# EMERGING TECHNOLOGIES I

MIS 284N | Fall 2018

## Project Milestone 2

**Overview**: For our second milestone, we'll (1) set up a Raspberry Pi device and (2) connect the microbit step counter to a python program on the Raspberry Pi. We'll use both a serial connection via USB and a Bluetooth connection to get data from the microbit to the Raspberry Pi. The Raspberry Pi will serve a secondary purpose: it will actually be a WiFi access point that the Android device can eventually connect to. You will also use this WiFi connection to run VNC from your laptop computer to program the Raspberry Pi without any need for peripherals (e.g., keyboard, mouse, monitor) for the Raspberry Pi.

**Step 0**: Get a Raspberry Pi, download the VNC viewer, and follow the instructions given in class.

**Step 1**: Partner with another group and write a microbit program to send a counter back and forth (or around a circle. Be sure to resolve the problem of identifying your counter from another group's counter. Can you do this with the *same* code running on the two microbits? With *different* code? Do you prefer to write this in python or in javascript or in makecode?

**Step 2**: Write a microbit program to send some dummy data from the microbit to the Raspberry Pi using a serial connection across USB. On the Raspberry Pi side, start by following the Linux instructions provided by microbit (https://makecode.microbit.org/device/serial). You can extend this to Python using the following template:

```
import serial, time
port = “/dev/ttyACM1”
baud = 115200
while true:
    s = serial.Serial(port)
    s.baudrate = baud
    data = s.readline()
    data = int(data[0:4])
    print(data)
    time.sleep(1)
```

Extend your Milestone 1 solution so that you send your step count via USB upon a button press, and it is then printed to the terminal on the Raspberry Pi.

**Step 3**: With the combined group again, create a data logger. This data logger should send accelerometer data from one microbit, over the radio module to another microbit, then over the serial connection to the Raspberry Pi, then be printed to the terminal on the Raspberry Pi.

**Step 4**: You no longer need the larger group. Everything from here on out will be done with just your project partner. In this step, get a simple bluetooth connection working from the microbit to the Raspberry Pi. The Raspberry Pi already has the bluezero python library installed (https://media.readthedocs.org/pdf/bluezero/latest/bluezero.pdf). Check out the section in the documentation that describes the integration of the microbit. Send some data from the microbit to the Raspberry Pi. I used a tutorial I found online (https://ukbaz.github.io/howto/ubit_workshop.html). I did this without pairing my microbit and Raspberry Pi. I thought that the second python example was the most fun (the one that says "Reading a button press"). First use the provided .hex file. Then try to recreate it on your own. Note: you have to use makecode; bluetooth is not supported in the python for the microbit.

**Step 5**: Send an Eddystone beacon from the microbit to the Raspberry Pi. This will be easy on the microbit side but was (for us) non-trivial on the Raspberry Pi python side. You will need to use the aioblescan library directly instead of attempting this through bluezero (at least I couldn't get the bluezero observer to work; you're welcome to try... there's an example in the bluezero documentation).

Step 5a: Create the microbit program to send an Eddystone beacon. The invoke the following command in the terminal on the Raspberry Pi to ensure that the Raspberry Pi is receiving the Eddystone beacon:

```
sudo python3 -m aioblescan -e
```

Step 5b: Write the python code to run on the Raspberry Pi. Here's some code you can start with:

```
import aioblescan as aiobs
from aioblescan.plugins import EddyStone
import asyncio

def _process_packet(data):
    ev = aiobs.HCI_Event()
    xx = ev.decode(data)
    xx = EddyStone().decode(ev)
    if xx:
        print("Google beacon: {}".format(xx))

if __name__ == '__main__':
    mydev = 0
    event_loop = asyncio.get_event_loop()
    mysocket = aiobs.create_bt_socket(mydev)
    fac = event_loop._create_connection_transport(mysocket,aiobs.BLEScanRequester,None,None)
    conn, btctrl = event_loop.run_until_complete(fac)
    btctrl.process = _process_packet
    btctrl.send_scan_request()
    try:
        event_loop.run_forever()
    except KeyboardInterrupt:
        print('keyboard interrupt')
    finally:
        print('closing event loop')
        btctrl.stop_scan_request()
        conn.close()
        event_loop.close()
```

Note: when you send an Eddystone beacon, you have to send a well-formatted URL, but you can send sensor values or other data as part of that URL. For instance, you could send a URL that looks like "http://steps?12345."

**Step 6**: Create a makecode program that (a) reliably counts steps and gives the user feedback on the microbit about his or her step count (this is just Milestone 1, in makecode); (b) sends the step count to a Raspberry Pi program when instructed to (e.g., by a button press); and (c) prints any received step count to the terminal. Some requirements and hints:

- The microbit should *not* continuously beacon. You need to decide when to start and stop beaconing.
- You want to create a URL that is somewhat unique to your program (that is, you don't want your Raspberry Pi program to read and parse *everyone's* step counts. Just yours.
- You can create interesting string variables on the microbit. You'll want to check out the Variables section of the makecode page and the Text section.

## *What to submit*

Via Canvas, submit a brief writeup that includes the following information for each step:

- **Step 1**: Answer the questions embedded in Step 1 (yes, some of them are yes/no questions). How did you solve the problem of conflicts with other groups' counters?
- **Step 2**: In your own words, what is *baud rate*?
- **Step 3**: Provide the code that your (combined) group used for the microbit that was directly connected to the Raspberry Pi.
- **Step 4**: Include a screenshot of the makecode solution for the microbit.
- **Step 5**: In your own words, explain what you think is happening in the provided code. You do not need to go line by line, but you should be able to explain the overall flow of activity. The documentation for both asynchio (https://docs.python.org/3/library/asyncio.html) and for aioblescan (https://github.com/frawau/aioblescan) may be helpful. Note that we're using the `_create_connection_transport` function in the asynchio `event_loop`. It's not part of the public interface. It's a "fix" I stole from the aioblescan source code. For the purposes of this response, you can assume it behaves basically like the `event_loop`'s `create_connection` function.
- **Step 6**: How did you solve the problem (not more than one paragraph)? Include a screenshot of the microbit program and a listing of the python receiver.

Wednesday, November 7, in office hours, demonstrate Step 6. Be prepared to explain what the code does on each side (both the makecode or javascript on the microbit and the python on the Raspberry Pi).