convenience and just stops all vibration motors.

Several high-level vibration sequences are provided through the *sequence_vibration* function, including:

- **"pulseall"** - pulses all vibration motors simultaneously
- **"train"** - does a sweep over all the vibration motors in order
- **"random"** - produce random vibes, like a "hum" at shorter delays
- **"accel"** - accelerate pulses from a slow rate to faster rate
- **"decel"** - decelerate pulses from a fast rate to slower rate

In order to initiate a sequence, call *sequence_vibration* with the name of the sequence (*seq*), delay between steps in the sequence (*delay*), and repetitions of the sequence (*reps*).

Alternatively, you can call the overloaded functions *play_{seq}* to play the corresponding sequence with given *delay* and *reps* parameters.

In [64]:
```
def set_vibration_analog(v0=0, v1=0, v2=0, v3=0, v4=0):
    v0 = int(min(max(0, v0), 99))
    v1 = int(min(max(0, v1), 99))
    v2 = int(min(max(0, v2), 99))
    v3 = int(min(max(0, v3), 99))
    v4 = int(min(max(0, v4), 99))
    ser.write(b"V0%02d_V1%02d_V2%02d_V3%02d_V4%02d\n" \
              % (v0, v1, v2, v3, v4))

def set_vibration_digital(v0_bool, v1_bool, v2_bool, v3_bool, v4_boo
l):
    v0 = 99 if v0_bool else 0
    v1 = 99 if v1_bool else 0
    v2 = 99 if v2_bool else 0
    v3 = 99 if v3_bool else 0
    v4 = 99 if v4_bool else 0
    ser.write(b"V0%02d_V1%02d_V2%02d_V3%02d_V4%02d\n" \
              % (v0, v1, v2, v3, v4))

def stop_vibration():
    set_vibration_digital(0, 0, 0, 0, 0)

def sequence_vibration(seq, delay=0.1, reps=10):
    if seq == "pulseall":
        for i in range(reps):
            stop_vibration()
            sleep(delay)
            set_vibration_digital(1, 1, 1, 1, 1)
            sleep(delay)
        stop_vibration()

    if seq == "train":
        for i in range(reps):
            set_vibration_digital(1, 0, 0, 0, 0)
            sleep(delay)
            set_vibration_digital(1, 1, 0, 0, 0)
            sleep(delay)
            set_vibration_digital(0, 1, 1, 0, 0)
            sleep(delay)
            set_vibration_digital(0, 0, 1, 1, 0)
            sleep(delay)
            set_vibration_digital(0, 0, 0, 1, 1)
            sleep(delay)
            set_vibration_digital(0, 0, 0, 0, 1)
            sleep(delay)
        stop_vibration()

    if seq == "random":
        for i in range(reps):
            set_vibration_analog(
                v0=int(100*random.random()),
                v1=int(100*random.random()),
                v2=int(100*random.random()),
                v3=int(100*random.random()),
                v4=int(100*random.random()))
            sleep(delay)
```

```
        stop_vibration()

    if seq == "accel":
        for i in range(reps):
            stop_vibration()
            sleep((reps-i)*delay)
            set_vibration_digital(1, 1, 1, 1, 1)
            sleep((reps-i)*delay)
        sleep(delay)
        stop_vibration()

    if seq == "decel":
        for i in range(reps):
            stop_vibration()
            sleep(i*delay)
            set_vibration_digital(1, 1, 1, 1, 1)
            sleep(i*delay)
        sleep(delay)
        stop_vibration()

def play_pulseall(delay, reps):
    sequence_vibration("pulseall", delay=delay, reps=reps)

def play_train(delay, reps):
    sequence_vibration("train", delay=delay, reps=reps)

def play_random(delay, reps):
    sequence_vibration("random", delay=delay, reps=reps)

def play_accel(delay, reps):
    sequence_vibration("accel", delay=delay, reps=reps)

def play_decel(delay, reps):
    sequence_vibration("decel", delay=delay, reps=reps)
```

In [61]:
```
play_pulseall(0.1, 10)
```

In [62]:
```
play_train(0.1, 10)
```

In [55]:
```
play_random(0.05, 10)
```

In [65]:
```
play_accel(0.01, 25)
```

In [66]:
```
play_decel(0.01, 25)
```

Sample Usage

This is a run-through of an example session of commands that might be sent to the Arduino to control the vibration motors.

It starts out with a pulse-train of vibrations at a moderate tempo, then accelerates to a much faster pace. The vibration motors then all pulse simultaneously at this quick pace for a while, and eventually mix it up with random pulses for another period. Finally, the vibration decelerates and eventually stops.

In [68]:
```
# Sample Usage
play_train(0.1, 20)  # do 20 sweeps
play_accel(0.01, 20)  # accel pulses from 0.2s delay to 0.01s
play_pulseall(0.04, 100)  # pulse all at fixed delay for 100 reps
play_random(0.04, 100)  # random pulses for 100 more reps
play_decel(0.01, 20)  # decel from 0.01s delay to 0.2s
stop_vibration()
```