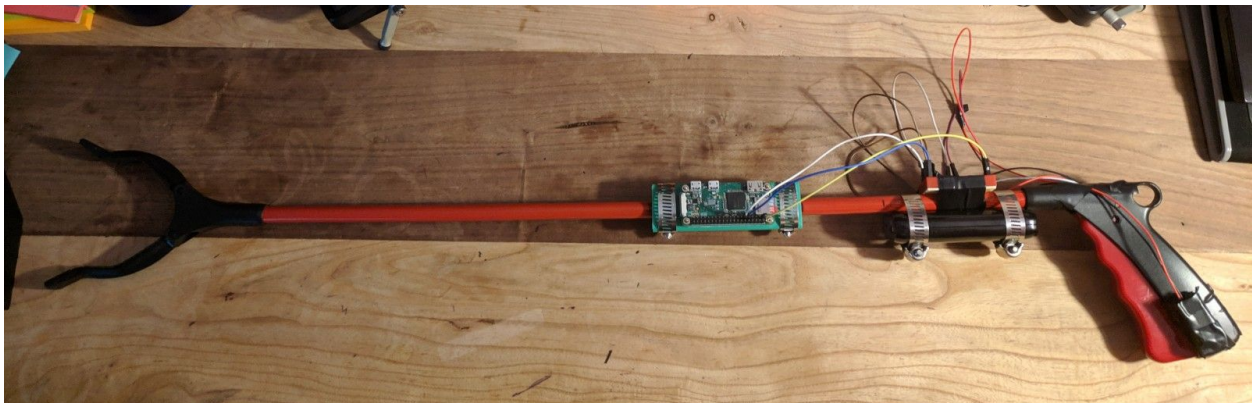
Final Project Written Reflection: Smart Trash Picker



Michael Rechenberg
(rchnbrg2@illinois.edu)
INFO 490: Intro to Makerspace

What I Made and What I'm Proud of (Question 1)

My final project was to create a "smart" trash picker (STP for short). A STP would detect when someone has picked up trash and coordinate with the user's smartphone to record the current user's location. I go over some of the possible benefits from collecting location while collecting garbage in my final presentation: https://drive.google.com/open?id=1Io211Lj2O5uOXhJjm8Kk9mmDFYy1v-sO8MCkPxU8lgQ

Essentially, I took a "dumb" $3 trash picker/reach tool and attached a Pi Zero W + IR break-beam sensor so the STP can detect when the user picked up trash and communicate with the user's smartphone (over Bluetooth Low Energy). The IR break-beam emitter and collector would be placed on opposite sides of the handle and with a hole drilled in the handle, the emitter and collector would have an unbroken beam. When the user closes the handle (to pick up trash), we could detect that because the IR beam would now be broken.

"Dumb" Trash Picker

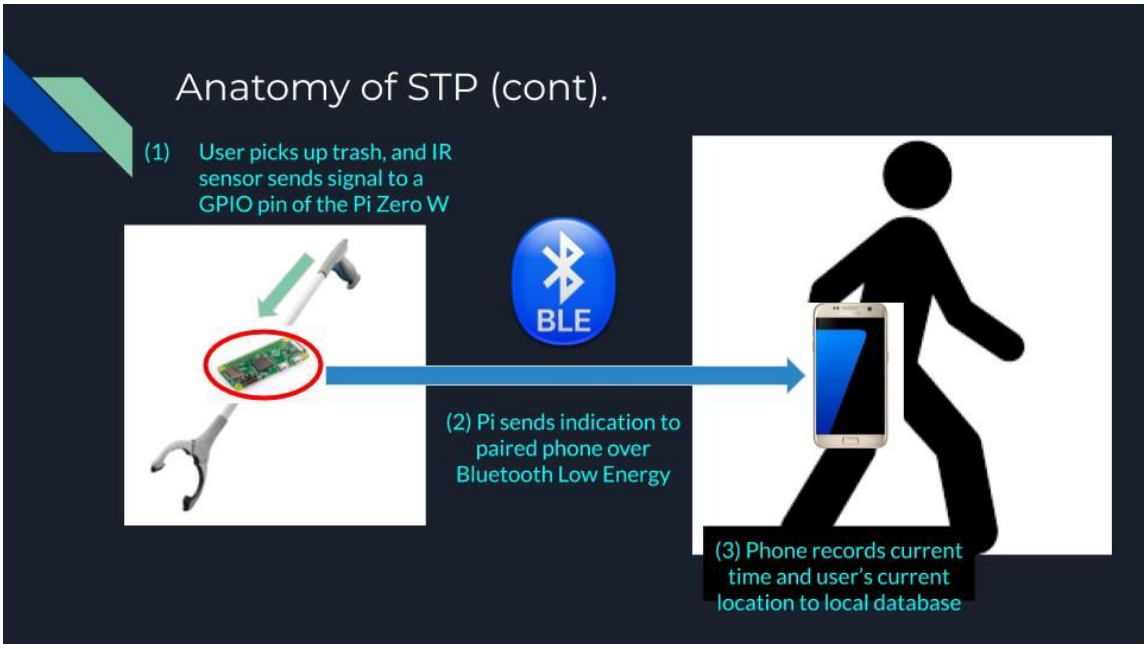
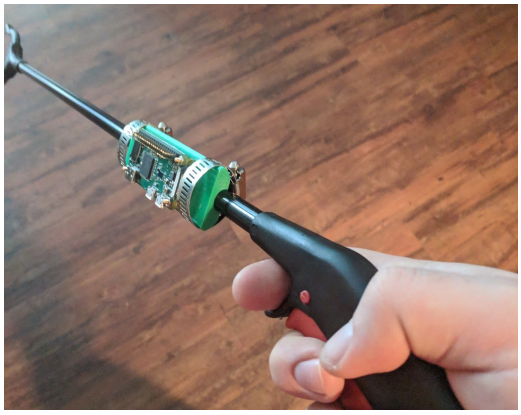


Diagram of how the STP works

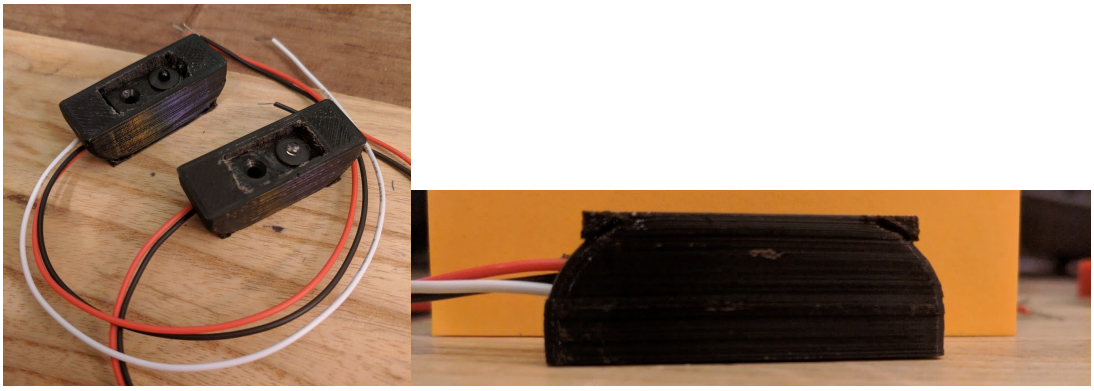
3D Printing and Circuits

I used two 3D printed parts for my final project. I made the first part for the 3D Printing Assignment and re-used it for the final project http://cucfablab.org/3d-printing-assignment-week-1/: a mounting that I could affix to the rod of the trash picker using hose clamps and the Pi to using screws.



The second 3D printed-part, one that I designed and printed during the final project weeks, was a pair of housings for the IR sensors. I learned from the 3D printing assignment to add some tolerances to my model and I did add some small tolerances in my model, but I didn't leave enough tolerances so I had to further file down the housings a bit for the IR sensors to fit.


The IR sensors from the Adafruit Store



The IR sensors placed within my 3D printed housing

After all the 3D printed parts were made, I wired up a circuit such that the IR sensors would be powered by the 3.3V pin of the Pi and the signal wire from the IR collector would go into a GPIO pin of the Pi, so the Pi could detect when the user closed the handle.

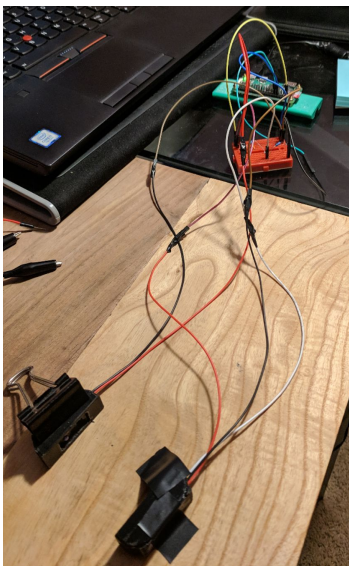
Programming

I programmed 2 main components for this project. The first unit of code was for the Pi, where I made it a Bluetooth GATT server that would send an indication to the smartphone when the user closed the handle of the STP (i.e. they picked up a piece of trash). My code for the Pi is on Github, and you can view it here: https://github.com/MichaelRechenberg/Smart-Trash-Picker. **Understanding the Bluetooth protocol and installing the required libraries on the Pi was the most challenging part of my whole project**. I had to read through many forum posts, Medium articles, and StackOverflow posts to understand and set everything up.

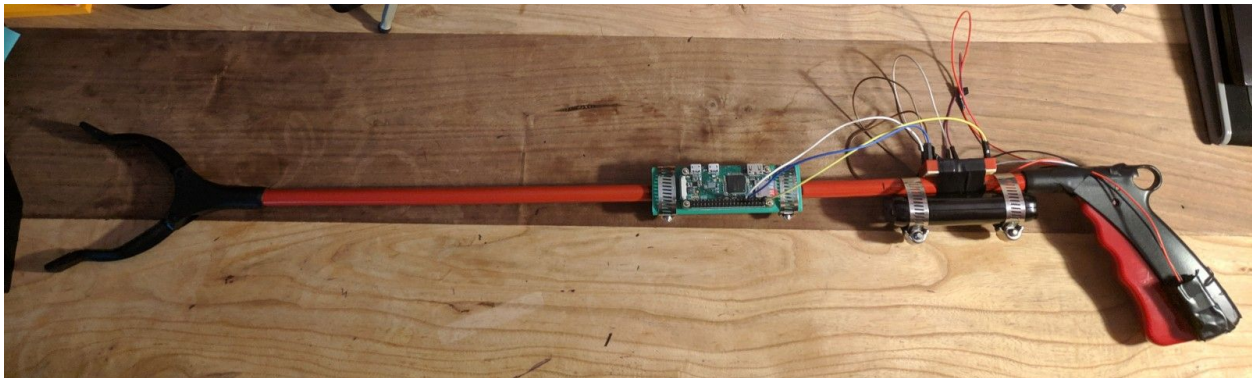
The second unit of code was an Android app for the user's smartphone. The app scans Bluetooth devices for available STPs that the user can pair with. Once paired with an STP, the app will record the current time and user's current location to an on-phone database once the

app receives an indication from the STP. When the user gets back home, they can export their location data to a server for later analysis. My code for this app is on Github, and you can view it here (or even run it yourself if you have and Android device and Android Studio): https://github.com/MichaelRechenberg/SmartTrashPickerAndroidApp

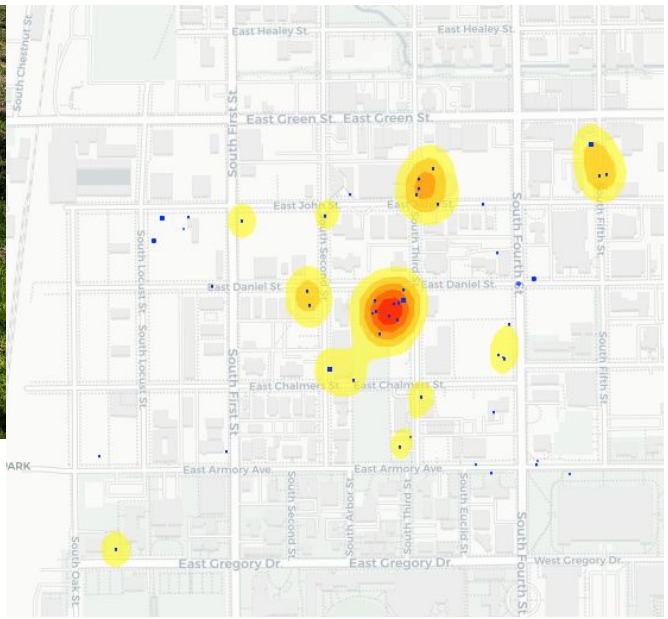
Build Photos



(Left) Building the circuit to connect the IR sensors and the Pi together
(Right) Attaching the IR sensors to the handle with electrical tape so the IR beam shoots through the drilled hole in the handle.



Some pictures of the fully wired-up STP. I decided to include a cardboard shield around the Pi to protect it from sunlight/dust/etc. while I was using the STP.



On May 6th, I used the STP for a few hours
between 1st and 6th Street. When I got back home, I exported my the location data I collected and generated a heatmap of where I picked up trash. As you can see on the heatmap, I picked up the most trash in Washington Park (where the main red spot is).

I am most proud of generating that heatmap, since it is the culmination of my whole project. The goal of the STP is to help collect location data when picking up trash, and visualizing the trash problem with a heatmap is one of the possible uses of the data the STP collects. It serves as a proof-of-concept for the whole project.

I'm also proud that I was able to make a fully functional "physical computing" device. Before taking this class an Internet of Things (CS 498), I would have been hesitant to step out of my pure software comfort zone. But the STP broadened my programming abilities to be more physical (circuits triggered by physical movement, a fully portable product) and to use a new protocol (Bluetooth Low Energy).

Learning Goals (Question 2)

I had 2 learning goals for my final project:

- Learn a new technology (Bluetooth Low Energy, BLE)
- Make a prototype and guide that others could build and deploy, or learn more about BLE

I wanted to learn how to use BLE since it was a protocol we discussed briefly in my IoT class but we never got to work hands-on with. BLE is a popular protocol for IoT and since I want to dabble in that field in my spare time, learning how to use BLE was a valuable skill to pick up. I started off knowing nothing about the nuts and bolts of the protocol, but after a couple of days of reading documentation I was able to understand the concepts of BLE (e.g. what a Service, Characteristic, and Descriptor is; how GATT profiles work; how advertising works). Then I was able to program a functioning product that uses both sides of a BLE connection (the peripheral device and the central device). I consider the creation of a working BLE product as an indicator that I successfully learned BLE and completed my first learning goal.

I completed my second learning goal by publishing a list of all the materials I used and provided all of the code I wrote for free on Github. This is in the Appendix of this document so anyone who wanted to make an STP (or just learn how to work with BLE) could do so. In writing the guide, I solidified my own understanding of BLE and my project. As Genesis singer Phil Collins said, "In learning you will teach, and in teaching you will learn."

But more importantly, the guide makes it easier for others to make an STP and possibly deploy them themselves: the data-collection aspect of the STP is most effective when there are more STPs deployed, and a detailed construction guide catalyzes the construction of more STPs. More data means a more accurate view of the city.

Looking back, I'm surprised how much the final project benefitted from consulting various people on the design. My roommate is a Mechanical Engineering student, so I asked him how best to mount a box (i.e. Pi) onto a cylinder (i.e. trash picker's rod). Our discussion led to the design of the Pi mount I ended up printing, with a cylindrical cavity for the rod to sit in and hose clamps to secure the mounting. TAs in my IoT class also helped brainstorm some ideas for what sensors to use to detect when the user picked up a piece of trash (button within the handle, contact sensors attached to the cable within the picker). I didn't end up using any of their suggestions, but that discussion prompted me to expand my search horizon, which led to me poking around the Adafruit website for various sensors and inspired my design of using IR breakbeam sensors.

What Have I Learned From Lab Assignments (Question 3)

The most significant thing I have learned throughout this semester is that a design you make on a computer will not be as clean and precise when it is fabricated out of physical material. This was most evident with my 3D printing, where I learned the hard way that a 10mm cylinder may fit into a 10mm hole in Fusion 360, but printing tolerances mean that those two parts won't fit as nicely once printed; I had to file down some portion of my prints to get the parts I wanted to fit together. This principle extends beyond 3D printing: in the sewing unit, my first embroidery attempt was poor because the type of stitch looked decent on the computer, but once the stitch was manifested in thread the shape was so thin that it was difficult to see in the fabric.

I feel comfortable using the various software tools we've used throughout the semester to make my designs. I understand *how* to create my designs on the computer, but I think I need more experience making physical objects with those software tools before I internalize physical tolerances effectively (i.e. no longer have to file down my 3D print) into my designs.

Future Goals/Interests and Making (Question 4)

To me, a good maker is someone who not only has technical aptitude in their craft(s), but also teaches their skills and techniques to others. I would not have been able to make my STP without help from staff in the Fab Lab or from authors of online tutorials and articles. By sharing their knowledge and resources, I was able to take my idea and turn it into a reality. I want to do the same with my STP tutorial/guide: I hope that anyone who wants to make an STP or just learn about working with BLE on another project can use my work to propel their own work. I enjoy teaching others and I feel an obligation to pay back a debt of aid to the community.

I consider myself a beginner maker who specializes in working with computing devices (e.g. Arduino and Pis) and networking. Over the semester I've accumulated the basic materials or hobbyist electronics, and combined with my INFO 490 and CS 498 IoT knowledge I'd like to continue making electronics-focused things in my spare time. Given my Computer Science background, I am an experienced programmer and I can probably pick up new programming environments than the average person. I'll be moving to Seattle after I graduate, and I want to be one of the "computer guy" mentors at one of the Makerspaces in the area. I'd like to

specialize in one computing paradigm (Seattle has more than one programmer in the area) or be the guy to try out new software/library X for the lab.

Appendix: STP Tutorial/Guide

One of my learning goals was to make my final project reproducible. So I've included a guide for anyone else who wanted to build and use their own STP or learn more about how to use BLE in their projects. I tried to include links to all materials and guides I used when building my own STP. I assume the user is somewhat familiar with the shell of a Linux environment and has some Python programming experience

Materials Needed

- Reach Tool/Trash Picker ($3) -> https://www.menards.com/main/tools-hardware/hand-tools/specialty-hand-tools/reach-tool/ad111014/p-1444423365274.htm
    - This picker has an outer-diameter of 0.475 inches. If you choose a different picker, you may have to adjust your 3D model so the cylindrical cavity fits the rod of the picker you're using
    - My design uses a picker with a handle that moves inside the handle. You may have to use a different method to detect when the user closes the handle if your picker's handle is different. Perhaps a sensitive enough Hall effect sensor paired with a magnet embedded in the handle?
- Raspberry Pi Zero WH (Zero W with Headers) ($14) -> https://www.adafruit.com/product/3708?gclid=EAIaIQobChMI3v2ulvCJ4gIVTdbACh15XwPBEAYYAiABEgJcr_D_BwE
    - Note that you could buy the Zero W instead ($10), but for convenience I bought a Pi Zero with headers already installed to make building the circuit easier
- Screws for Pi (to screw the Pi into its mounting)
    - I had a spare kit for a Pi 3B+ lying around, but the Pi Zero W has the same screws (M2.5)
- A portable power bank **and Micro USB cable** to power the Pi Zero W
    - I used a 2A Anker power bank for my STP because I had it around to charge my phone ($20) https://www.amazon.com/gp/product/B00P7N0320/ref=ppx_yo_dt_b_asin_title_o02_s01?ie=UTF8&psc=1
    - However, you could probably get away with a 1.2 A power bank (which might be cheaper and lighter as well)
- IR Breakbeam Sensors (3mm LEDs, 3.3V) -> https://www.adafruit.com/product/2167
- Hose Clamps
    - I used two 1 1/16 - 2 inch hose clamps to secure my power supply, and two 9/16 - 1 1/16 inch hose clamps to secure the Pi Zero mounting
- Micro SD Card with at least 8GB (for the Pi)

- Two 10K Ohm resistors
- Various female-to-male jumper wires
- Mini Breadboard -> https://www.amazon.com/DGZZI-Solderless-Prototype-Breadboard-Protoboard/dp/B07N41ZPC3/ref=asc_df_B07N41ZPC3/?tag=hyprod-20&linkCode=df0&hvadid=312400581241&hvpos=1o3&hvnetw=g&hvrand=7987298867202488962&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=9022196&hvtargid=pla-681712466097&psc=1
    - If you plan to solder parts together, then you don't need this
- 3D printer and plastic (probably at a Makerspace facility)

You can use my models for the Pi mounting and IR sensors if you want: https://drive.google.com/open?id=1H5B4G76glptIoTFYgsUFrt0W3_fURfMz, https://drive.google.com/open?id=1qpB3Azs1quGZY0LRpcRjVgJ3Zgkl8b5S. However as I discussed earlier in the report, I had to file down some of the prints and drill holes into the IR sensor casing...so perhaps see those models as one potential solution, but create your own model more suitable for mass production.

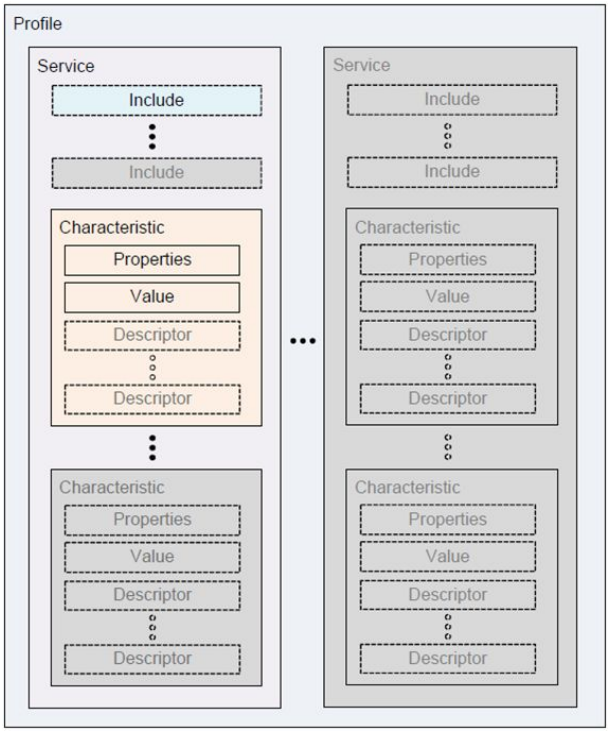
Optional Materials

To make it easier to initially setup the Pi Zero, I used a **mini (NOT micro)** HDMI adapter https://www.amazon.com/HDMI-Adapter-VCE-2-Pack-Plated/dp/B0111HRJT0/ref=sr_1_1?gclid=EAIaIQobChMI75KAo_aJ4gIVAp7ACh0xBQHIEAAYASAAEgJH-PD_BwE&hvadid=234297351084&hvdev=c&hvlocphy=9022196&hvnetw=g&hvpos=1t1&hvqmt=e&hvrand=5346005375344158450&hvtargid=kwd-24343202982&hydadcr=19107_10158132&keywords=mini-hdmi+adapter&qid=1557249454&s=electronics&sr=1-1 to connect the Pi to a monitor and a micro USB to USB hub https://www.amazon.com/LoveRPi-MicroUSB-Port-Black-Raspberry/dp/B01HYJLZH6/ref=sr_1_4?gclid=EAIaIQobChMIituVwPaJ4gIVEZ7ACh2YFgWkEAAYASAAEgJGUfD_BwE&hvadid=241918566923&hvdev=c&hvlocphy=9022196&hvnetw=g&hvpos=1t1&hvqmt=e&hvrand=16870267887497176262&hvtargid=kwd-38933259364&hydadcr=24634_10399774&keywords=micro+usb+to+usb+hub&qid=1557249514&s=gateway&sr=8-4 to connect a keyboard and mouse to. You can ignore these if you do a headless setup from the get-go

Bluetooth Low Energy (BLE) Primer

Before going further, you need to understand some of the key concepts for BLE and that BLE is not the same as regular Bluetooth. BLE is designed to transfer small pieces of data and conserve battery power. You should check out https://developer.android.com/guide/topics/connectivity/bluetooth-le.html and https://www.bluetooth.com/specifications/gatt/generic-attributes-overview/ first; I'll just cover the relevant parts of BLE for the STP here.

BLE uses Generic Attribute profiles (GATT) to specify how that device sends little chunks of data called "attributes" to interested devices. GATT is hierarchical: one profile can contain many Services, each Service can contain many Characteristics, and each Characteristic can contain 0 or more Descriptors. For example, a heart monitor implant could have a HeartRate Service that contains some Characteristics like HeartRateMonitoring and BodyLocation.



Graphic of GATT hierarchy

For the STP we have a lean GATT profile with the following hierarchy

- Service: Smart Trash Picker Service (0x1337)
    - Characteristic: Trash Grabbed Characteristic (0x1574)
        - Descriptor: Client Characteristic Configuration Descriptor (CCCD, 0x2902)

Services, Characteristics, and Descriptors each have an associated UUID. Read this article for more information about the structure of BLE UUIDs https://www.argenox.com/library/bluetooth-low-energy/ble-advertising-primer/#a-quick-look-into-uuids. Some UUIDs are baked into the BLE standard (such as the UUID for CCCD = 0x2902). The STP defines two new UUIDs for its Service (0x1337) and Characteristic (0x1574). These two UUIDs are arbitrary and you can make them whatever you want for your application (as long as they don't clash with a standard UUID).

CCCD allows clients to control how they receive notifications and indications. We use the CCCD so that when the Trash Grabbed Characteristic is modified (i.e. we detect the user picked up trash), we send an indication to the user's phone. You can read more about CCCD here https://devzone.nordicsemi.com/f/nordic-q-a/561/what-does-cccd-mean. Tldr; The Android App needs to write to this CCCD before it will receive indications from the Pi when trash is grabbed.

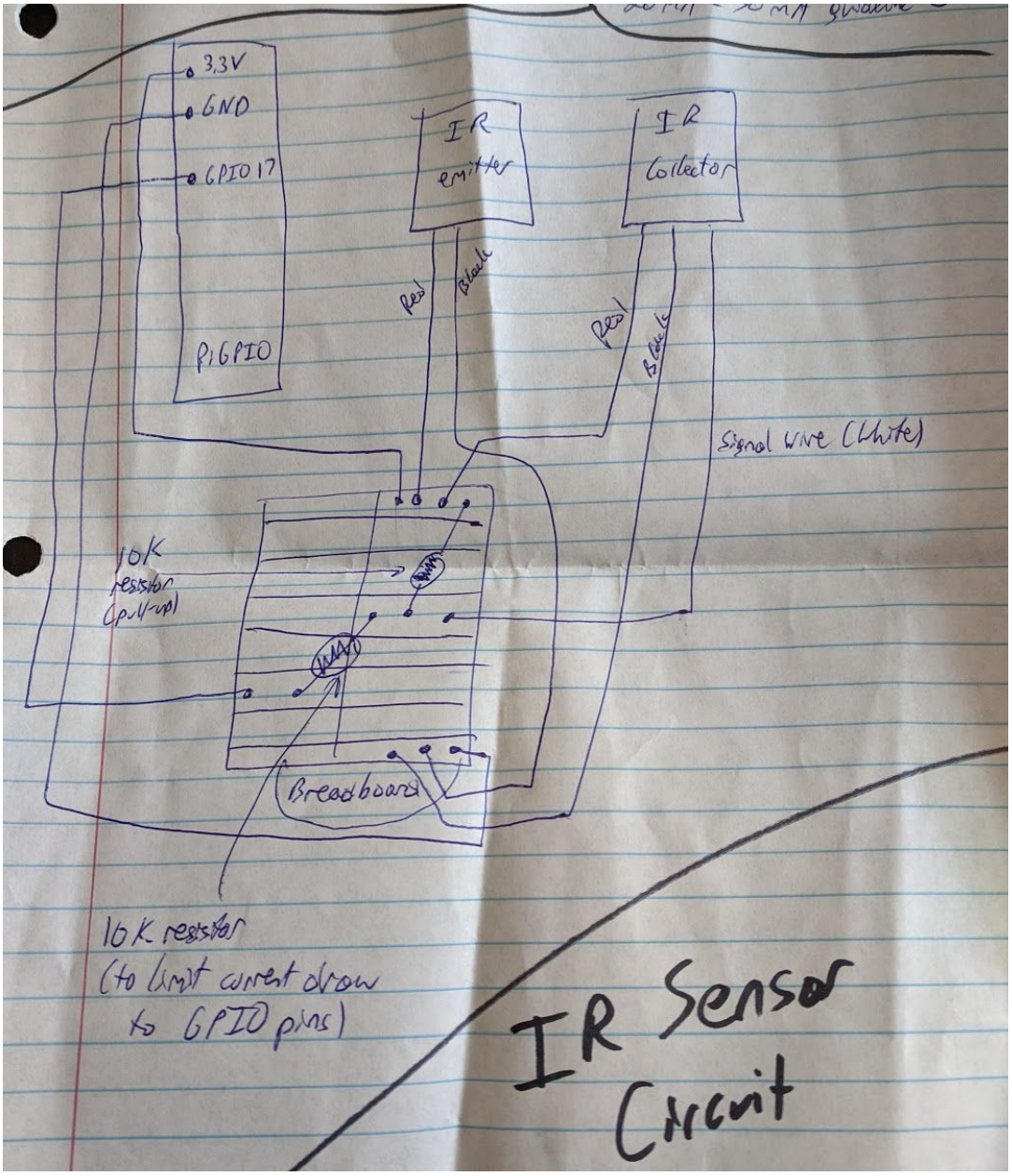
BLE devices take on one of two roles in two orthogonal dimensions: GATT server vs GATT client, and central vs peripheral device. A BLE network can only have one central device, but can have several peripheral devices connected simultaneously. For the STP, we have two BLE devices: the Pi Zero W and the smartphone. **The Pi Zero W acts as the GATT server and a peripheral device, whereas the smartphone acts as a GATT client and as the central device**.

Finally, the Pi has to broadcast BLE *advertisements* so other Bluetooth devices know what Services the Pi can provide. For the STP, the Pi broadcasts that it can perform the Smart Trash Picker Service (UIUD = 0x1337). The Android app will listen for Bluetooth advertisements and filter results based on that UIUD. That way, the app can discover the Pi while disregarding other Bluetooth devices in the area (e.g. wireless speakers, headphones).

Setting Up the Pi: IR Circuit and GATT Server

IR Circuit

First, you'll have to wire up the circuit connecting the IR sensors and the Pi by soldering or using a mini breadboard. Use the following circuit diagram:



Installing Dependencies

Before we can use the IR circuit and run the GATT server on the Pi, we need to install some system libraries (BlueZ) and Python libraries (pybluez) on the Pi Zero W. This was the most difficult part for me, so you might have to troubleshoot your installation as you go.

- First, we have to install some system packages using apt-get
    - `sudo apt-get update`
    - `sudo apt-get install bluetooth`

    - `sudo apt-get install bluez`
    - `sudo apt-get install python-bluez`
    - `sudo apt-get install libbluetooth-dev python-dev libglib2.0-dev libboost-python-dev libboost-thread-dev`
- And some other development libraries
    - https://stackoverflow.com/questions/23436909/where-is-the-bluetooth-bluetooth-h-located-in-linux
    - `sudo apt-get install libbluetooth-dev`
    - https://bitbucket.org/OscarAcena/pygattlib/issues/38/build-fails-on-boost-python-listhpp
    - `sudo apt-get install libboost-python-dev`
    - `Sudo apt-get install libdbus-glib-1-dev libgirepository1.0-dev`
- One dependency of pybluez, gattlib, was ornery when trying to install. I was able to install it by following the instructions given here (especially the sed line to make the boost version 3.5) to install from source-> https://github.com/pybluez/pybluez/wiki/Installation-on-Raspberry-Pi-3
    - Note, this might take a few minutes to complete
    - Note that you may have to change the swap size of Pi to a larger amount (1024MB) so the compiler doesn't complain about running out of memory when compiling from source https://www.bitpi.co/2015/02/11/how-to-change-raspberry-pis-swapfile-size-on-rasbian/, After pybluez is installed, SWAP IT BACK TO 100MB WHEN YOU'RE DONE TO SAVE YOUR SD CARD FROM UNNECESSARY WRITES
- We need BLE support, so I installed blueZ and BLE support via pip. You might want to grab some coffee since this build will take a couple of minutes
    - `pip install pybluez[\ble\]`
- Take a deep breath because BlueZ and its bindings are finally installed :)

Now install BerryConda on the Pi Zero W, following the instructions here https://github.com/jjhelmus/berryconda

Now you can clone my Git repository with the STP python code from here https://github.com/MichaelRechenberg/Smart-Trash-Picker. Inside the cloned directory, run the following commands:

- `conda env create -f environment.yml`
    - Create a Conda environment
- `source activate smart-trash-picker`
    - Activate that Conda environment
- `pip install python-dbus`
- `pip install -r requirements.txt`
    - Install the required python libraries

Running the GATT Server and Circuit Code

After all that installation, we can finally start running the code on the Pi. my-gatt-server.py is the script that runs the GATT server and sends out indications when the Pi detects a rising edge on GPIO pin 17 (the signal line of the IR collector).

Note if you want understand what's going on in my-gatt-server.py and write your own BLE application, you should read up on D-Bus https://www.freedesktop.org/wiki/Software/dbus/ and try to understand the example Service here https://scribles.net/creating-ble-gatt-server-uart-service-on-raspberry-pi/ and BlueZ's GATT API https://git.kernel.org/pub/scm/bluetooth/bluez.git/tree/doc/gatt-api.txt...but this isn't required to setup a STP. tldr; BlueZ runs on the Linux kernel, and D-Bus allows our Python code to communicate with BlueZ

To run the code, activate the smart-trash-picker Conda environment and run

```
python my-gatt-server.py
```

This runs the server indefinitely. If you setup the IR sensors to face each other and then move your hand in and out of the beam, you should see print statements in the console saying that we noticed the rising edge.

Testing

To test if your Pi is operating as intended, you can download the wonderful nRF Connect app on Android, which you can find here: https://play.google.com/store/apps/details?id=no.nordicsemi.android.mcp&hl=en_US. nRF Connect is great for debugging Bluetooth devices. After installing that app, you can scan for BLE devices and try to connect to your Pi.

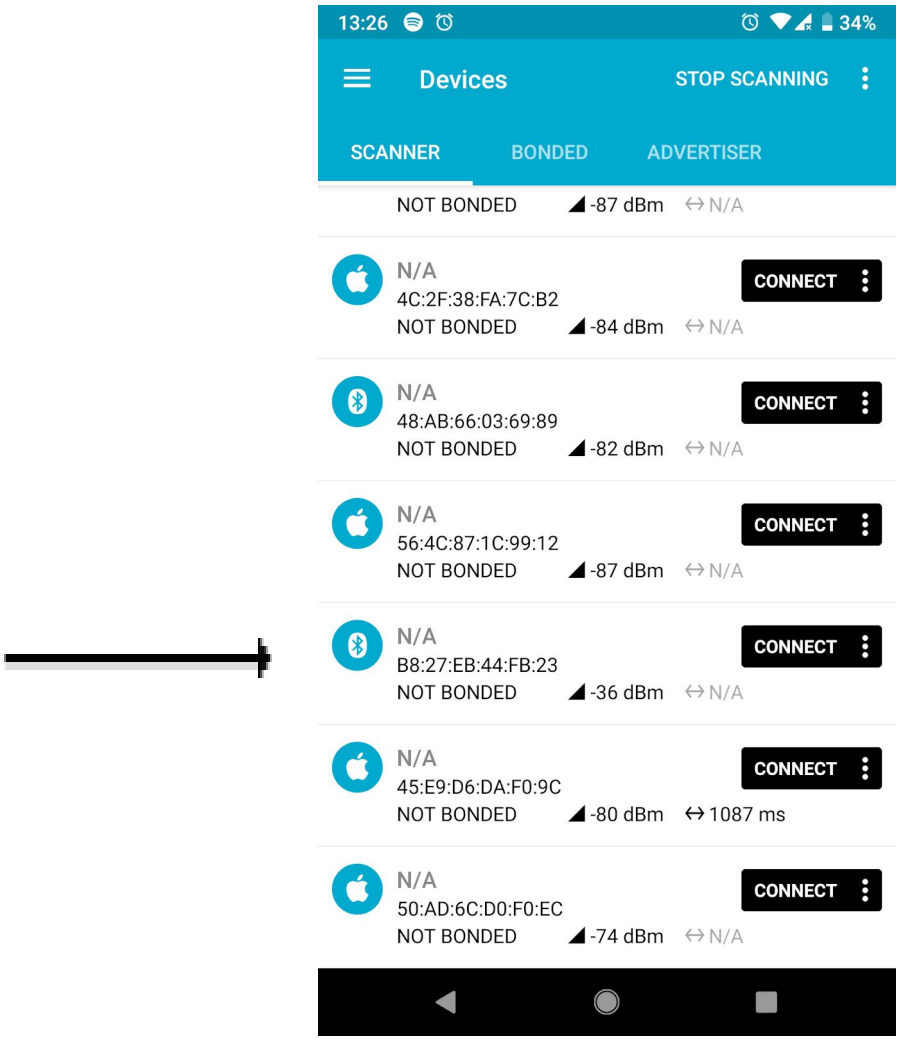
Note that you'll need to know the BLE address of you Pi to connect to it in the nRF Connect app. You can find your Pi's BLE address by running

```
sudo hciconfig hci0
```

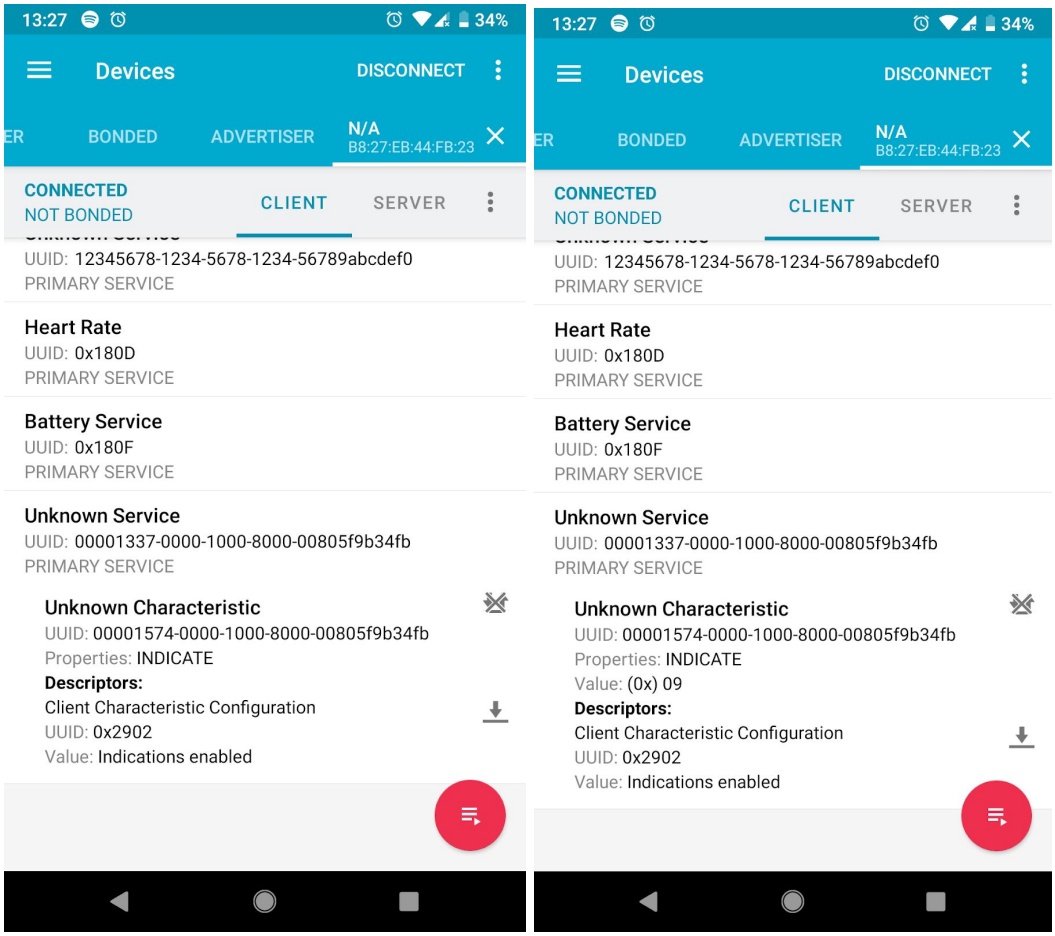
You'll get an output like below. The BLE address is highlighted in yellow

```
hci0:   Type: Primary  Bus: UART
        BD Address: B8:27:EB:44:FB:23  ACL MTU: 1021:8  SCO MTU: 64:1
        UP RUNNING PSCAN ISCAN
        RX bytes:7863 acl:0 sco:0 events:299 errors:0
        TX bytes:2695 acl:0 sco:0 commands:71 errors:0
```

Now open up the nRF Connect app and scan for devices. If everything is running correctly, you'll see your Pi's BLE Address in the list of devices.



Now click the "Connect" icon next to your Pi's address and connect to your Pi. If successful, you'll see a list of Services. For the STP, we only care about the Smart Trash Picker Service (0x1337), so tap the list entry corresponding to that service. Then tap the double arrow icon next to the Trash Grabbed Characteristic (0x1574)



Now go back to the IR sensors and use your hand to momentarily break the beam and then restore the beam. After you take your hand away, you should see a random number pop up in the "value" field of the Trash Grabbed Characteristic. If this happens, your Pi is set up correctly

Starting the GATT Server on Pi Boot

At this point we can start the GATT server manually, but we'd prefer if the server was started automatically when the Pi is powered on. To do this, I used **systemd** https://www.freedesktop.org/software/systemd/man/systemd.service.html. My Git repo includes the .service file I used (I stored it at /lib/systemd/system/ble-stp.service) for my final project. It starts the Bash script start_server.sh when the Pi boots up. See https://www.digitalocean.com/community/tutorials/how-to-use-systemctl-to-manage-systemd-services-and-units for an example of how to use that .service file and register it with systemd

Installing Android App

My code for the Android app is here: https://github.com/MichaelRechenberg/SmartTrashPickerAndroidApp. If you just want to see an example of BLE code for an Android app, you can look at the Activities and Services of the app here https://github.com/MichaelRechenberg/SmartTrashPickerAndroidApp/tree/master/app/src/main/java/mrechenberg/smarttrashpickerapp. You might also find the following articles useful:

- https://developer.android.com/guide/topics/connectivity/bluetooth
- https://medium.com/@avigezerit/bluetooth-low-energy-on-android-22bc7310387a (especially the CCCD section)

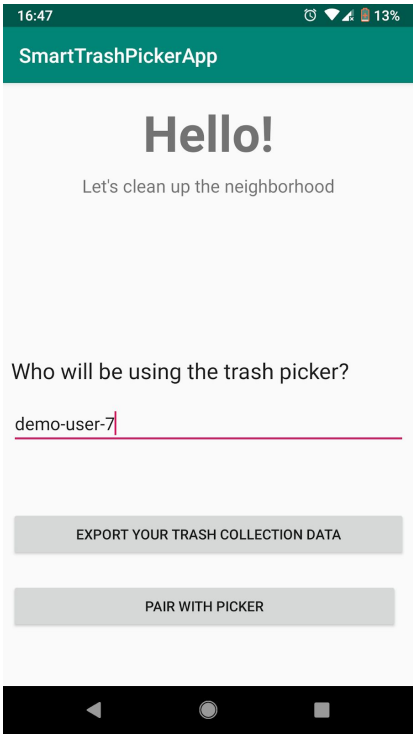
If you want to install my app on your own Android device, keep reading.

First, you'll need to install Android Studio on your computer https://developer.android.com/studio. Then you'll have to enable Developer Options on your Android device (see https://www.digitaltrends.com/mobile/how-to-get-developer-options-on-android/). Now open Android Studio and clone my Android app's Github repo. If everything goes well, Android Studio will download all the required packages after cloning the repo. Then you can plug in your Android device and deploy the app on your phone (see https://developer.android.com/training/basics/firstapp/running-app for more info on how to run Android apps on real devices)

Side note: the master branch doesn't have export functionality...for that you'll have to switch to the volley-export-data Git branch, then install the app on your device

To use the app, open it up and enter a username on the main screen (this is used to tag trash pickups to a particular user, so remember what name you use if you want to use the app over multiple days). Then click on "Pair With Picker". The app will scan for supported BLE devices (i.e. devices advertising the Smart Trash Picker Service) and list the addresses of any devices found. You may have to scan multiple times if your STP doesn't show up at first. Then click the address of the STP you want to pair with.

Once you're paired, you can use the STP like usual--picking up trash. When you're done for the day, click the notification for the app and disconnect from the picker.



To export your trash data, you'll have to create a HTTP server that accepts a POST request of JSON data at the endpoint "/export". I made a quick one using Flask in a couple of minutes. To export your data, go back to the home screen, enter your username, and click "Export Your Trash Collection Data". Then enter in the IP address and port of your server and hit "Export". If everything goes correctly, your trash data should be sent to the server as a JSON array.