Fingerprint Scanner Robot Documentation

Release 1.1

Jacob Glueck

July 17, 2015

CHAPTER

ONE

INTRODUCTION

The fingerprint scanner robots are designed to automatically test fingerprint scanners. There are two robots:

1. NAFSTR: Networked Automated Fingerprint Scanner Test Robot. A robot with four individually-actuated fingers that simulates simple fingerprinting.
2. NARFSTR: Networked Automated Rolled Fingerprint Scanner Test Robot. A robot that simulates rolling a finger across a fingerprint scanner.

Both robots are controlled by an Arduino Uno, and have an Arduino Ethernet Shield. They accept commands sent over USB and over a network.

CHAPTER

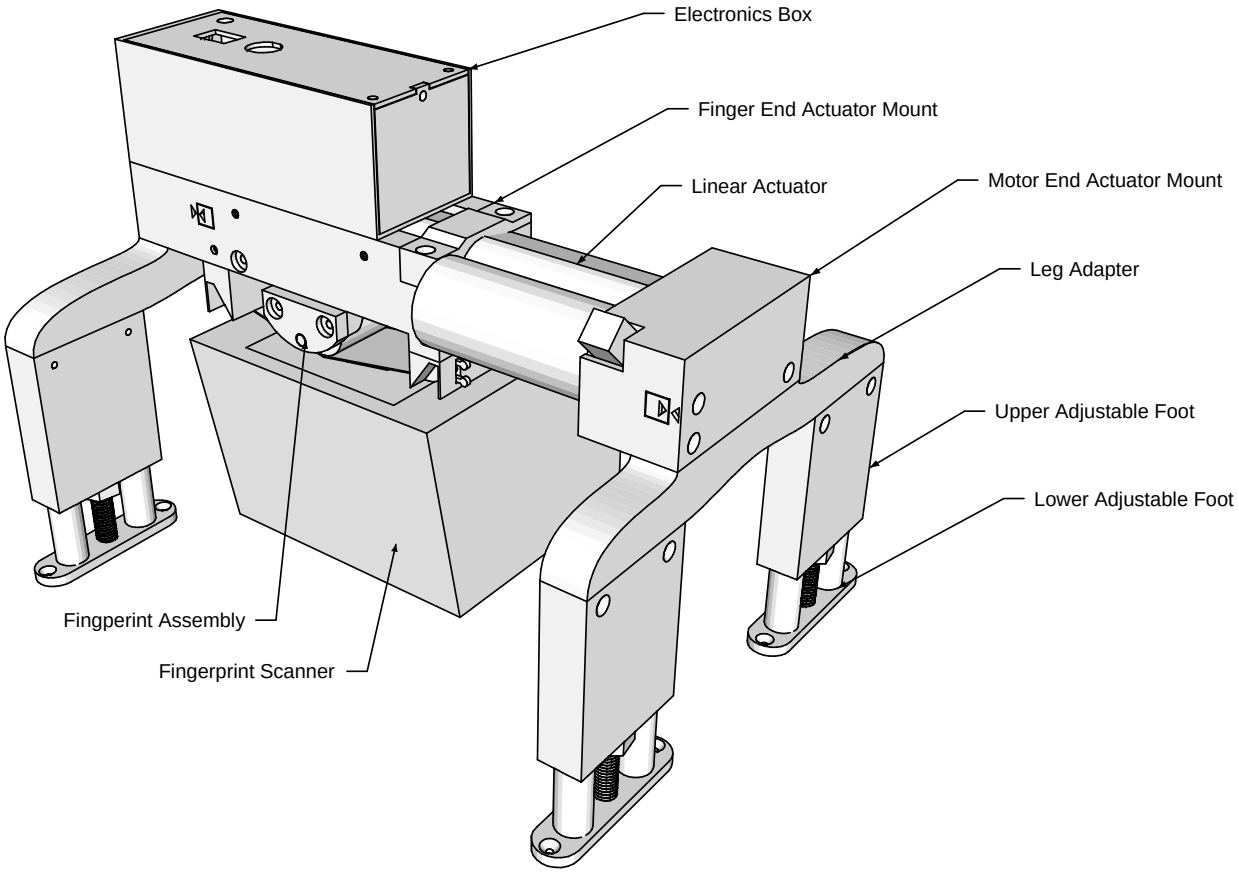
TWO

NARFSTR

2.1 Design

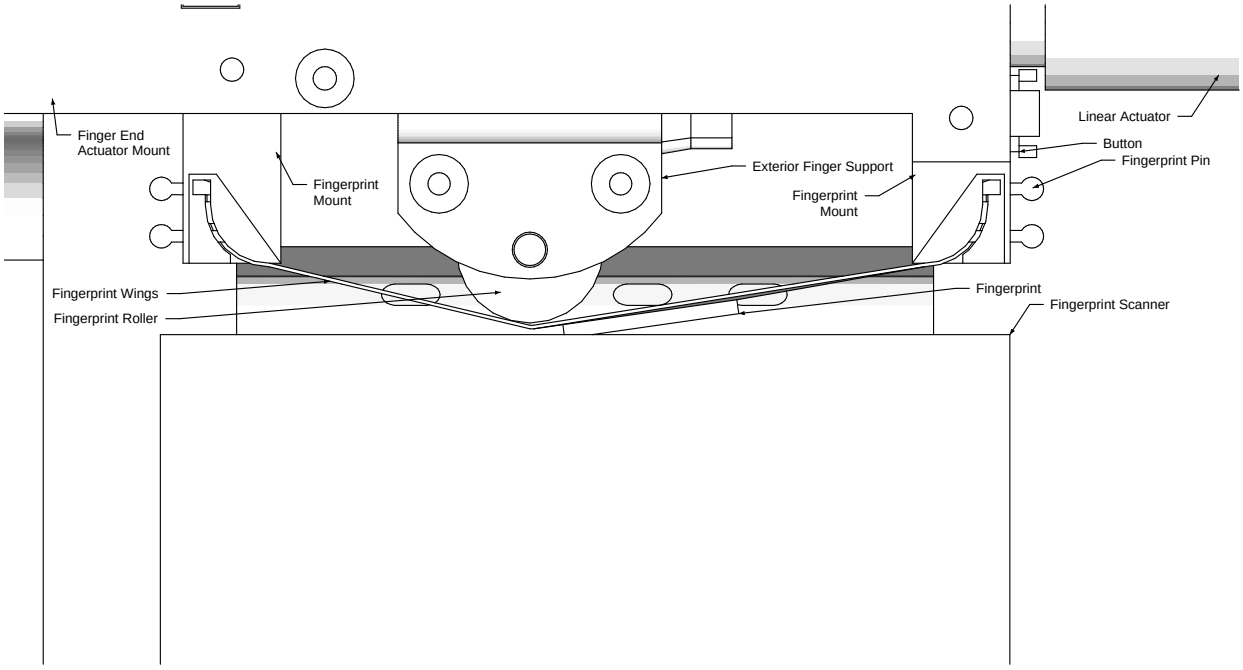
2.1.1 Structure

The robot has four adjustable legs, which support a linear actuator, the electronics, and the fingerprint assembly. The following image labels all the main parts:



Fingerprint Assembly

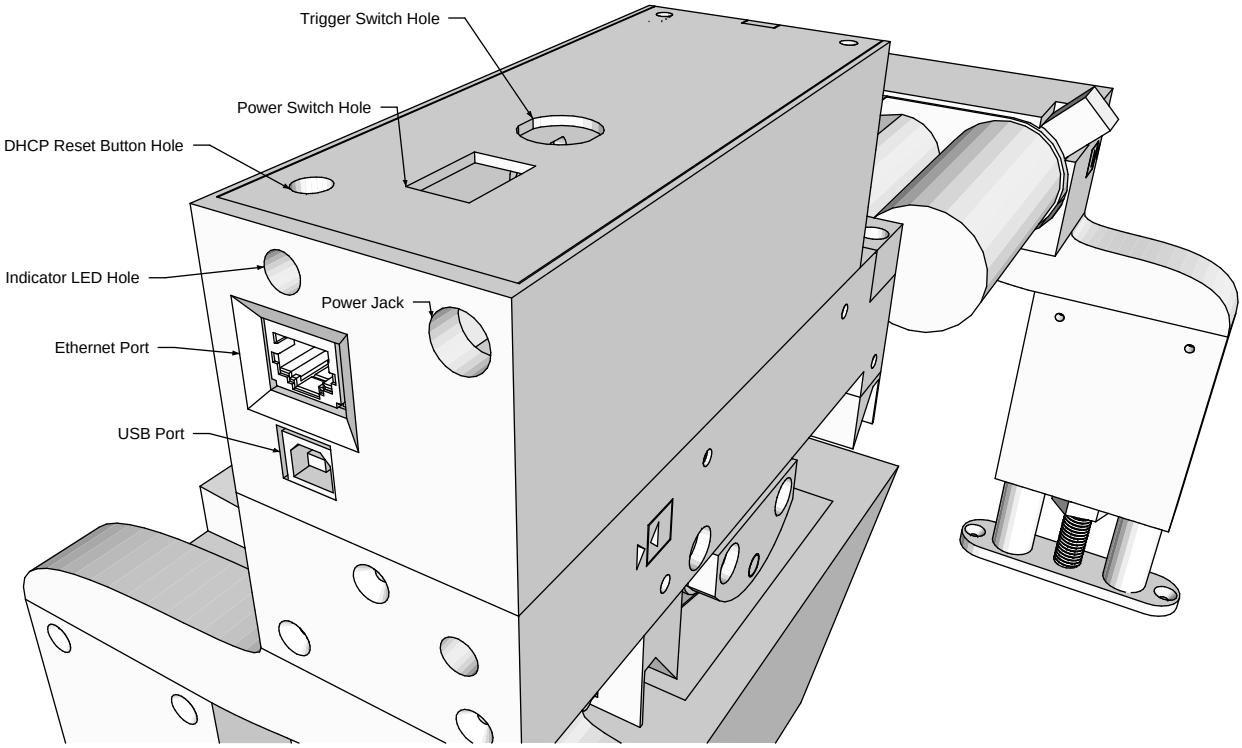
The fingerprint consists of a fingerprint, held by two supports. A roller rolls over the fingerprint, pressing it against the scanner. The following image shows how it works:



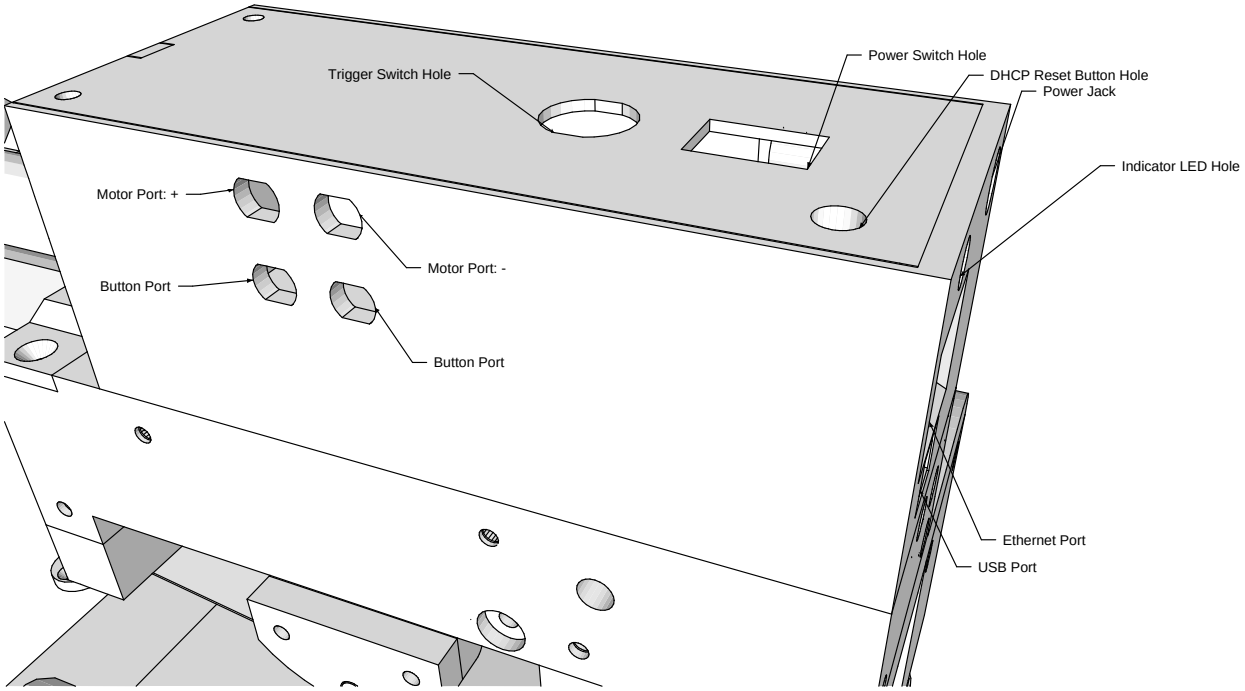
The linear actuator pulls the roller towards the button. As the roller moves, it compresses the fingerprint against the scanner, causing the scanner the capture a rolled print. Once the moving assembly (consisting of the roller and its support structure) hits the button, the linear actuator stops, and then reverses.

Electronics Box

The electronics box contains openings for all the connections and buttons on the robot. The following picture shows all the main features:



There are also four more openings on the back. These are to allow the linear actuator and the button in the fingerprint assembly to connect to the electronics in the box. The following image shows these connections:



2.2 Construction

2.2.1 Parts

3D Printed Parts

Almost all of the parts for the robot are 3D printed. The STL files for the parts are located in `Final STL Files/NARFSTR`. All of the 3D printed parts were printed in PLA except for the fingerprints. (For more information abou the fingerprints, see *Fingerprints*.) The PLA parts were printed with a layer height of 0.2 mm using MakerBot's "Standard" profile. We used a MakerBot Replicator 2 and a 5th Generation Replicator. The following table lists each part's name and quantity.

Table 2.1: 3D Printed Parts

Part Name	Quantity
Actuator Mount - Finger End	1
Actuator Mount - Motor End	1
Actuator Pin - Finger End	1
Actuator Pin - Motor End	1
Actuator Strap	1
Bolt Head	4
Button Pusher	1
Button Strap	1
Electronics Box	1
Electronics Box Front Tab	1
Electronics Box Side Lid	1
Electronics Box Top Lid	1
Fingerprint	1
Fingerprint Base	1
Fingerprint Bracing	2
Fingerprint Clip	2
Fingerprint Mount - Finger End	1
Fingerprint Mount - Motor End	1
Fingerprint Roller A	1
Fingerprint Roller B	1
Leg Connector - Finger End	1
Leg Connector - Motor End	1
Leg Lower	4
Leg Upper	4
Lock Nut	4
Power Jack Holder	1

Electronic Parts

The following table lists all the electronic parts. Parts that are marked with * are close to the parts I used, but may not be exact. This is because the parts I used came from kits or other sources, and I was unable to determine their exact information. For the male and female banana plugs, 3 different colors are recommended: 1 red, 1 black, 2 blue.

Table 2.2: Electronic Parts

Part Name	Manufacturer	Number	Quantity
10 kΩ resistor			3
12 mm push button*	Sparkfun	COM-09190	2
12V 5A switching power supply	Adafruit	352	1
20 gauge stranded wire			1
470Ω resistor			3
5 cm x 7 cm double sided perfboard	Amico	s14011600am1317	1
Arduino Ethernet shield	Arduino	A000056	1
Arduino Uno R3	Arduino	A000066	1
Common cathode RGB LED*	Sparkfun	COM-00105	1
Female tab connector	TE Connectivity / AMP	3-350820-2	3
LED Cap*	Keystone	8665	1
Linear actuator - 5” Stroke	Firgelli Automations	FA-TR-35-12-5”	1
Male tab connector	TE Connectivity / AMP	3-520107-2	3
Power pigtail female	Monoprice	6881	1
Solderless banana plug female	Johnson	108-0910-001	4
Solderless banana plug male	Johnson	108-0310-001	4
Solid core mounting wire kit	Velleman Components	K/MOWM	1
SPST 9 amp on off illuminated green	Jameco Valuepro	R13-66B-G-02	1
Switch push button SPST off momentary (on) red amp LED illuminated	Jameco Valuepro	R13-508AL-05-BRR-L3	1
UBEC DC/DC step-down (buck) converter - 5V @ 3A output	Adafruit	1385	1
VNH5019 motor driver carrier	Pololu	1415	1

Screws and Miscellaneous Parts

I did not order these parts, and so have limited information about them.

Table 2.3: Screws and Miscellaneous Parts

Part name	Color	Quantity
19 mm diameter rubber feet	Black	8
#4 US wood screw, 15.7 mm	Gold	17
#4 US wood screw, 8.5 mm	Black	19
#8 US machine screw, 12 mm	Silver	4
Epoxy		
Super glue		

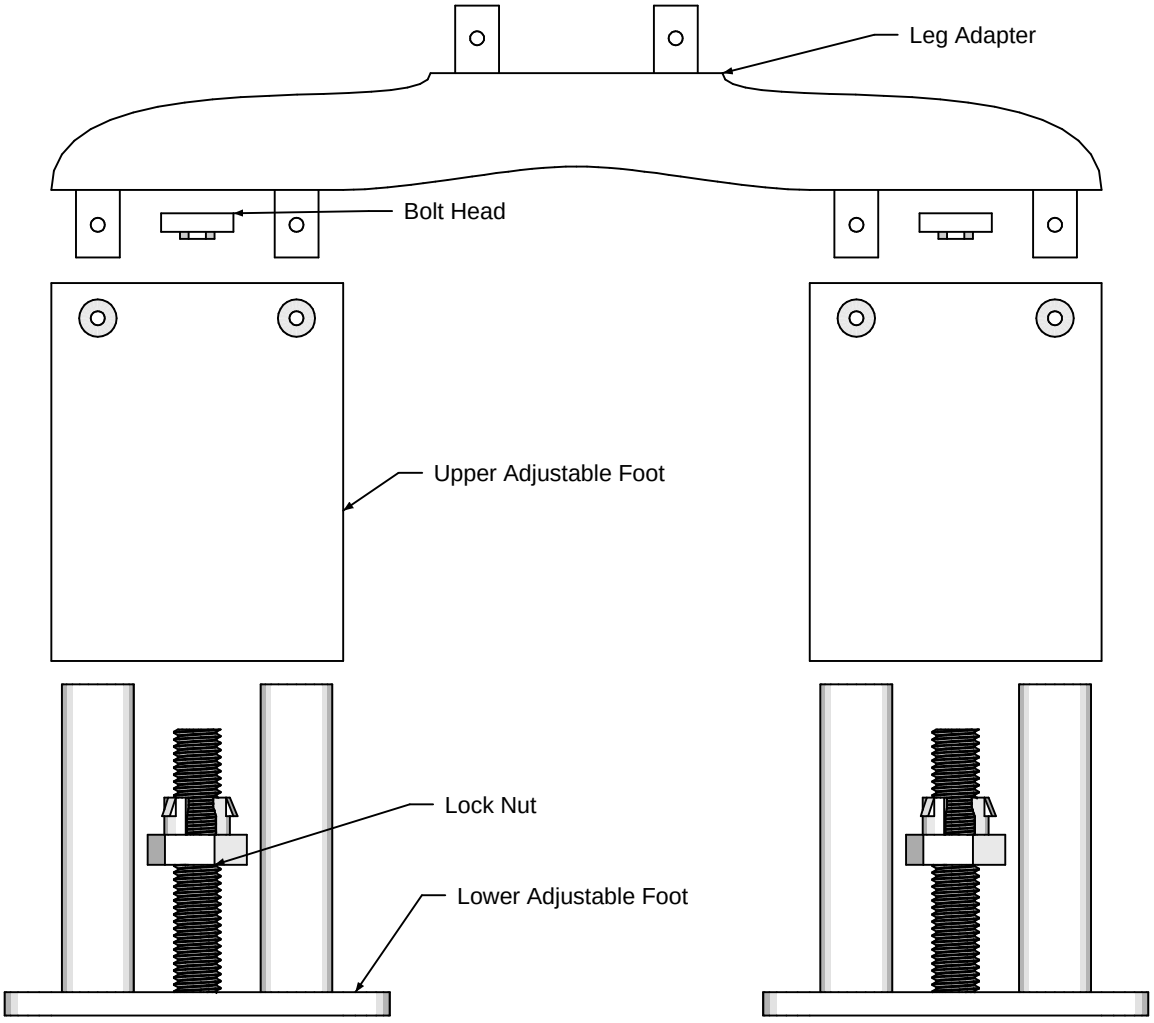
Here is a picture of the screws:

From now on, I will refer to the screws by their color.

2.2.2 Legs and Actuator Mount Assembly

Legs

There are two leg assemblies: one on the side of the actuator with the motor, and one on the other side. Both are assembled the same way, but they use different leg adapters. To assemble a leg assembly, first screw a lock nut onto the threaded bolt in the center of the lower adjustable foot. Repeat for the other leg. Then, put an upper adjustable foot piece on top of each lower adjustable foot piece. Push down until the lock nut clicks, locking the lower part to the upper part. Then, insert a bolt head into the square hole on top of each upper adjustable foot. Screw the square head into the bolt using one of the black screws. Finally, connect each foot assembly to the lower part of a leg adapter. Secure the connection by screwing golden screws into the holes on the upper adjustable foot. Repeat this process with the other leg adapter to make the other leg assembly.

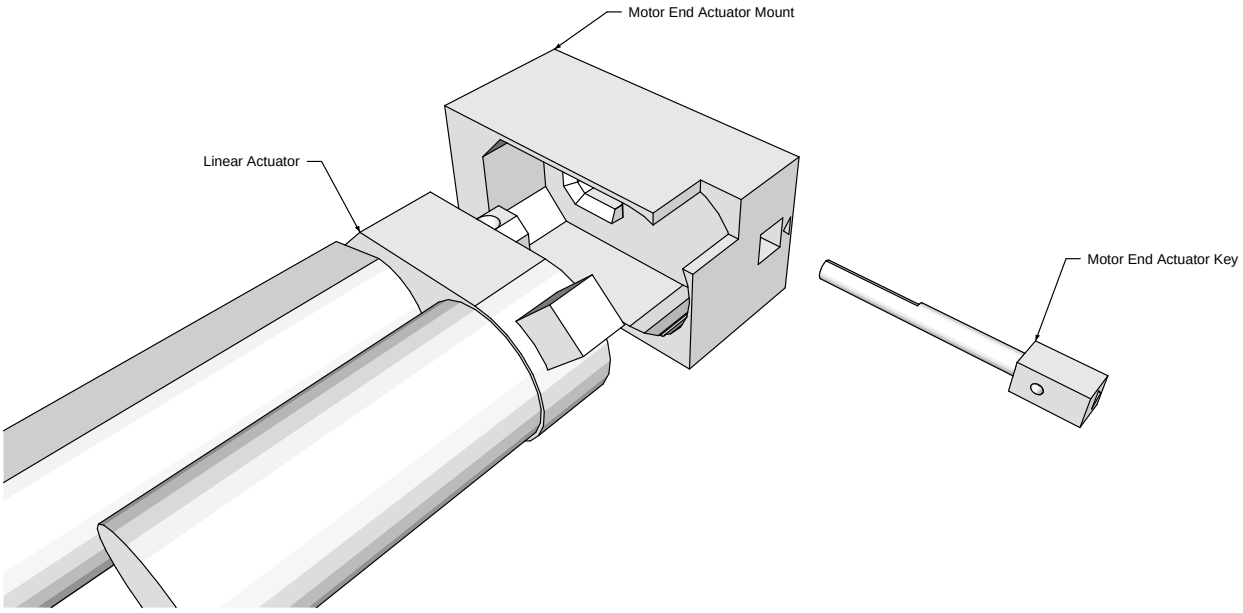


Actuator Mounts

There are two actuator mounts: one for the side of the actuator with the motor, and one for the other side.

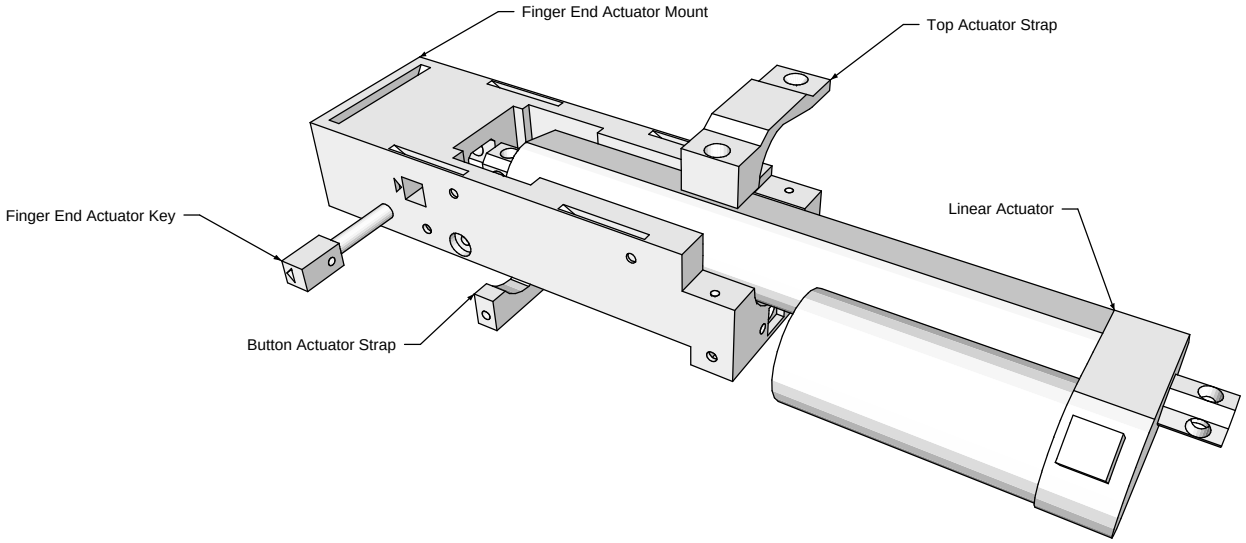
Motor End Actuator Mount

Slide the actuator into the mount. Insert the motor end actuator key into the hole in the actuator mount. Make sure the arrows point towards each other. Secure the key with a golden screw.

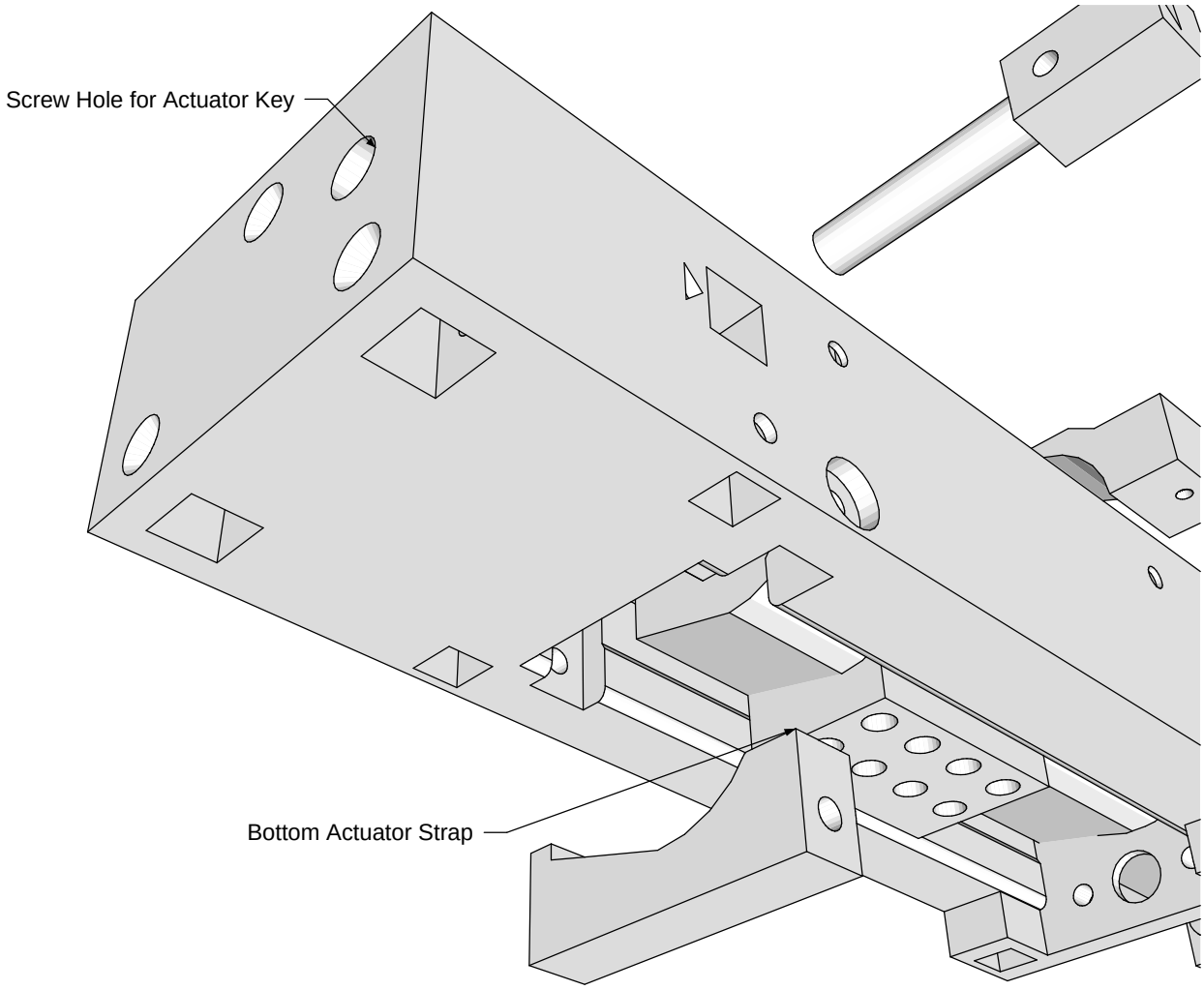


Finger End Actuator Mount

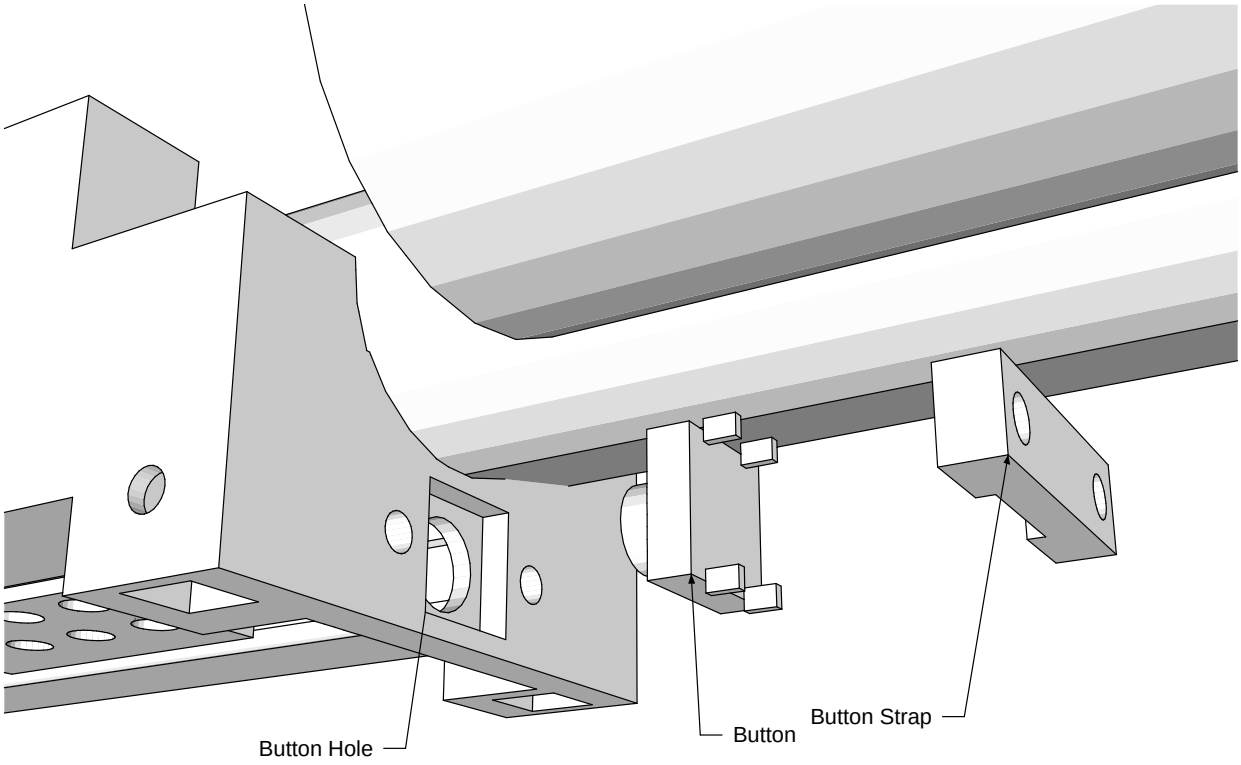
Slide the actuator into the mount. Insert the finger end actuator key into the hole in the actuator mount. Make sure the arrows point towards each other. Attach the top actuator strap to the actuator using two golden screws.



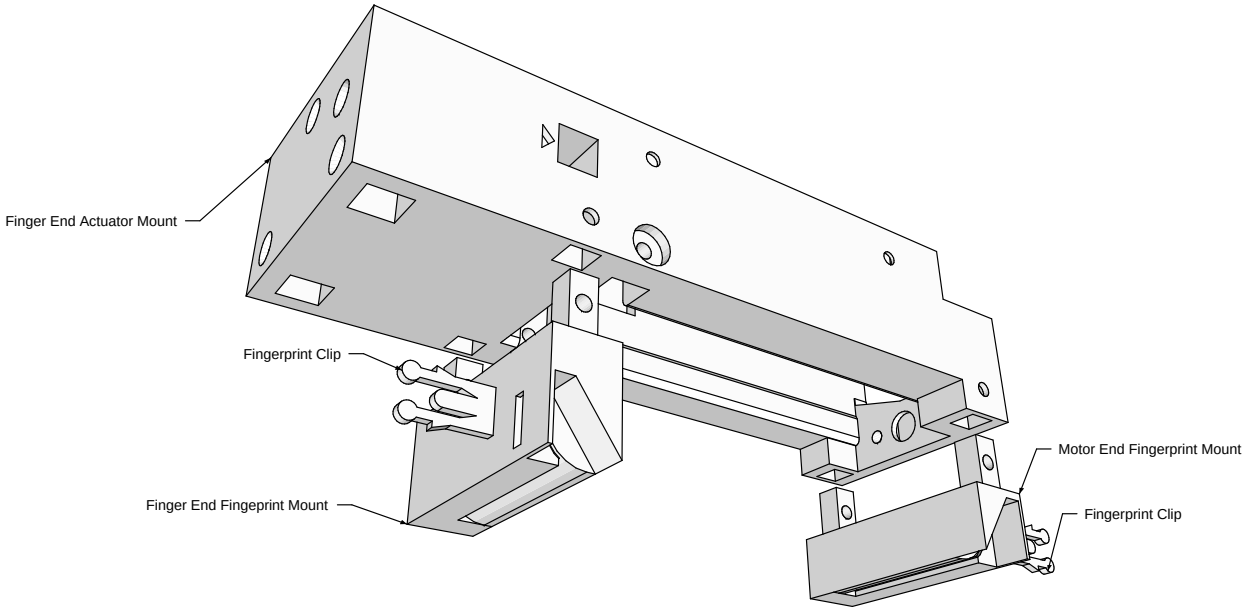
Insert the bottom actuator strap as shown in the diagram below. Secure it with two black screws. Finally, screw a golden screw into the hole labeled “Screw Hole for Actuator Key” on the diagram below. This secures the key.



Put a 12 mm push button into the button hole on the actuator mount. Secure the button using the button strap and two black screws.

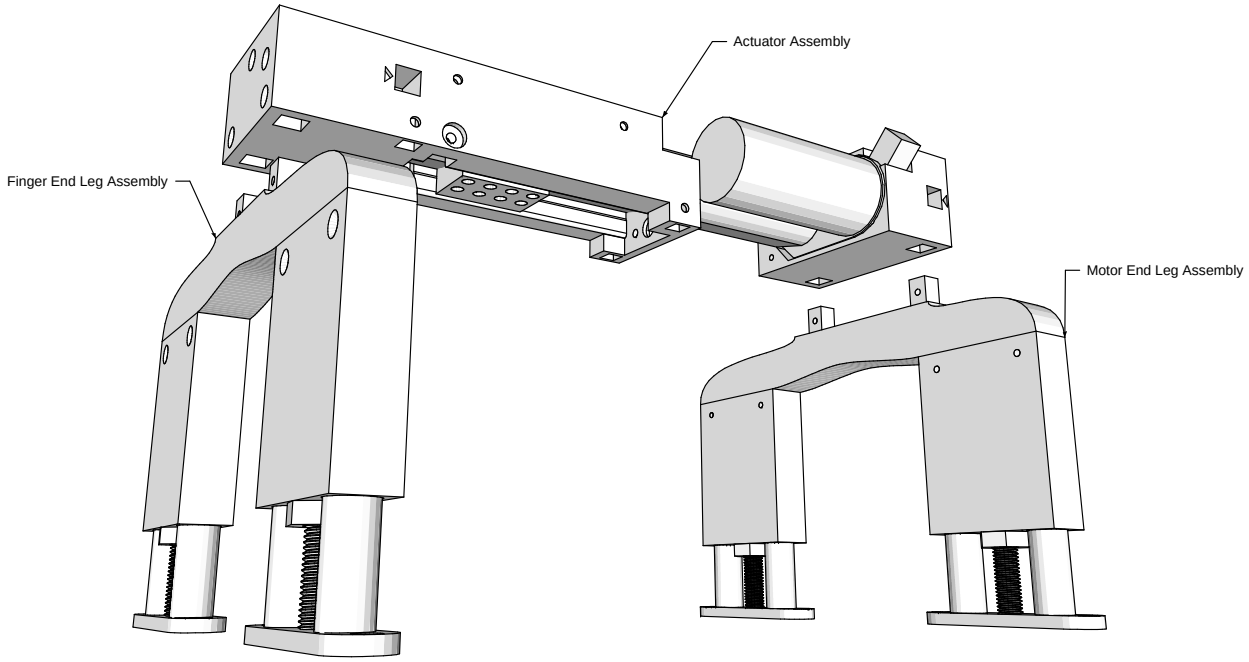


Insert the finger end fingerprint mount and the motor end fingerprint mount into the bottom of the actuator mount. Secure each one with two black screws. Then, insert a fingerprint clip into each of the fingerprint mounts.



Connecting the Legs and Actuator Mounts

Insert the leg assemblies into the bottom of the actuator assembly. Secure each leg assembly with two golden screws.

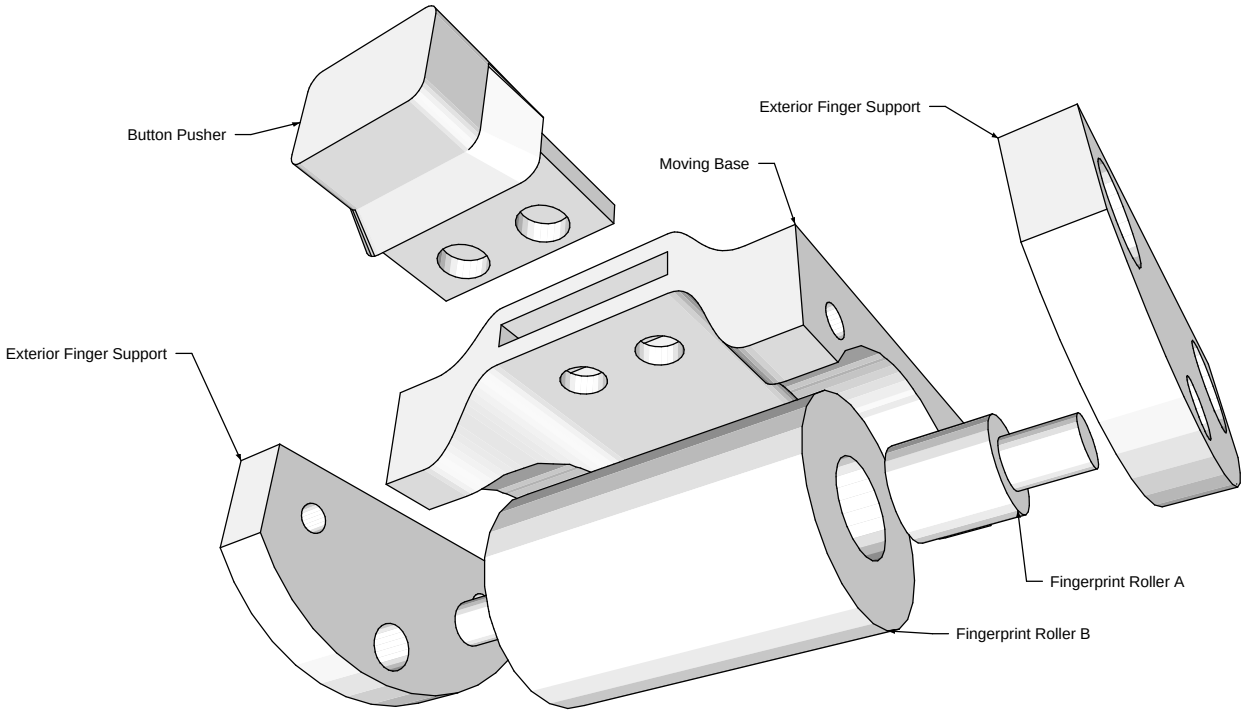


Rubber Feet

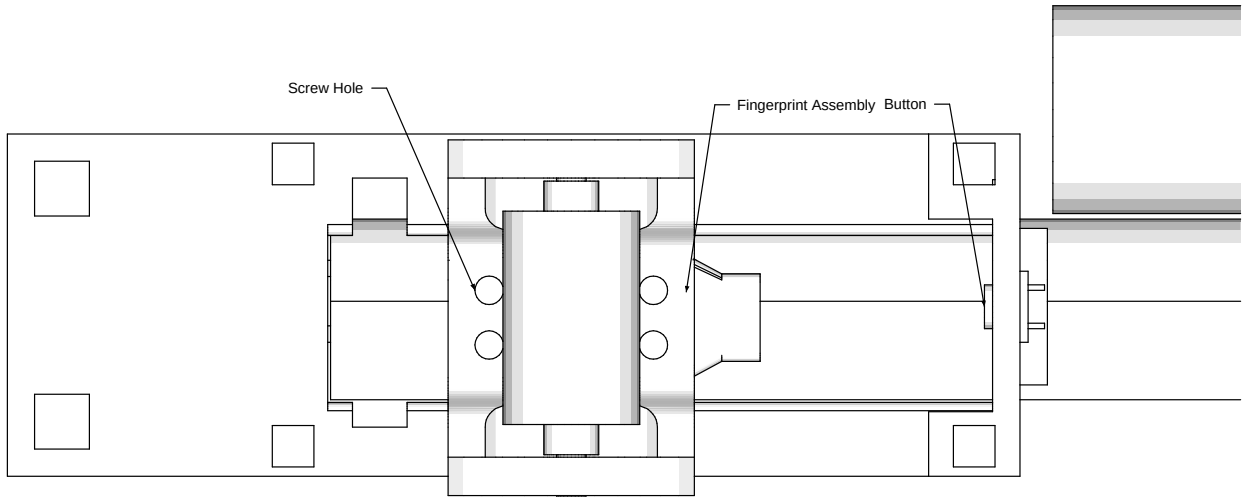
Insert two rubber feet into the bottom of each foot.

2.2.3 Fingerprint Assembly

Glue fingerprint roller A into fingerprint roller B using super glue. Let dry. Connect fingerprint roller to moving base with the exterior finger supports. Secure each support with two black screws. Insert the button pusher into the moving base.



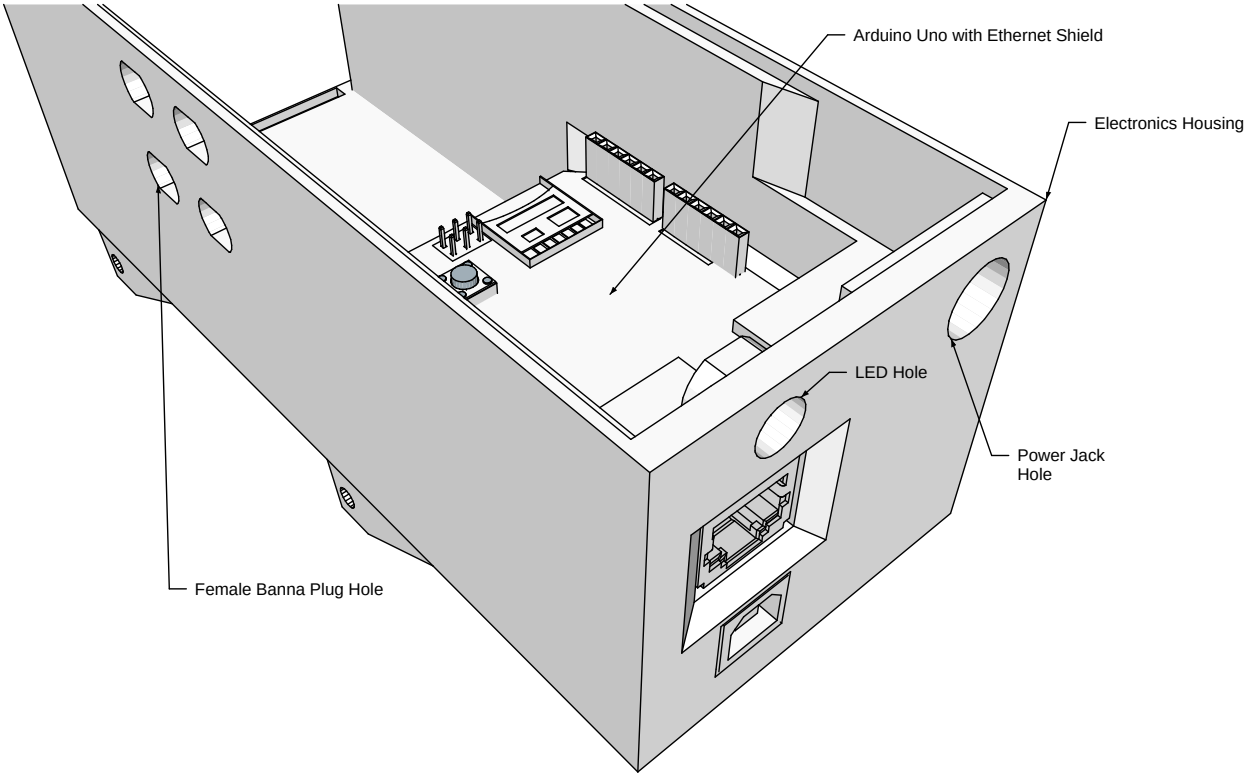
Attach the fingerprint assembly to the actuator by first aligning the four holes on the moving base with the holes on the actuator, ensuring that the button pusher faces the button. Then, screw a silver screw into each of the four holes on the moving base.



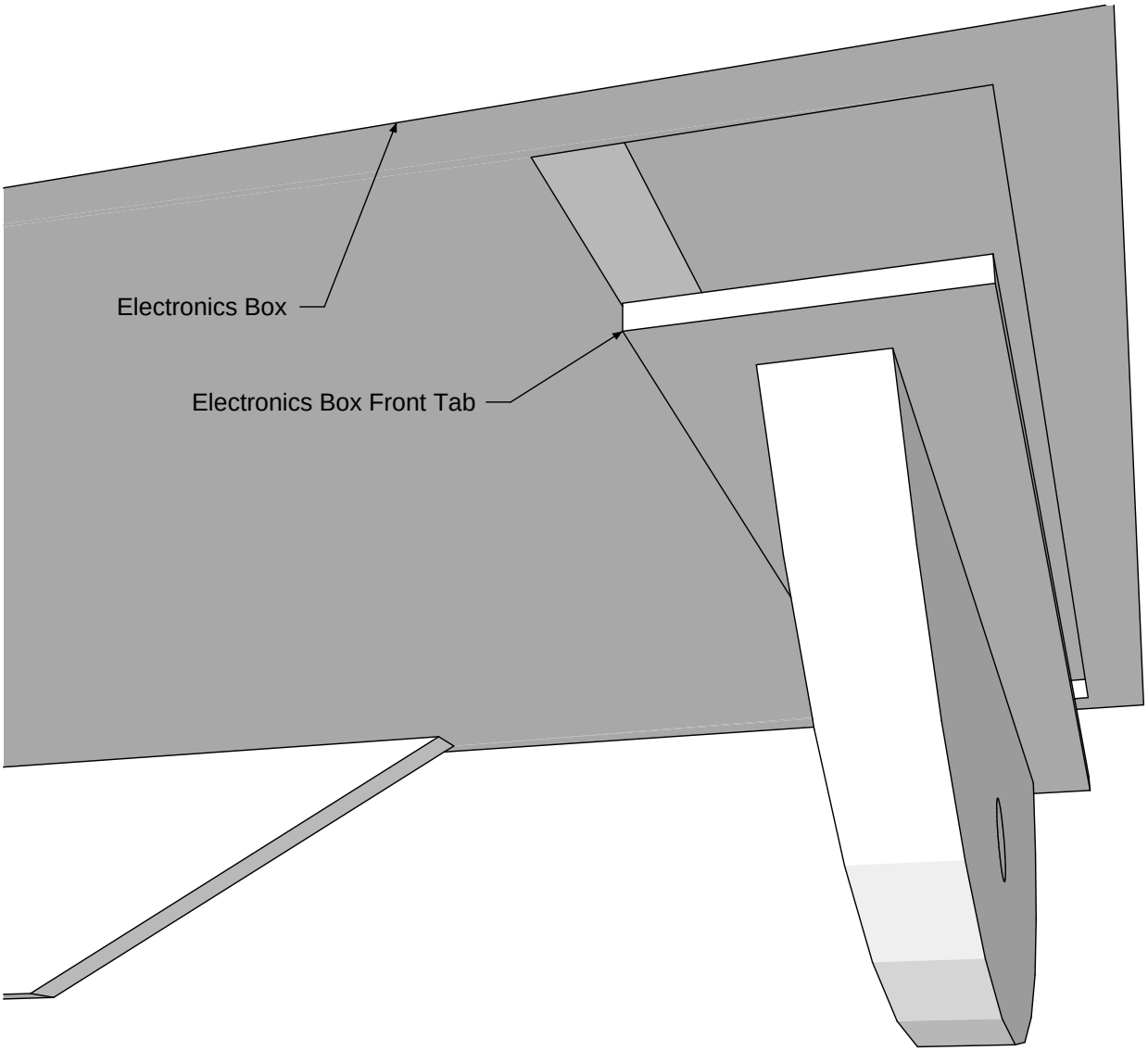
2.2.4 Electronics Box Assembly

Note: assemble the electronics before assembling the box.

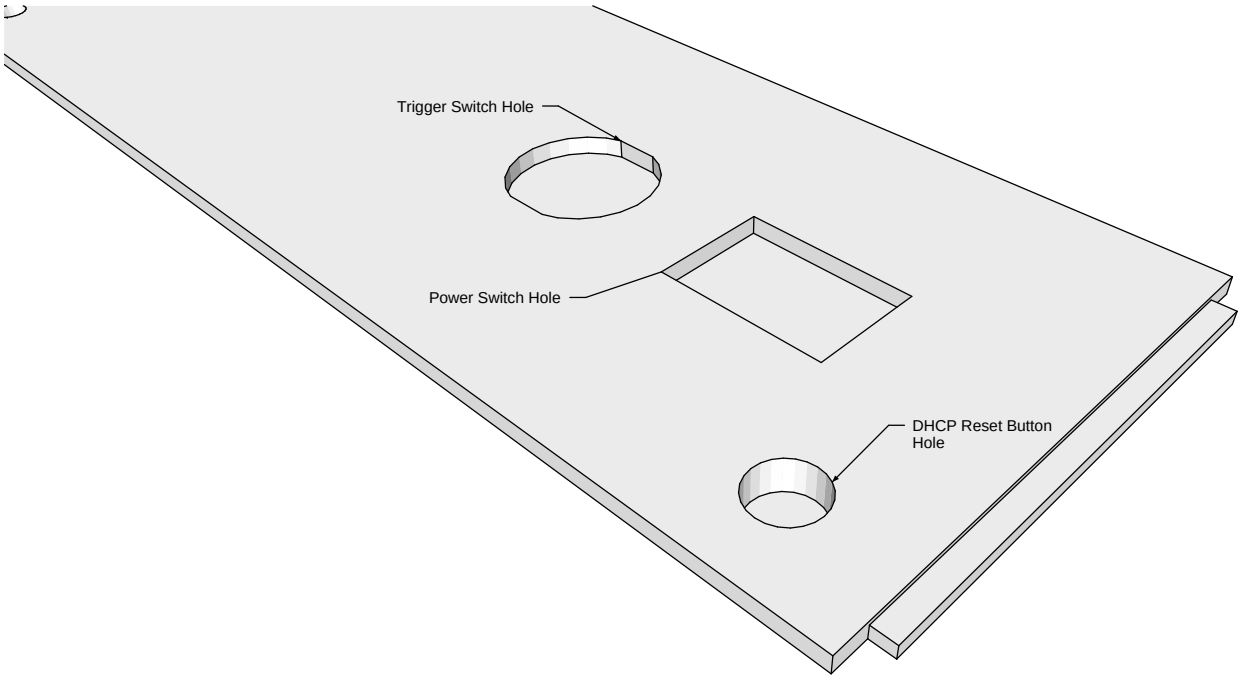
Slide Arduino Uno (with Arduino Ethernet Shield) into the box. Screw female banana plugs into holes of electronics housing. If using colors, put a red jack and a black jack into the top row, and put two blue jacks in the bottom row. Then, super glue the LED cap into the LED hole.



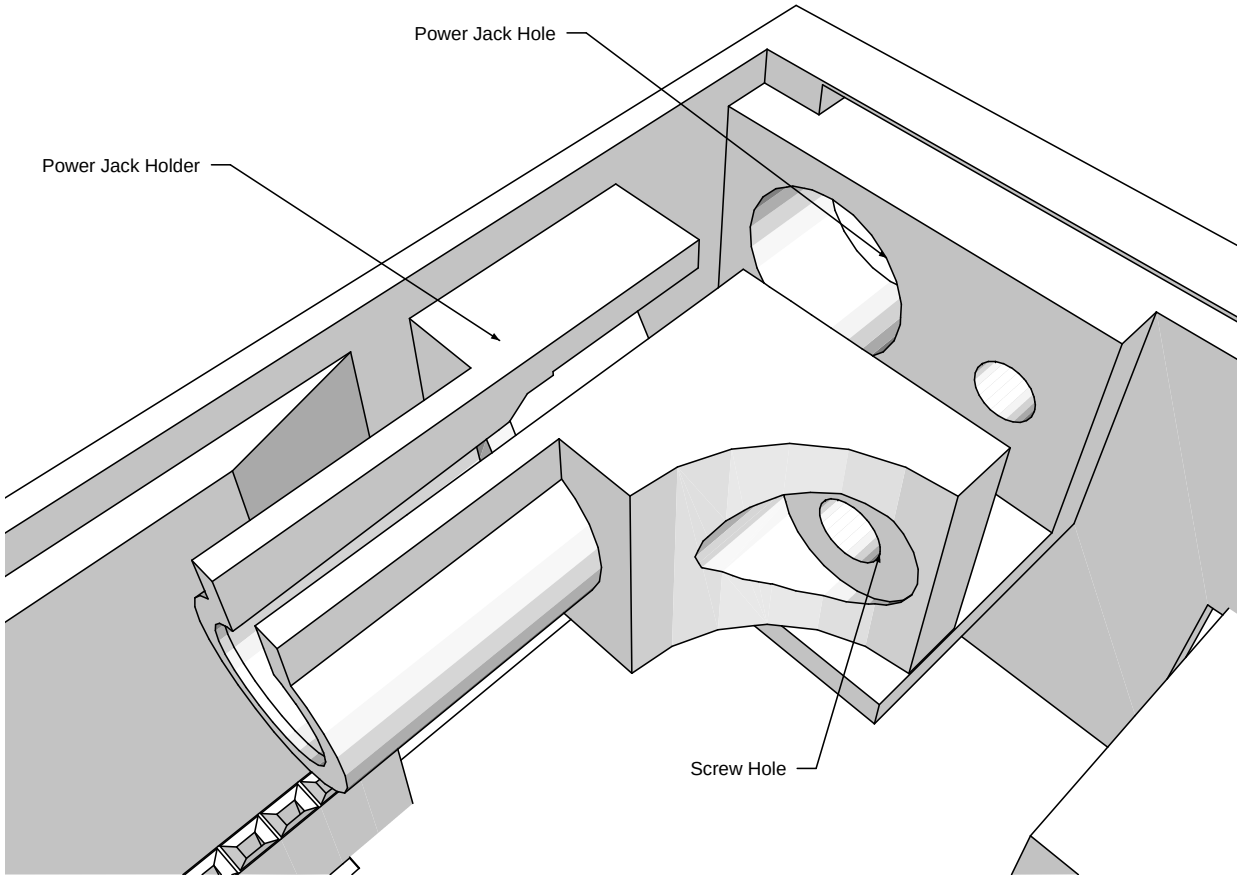
Super glue the front tab onto the bottom of the electronics box.



Assemble the top lid by inserting the power switch and the trigger switch into their holes. The power switch is the green rocker switch, and the trigger switch is the red push button. Using epoxy, glue a 12 mm button into the DHCP reset button hole. The square base of the button should be glued onto the plastic square on the bottom of the top lid.



Assemble the power jack by inserting a female power pigtail into the power jack holder. This may require some force. Screw power jack holder into electronics box using a black screw.



Put all other electronics into the box in the empty area behind the Arduino and put LED into LED cap. Ensure that everything is properly connected. Then, put top on box and secure with two black screws. Put back side on box and secure with one black screw. Finally, connect all cables.

2.3 Electronics

2.3.1 Design

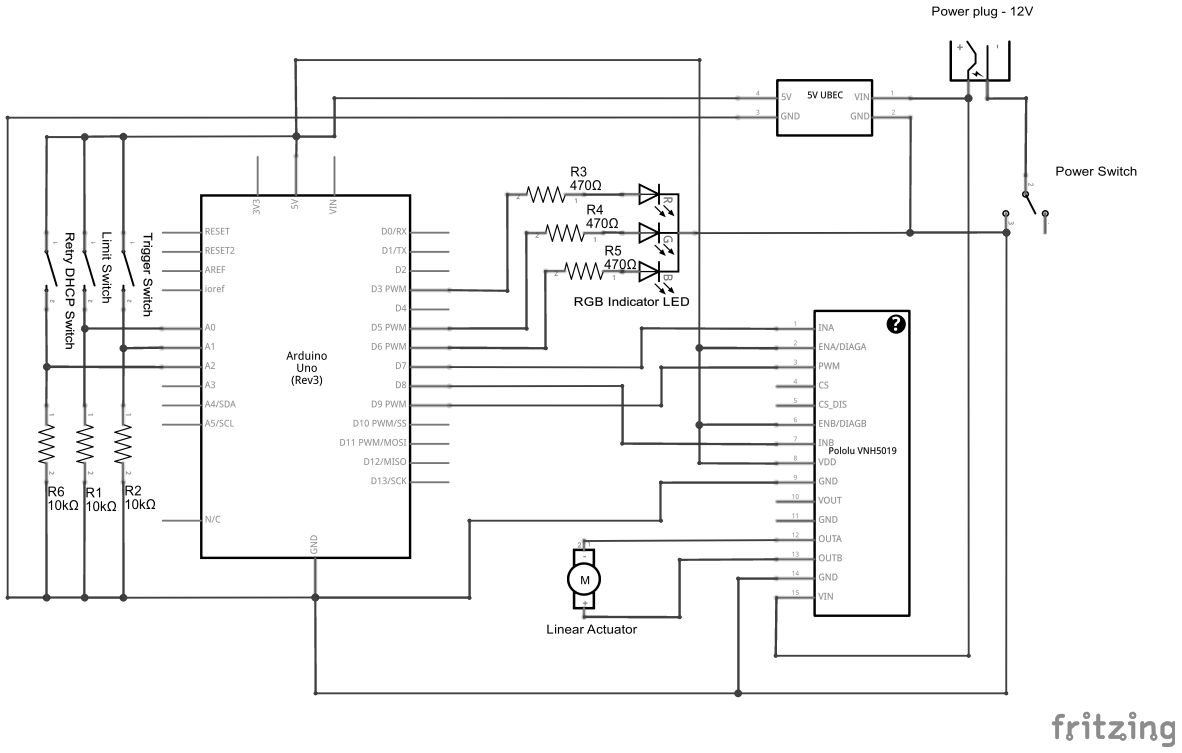
The robot is controlled by an Arduino Uno microcontroller. On top of the Arduino, there is an Arduino Ethernet Shield, which, through its single Ethernet port, allows the robot to connect to a network. The following components are connected to the Arduino through the header pins on the shield:

1. RGB LED: serves as an indicator light.
2. Trigger switch: a red pushbutton, causes the robot to move its finger back and forth when pressed.
3. Limit switch: a 12 mm push button which detects when the fingerprint has finished rolling.
4. DHCP reset button: a 12 mm push button, causes the robot to request a DHCP lease.
5. Pololu VNH5019 motor driver chip: used to drive the linear actuator.

The robot has a single 2.1 mm DC barrel jack for power and requires 12 V DC at around 3A (estimated; based on peak motor current draw). The 12 V lines are attached directly to the motor control board, and the Arduino is powered by the output of a 5V Universal Battery Elimination Circuit (UBEC). The UBEC can produces a regulated 5 V output at a current of up to 3A. This works far better than the Arduino's built-in regulator, which, though it can handle 12 volts,

gets very hot when doing so and has trouble providing the necessary current for both the Arduino and the Ethernet Shield. The robot also has a power switch, which is connected to the 12 V input line.

Here is a schematic:



There is no Ethernet Shield in the schematic, but it would be placed on top of the Arduino.

2.3.2 Assembly

First, put the Arduino Ethernet Shield onto the Arduino. Then, on the 5 cm x 7 cm perfboard, assemble the circuit using the schematic above. For the motor and limit switch, which are mounted outside of the box, connect their leads to the female banana plugs, which will go into holes on the side of the electronics box. On the ends of the leads for the motor and button (which will be outside the box), attach the male banana plugs. Use the two blue plugs for the button's leads, the red one for the motor's red wire, and the black one for the motor's black wire. For the LED and other buttons, make sure they are connected with long wires so they can reach from the circuit board to their designated location in the box. Also, do not directly connect any components that will be mounted in the box directly to the circuit board. You will not be able to put the components into their holes. Instead, use the tab connectors to attach the component to the rest of the circuit. Finally, on all the 12 V lines (which, because they are connected to the motor, would have high currents), use the 20 gauge wire.

2.4 Fingerprints

The fingerprints used on the robot are 3D printed. The first step in making a fingerprint is to generate a PNG image of the fingerprint. To do this, I used SFinGe (http://biolab.csr.unibo.it/research.asp?organize=Activities&select=&selObj=12&pathSubj=111%7C%7C12&) (**S**ynthetic **Fin**gerprint **Ge**nerator) from the Biometric System Laboratory at the University of Bologna . With the program, I configured various parameters of the fingerprint, and once the ridges were generated, clicked the "View

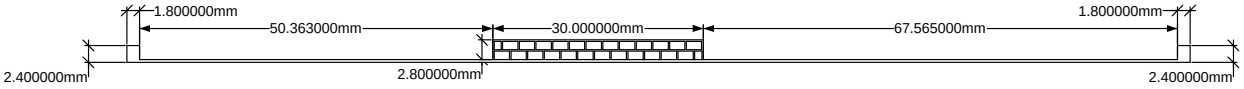
original" button after every step to remove all the distortion and scratches. Once the fingerprint was generated, I saved it to a bmp file and then converted it to a png.

To turn the PNG into a 3D-printable file, I wrote an OpenSCAD script, located at `Fingerprints/NARFSTRRolledFingerprints/roll.scad`. This script can take any black and white PNG image and produce a STL file that, when 3D printed, will fit perfectly into the robot. To use the script, configure the variables at the top of the file:

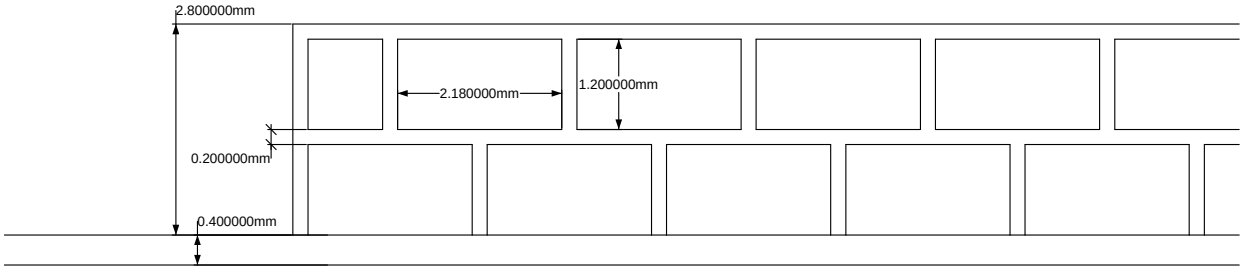
```
imageResolution = 19.7; //dots per mm
imageX = 416; //px
imageY = 560; //px
file = "FGen1.png";
fingerprintX = 30; //mm
fingerprintY = 30; //mm
fingerprintScale = 1.4;
```

Once these variables are configured, hit "Preview" in OpenSCAD to make sure everything looks good, and then hit "Render". This may take about an hour. Once the rendering is complete, export the file as an STL.

This script will generate a very complex 3D file which contains the fingerprint, mounted on a spongy area, which is mounted on tabs that connect to the robot. Here is a cross-sectional view of the finished product:



The two tabs at either end allow the fingerprint to be secured to the robot. The fingerprint is on top of the raised area in the middle. The holes in the raised area allow the fingerprint to compress, which results in a better scan. Here is a close up view of the holes:



To 3D print the STL file, use MakerBot Flexible Filament. Be sure to set the printing profile to the custom profile located at `Fingerprints/ NARFSTR Rolled Fingerprints/fingerprintprofile.txt`. The custom profile, which is based off of the MakerBot flexible filament profile, uses 1 shell and an extruder temperature of 140 °C.

CHAPTER

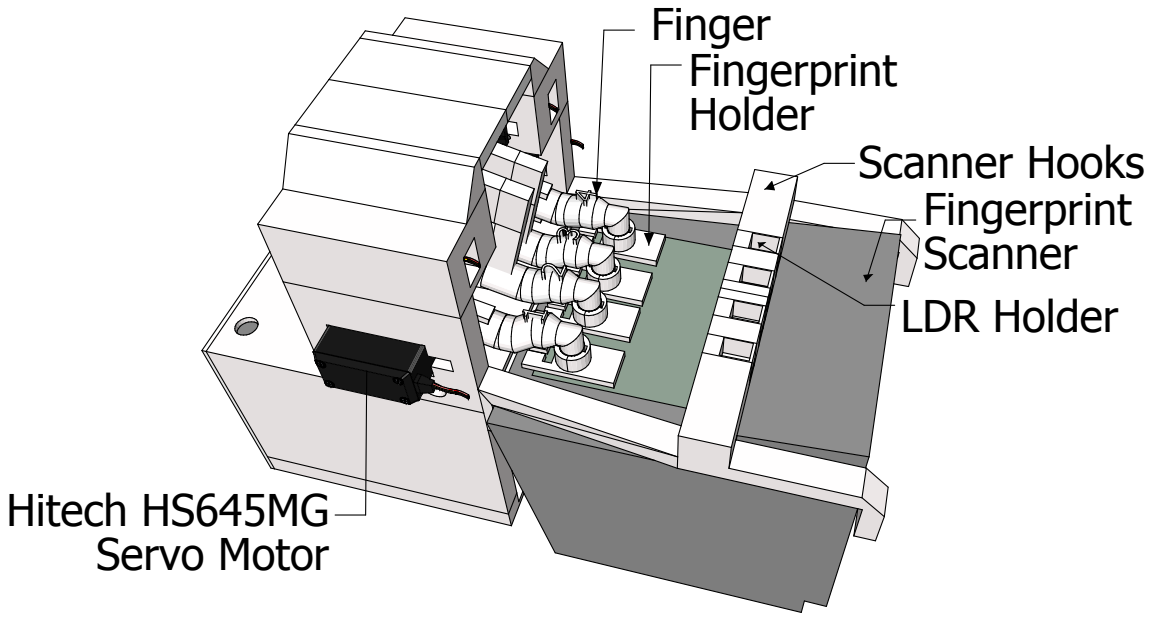
THREE

NAFSTR

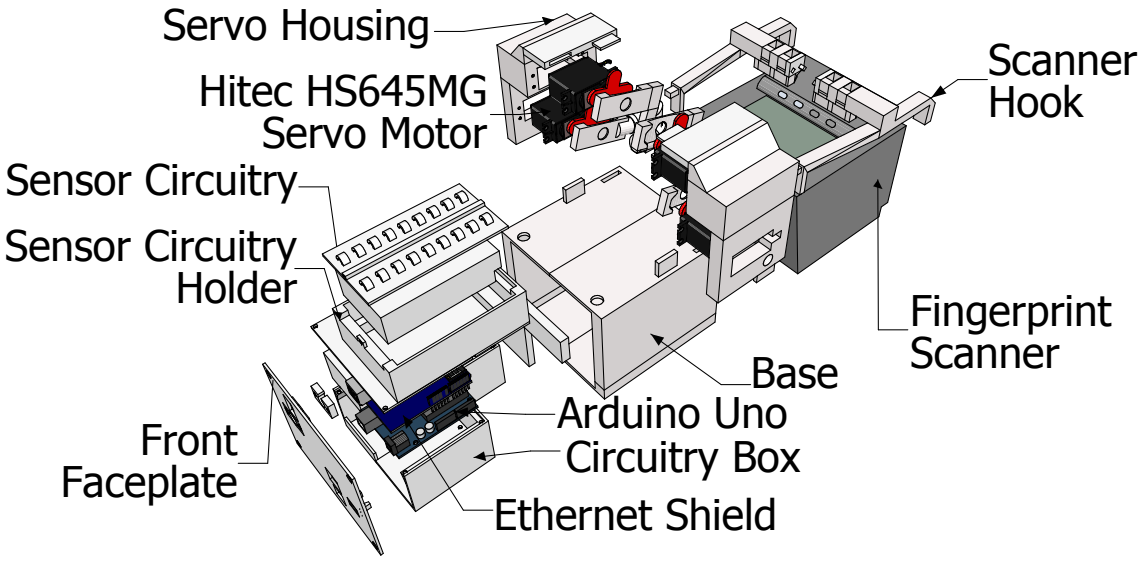
Note: The documentation below is sparse and needs to be improved. Have fun!

3.1 Design

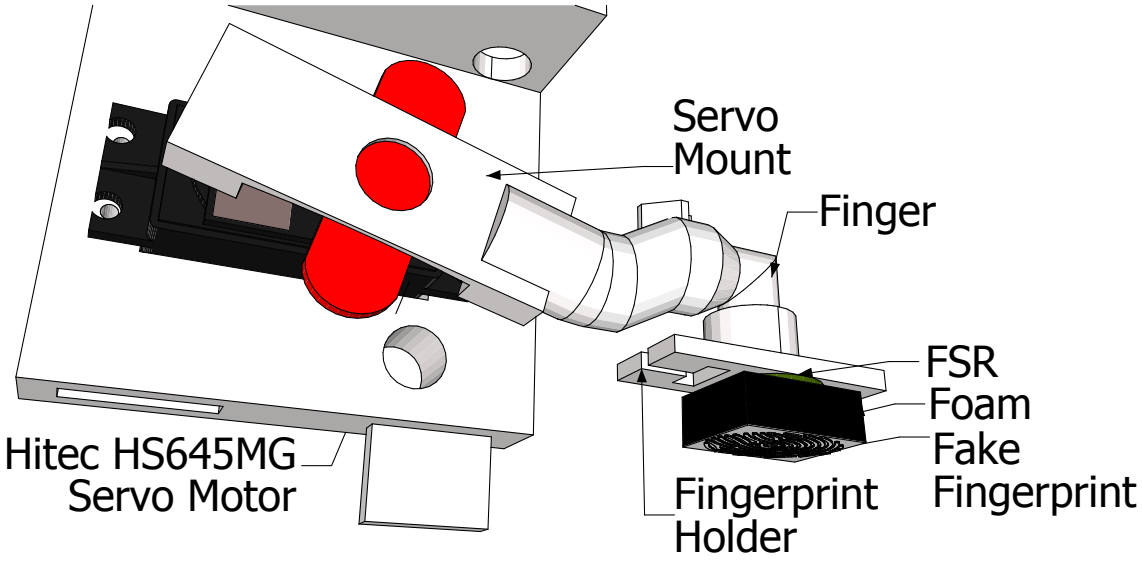
Here is an image showing the main parts of the robot:



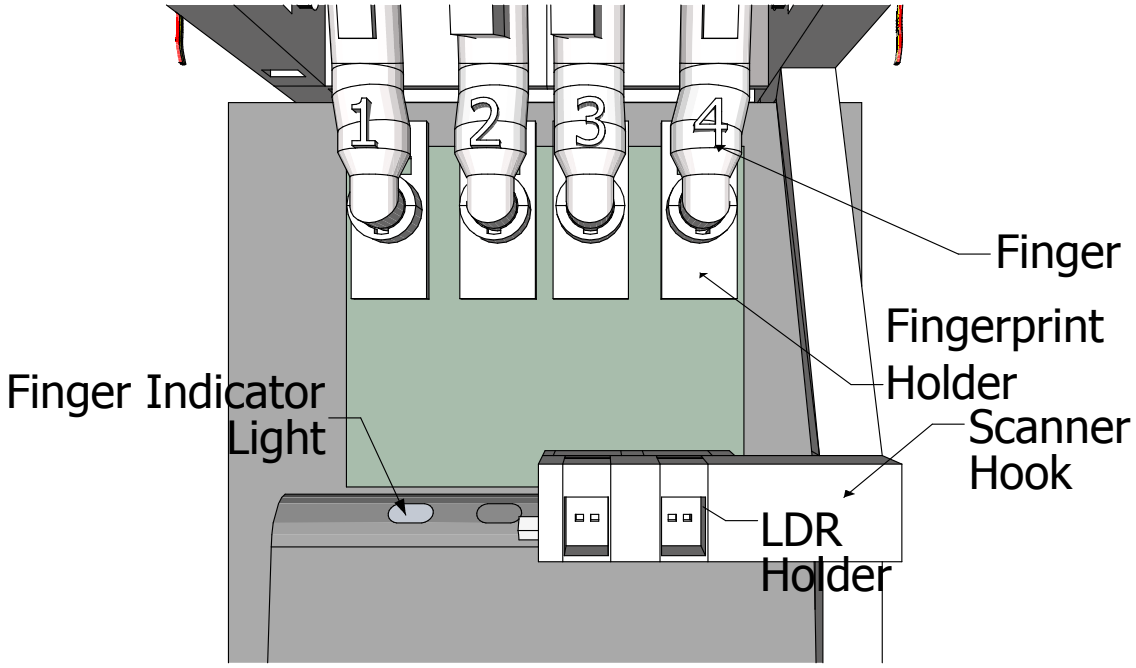
Here is an exploded view of the robot, which is helpful for assembly:



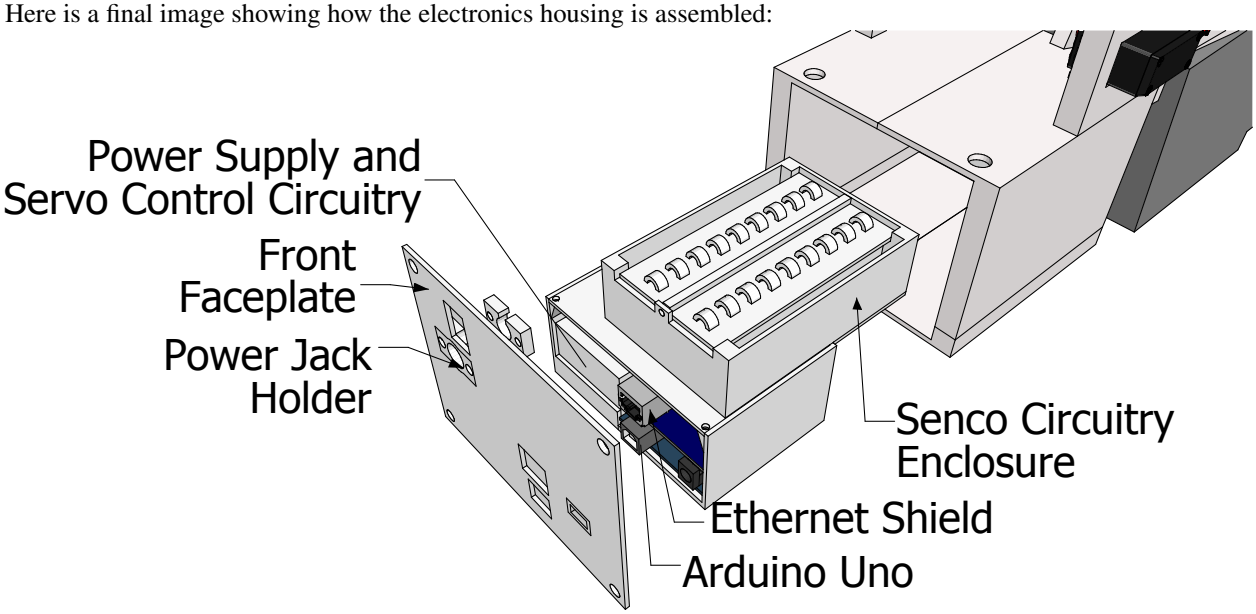
Here is an image showing how the finger attaches to the servo and how the fingerprint attaches to the finger:



Here is an image showing how the robot rests on the fingerprint scanner:



Here is a final image showing how the electronics housing is assembled:



3.2 Construction

To construct the robot, first print appropriate quantities of the parts in `Final STL Files/NAFSTR`. Then, attach all the parts together and insert all the electronics. Here is a parts list of all the required parts that are not 3D printed:

Table 3.1: Parts of NAFSTR

Part Name	URL	Quantity	Price	Total Price
Photoresistor GL5537 5537	http://www.amazon.com/gp/product/B008QVE9BG/ref=ox_sc_act_title_1?ie=UTF8&ps	4	0.24	0.99
1/2" Circular Force Sensing Resistor	http://www.jameco.com/webapp/wcs/stores/servlet/Product_10001_10001_2128260_-1	4	7.95	31.8
Hitech HS645MG Servo Motor	http://www.amazon.com/Hitec-32645S-HS-645MG-Torque-Metal/dp/B003T6RSVQ	4	49.99	199.96
Arduino Uno	https://www.sparkfun.com/products/11021	1	24.95	24.95
Arduino Ethernet Shield	https://www.sparkfun.com/products/9026	1	45.95	45.95
Rocker Switch On Off Single Pole Single Throw Quick Connect Rocker 10 Amp 250 Volt AC Straight	http://www.jameco.com/webapp/wcs/stores/servlet/Product_10001_10001_316128_-1	1	1.65	1.65
DC Power Pigtail Female Plug	http://www.monoprice.com/Product?seq=1&format=2&p_id=4888&CAWELAID=1329 LD8kTESJADsBMxSlZYFRPGB2LwI09oGs0PsjcrUS6Nd1nQFYJR-kuWp-BoCAAjw_wcB	1	0.48	0.48
5V 3A UBEC	http://www.adafruit.com/products/1385	5	9.95	49.75
Multiplexer CD74HC4067E	http://www.digikey.com/product-detail/en/CD74HC4067E/296-33087-5-ND/1507234	1	2.33	2.33
1 kΩ Resistor	http://www.jameco.com/webapp/wcs/stores/servlet/Product_10001_10001_690865_-1	15	0.099	1.485
Jumper Wires		100		0
Solid Core Wire (22 Gauge)				0
Mini Breadboard	https://www.sparkfun.com/products/12043	2	3.95	7.9
Half Breadboard	https://www.sparkfun.com/products/12002	1	4.95	4.95
10 μF Capacitor	https://www.sparkfun.com/products/523	1	0.45	0.45
10x Single Row Male and Female 40 Pin Header Strip 2.54mm / Square Pin Type, Single Row Pin Header Strips, Great Components for PCB	http://www.amazon.com/Single-Female-Header-2-54mm-Components/dp/B008QUVM4E	1	5.24	5.24
12V 5A Switching Power Supply	https://www.adafruit.com/products/352	1	24.95	24.95
Total:				402.835

3.3 Electronics

To assemble to electronics, look at the Fritzing file located at: `Circuit/NAFSTRCircuit.fzz`.

CHAPTER

FOUR

PROGRAM DEVELOPMENT

There are two programs associated with each robot: (1) a Java program that runs on a computer and controls the robot and (2) a C++ program that runs on the robot. The Java program works with both a Graphical User Interface (GUI) and the command line. With the GUI, the user specifies the actions for the robot to execute, and then connects to the robot and executes the actions. With the command line, the user can supply a text file of commands for the robot to execute, and the program will send the commands to the robot. The C++ program on the robot receives all the commands and executes them. It also responds to any button presses.

4.1 Java Robot Control Program

The Java program has 3 main parts:

1. The back end, which handles all communication with the robot.
2. The GUI, which provides a simple way to control the robot.
3. The command line interface, which lets the user send a list of commands to the robot.

This guide will explain each of the three parts.

4.1.1 Back End

The back end consists of the robot package, which contains four main classes:

1. `Robot`
2. `RobotCom`
3. `RobotInfo`
4. `Command`

`Robot` is an abstract class, and it represents any type of robot, either a NAFSTR or a NARFSTR. It currently has one subclass, `NARFSTR`. The current version of the program does not support the NAFSTR robot; to control a NAFSTR, the older version must be used. `NARFSTR` has one constructor, which takes a `RobotCom` object. A `RobotCom` is a channel for communicating with the robot, and has methods for reading and writing data. `RobotCom` has two subclasses: `SerialRobotCom`, which represents a connection to a robot over a serial port (USB), and `NetworkRobotCom`, which represents a connection to a robot over a network. In order to make an instance of either class, a `RobotInfo` object is required. A `RobotInfo` object describes the type of robot, the location of the robot (an IP address and a serial port), and the robot's MAC address, which is used as an ID.

In order to easily find robots, the `Robot` class provides three static methods:

1. `findSerialRobots()`: finds all the robots connected over USB by checking every USB port.

2. `findNetworkedRobots()`: finds all the robots connected over the network by sending out a UDP broadcast and listening for responses.
3. `findRobots()`: finds all the robots over both USB and the network.

In order to execute commands on a robot, the `Robot` class provides an `executeCommand(Command c)` method, and an `executeCommands(List<Command> commands)` method. Both methods send commands to the robot, and wait for execution to finish. A `Command` object consists of two components: a String representing the command name, and a list of Objects representing the parameters.

For more documentation, see the Javadoc, in `Code/FingerprintRobotControl/doc`.

4.1.2 GUI

The code for the GUI is in three packages: `view`, `view.configurator`, and `view.robotpanel`. The program's GUI is an instance of `view.RobotController`. The `RobotController` class has two main components: a `view.robotpanel.RobotConnectionPanel`, which manages the connection with the robot, and a `view.robotpanel.CommandEditorPanel`, which allows the user to edit the commands being sent to the robot.

4.1.3 Command Line

The code for the command line interface is in `main.FingerprintRobotControl`. When `FingerprintRobotControl` is executed with no arguments, it automatically creates an instance of `view.RobotController`, creating the GUI. If there are arguments, it attempts to use them. The argument format is: `<robot type> <file name> <connection location>`. The robot type should be either NAFSTR of NARFSTR. (NAFSTR is not currently supported, but will be soon.) The file name should be an absolute or relative path to the file of commands which the program is to execute. The file format is a text file with one command per line. Finally, the connection location could be one of four things: "ethernet", "usb", an IP address, or a serial port. If it is "ethernet" or "usb", to program will search for robots connected over the specified channel, and pick the first one.

4.2 C++ Local Control Program

4.2.1 Development Environment

The program was developed in Eclipse using the the Arduino Eclipse Extensions version 2.4.201506210212 by Jan Baeyens (http://www.baeyens.it/eclipse/), with the Arduino libraries version 1.6.5. The Robot's program consists of two parts: C++ libraries which are common to both robots (the libraries are currently only used with the NARFSTR, but should be used on the NAFSTR), and a `.ino` file which contains the main program for the robot. The `FingerprintRobotCom` library is stored in `Code/FingerprintRobotLocalControl/lib`. In order to make the Arduino plugin include this library (and other libraries which are stored in the lib folder) in the build path, the "Private Library Path" in the Arduino Plugin Preferences must be set to `Code/FingerprintRobotLocalControl/lib`.

4.2.2 Libraries

The code needs several libraries in order to compile. First, it needs the `FingperintRobotCom` library, which handles communication between the robot and the computer. Second, it needs the `EthernetNonBlocking` library, which is a modified version of the standard Arduino Ethernet Library. I modified the standard library to make the DHCP functions non-blocking. This significantly improves responsiveness; with the standard blocking DHCP

functions, the robot would freeze for 60 seconds if no Ethernet cable was plugged in. Both programs also need the following standard Arduino libraries:

1. `EEPROM`
2. `SPI`

Each of the NAFSTR and NARFSTR programs also require some specific libraries:

NAFSTR

1. `Servo`: standard Arduino library.

NARFSTR

1. `Arduino-RGB-Tools`: an open-source library for controlling RGB LEDs. It is on github (https://github.com/joushx/Arduino-RGB-Tools), stored in the git repository as a submodule, and licensed under The MIT License (MIT).
2. `RGBConverter`: an open-source library for converting between RGB and HSV. It is on github (https://github.com/ratkins/RGBConverter), stored in the git repository as a submodule, and its license says that you can do whatever you want with it.
3. `VNH5019Driver`: a library I wrote for controlling the VNH5019 driver chip.

4.2.3 NARFSTR

The NARFSTR control program is located at `Code/FingerprintRobotLocalControl/NARFSTRLocalControl`. Its main file is `NARFSTRLocalControl.ino`.

4.2.4 NAFSTR

The NAFSTR control program is located at `Code/FingerprintRobotLocalControl/NAFSTRLocalControl`. Its main file is `NAFSTRLocalControl.ino`. The NAFSTR program is old, and so does not use either the `FingerprintRobotCom` library or the `EthernetNonBlocking` library.

4.2.5 Development Information

The `FingerpintRobotCom` library attempts to read a MAC address from the Arduino's EEPROM on startup. If no MAC address is found, it will not work. In order to load a MAC address into EEPROM for the first time, use the `MACUploader` program located at `Code/FingerprintRobotLocalControl/MACUploader`.

`FingerprintRobotCom` Library

The `FingerprintRobotCom` library header file (located at `Code/FingerprintRobotLocalControl/lib/Fingerprint` contains comments which explain all the important functions.

CHAPTER

FIVE

USING THE ROBOTS

The robots are controlled by text commands send to them over either a serial connection, a TCP network connection, or a UDP network connection. These text commands can be sent in many ways. They can be sent through the Java control software, using both the command line interface and the GUI, they can be sent using Telnet or Putty, and they can be sent using serial terminal, such as gtkterm. This guide will explain the command format, and how to send commands using all the different channels.

5.1 Communication Protocol

5.1.1 Protocol

General Information

1. The robot will receive input from a serial stream via USB or over a network connection on port 2424 (the NAFSTR uses port 80 because its software is old) via Ethernet. The format of the requests will be the same in all cases.
2. When the robot is connected over the network, it is capable of receiving requests over both a UDP connection and a TCP connection. However, if any request is received over a UDP connection, the response message will be sent over a TCP connection.
3. With the exception of UDP messages as mentioned above, the robot will respond to all messages over the channel the message was received on.
4. The robot only has a 64 byte serial buffer. Thus, if many commands are sent rapidly over USB, some commands will be lost. However, when connected over Ethernet, the robot has a 16 kibibyte buffer, thus making an overflow less likely.

Format

1. Each command has the format `commandName arg_0 arg_1 arg_2...`, where `arg_n` is the n^{th} argument. If the command has no arguments, then the command is simply the name. Each part of the command is separated from the others with a single space.
2. Upon receiving and successfully parsing a command, the robot will send a message of the format `commandName-received`. If there is a problem parsing a command, the robot will instead send `bad-command`.
3. Upon completion of a command, the robot will send a message of the format `commandName-end`.

4. When sent the string "fingerrobot", the robot will respond with a string of the format: `found:robot type:MAC Address:`. For example, a NARFSTR robot with a MAC address of `ba:db:ad:ba:db:ad` would respond with `found:NARFSTR:ba:db:ad:ba:db:ad:`.

5.1.2 Commands

NARFSTR Commands

Stroke

Description causes the robot to move the finger through one cycle.

Format `stroke`

Set

Description sets the parameters of the robot's following moves.

Format `set forwardSpeed buttonWaitTime reverseSpeed returnWaitTime`

Parameters

`forwardSpeed` the motor's speed when driving the finger away from the starting position. Between 0 and 255, inclusive.

`buttonWaitTime` the amount of time, in ms, the robot should wait once the finger hits the limit switch.

`reverseSpeed` the motor's speed when driving the finger back to the starting position. Between 0 and 255, inclusive.

`returnWaitTime` the amount of time to wait, in ms, once the finger has returned to its original position.

Reset

Description causes the robot the drive the finger towards the starting position.

Format `reset time speed`

Parameters

`time` the amount of time the finger will be in motion, in ms.

`speed` the motor's speed, between 0 and 255 inclusive.

NAFSTR Commands

See `DocumentationOld/Arduino - Computer Communication Protocol.docx` or `DocumentationOld/Arduino - Computer Communication Protocol.txt` for details.

5.1.3 Examples

The following are examples of what an exchange with the robot would look like.

NARFSTR Example

```
fingerrobot
found:NARFSTR:90:a2:da:0f:95:39:
set 120 2000 120 2000
set-received
set-end
stroke
stroke-received
stroke-end
reset 1000 255
reset-received
reset-end
notacommand
bad-command
```

NAFSTR Example

Please put one here!

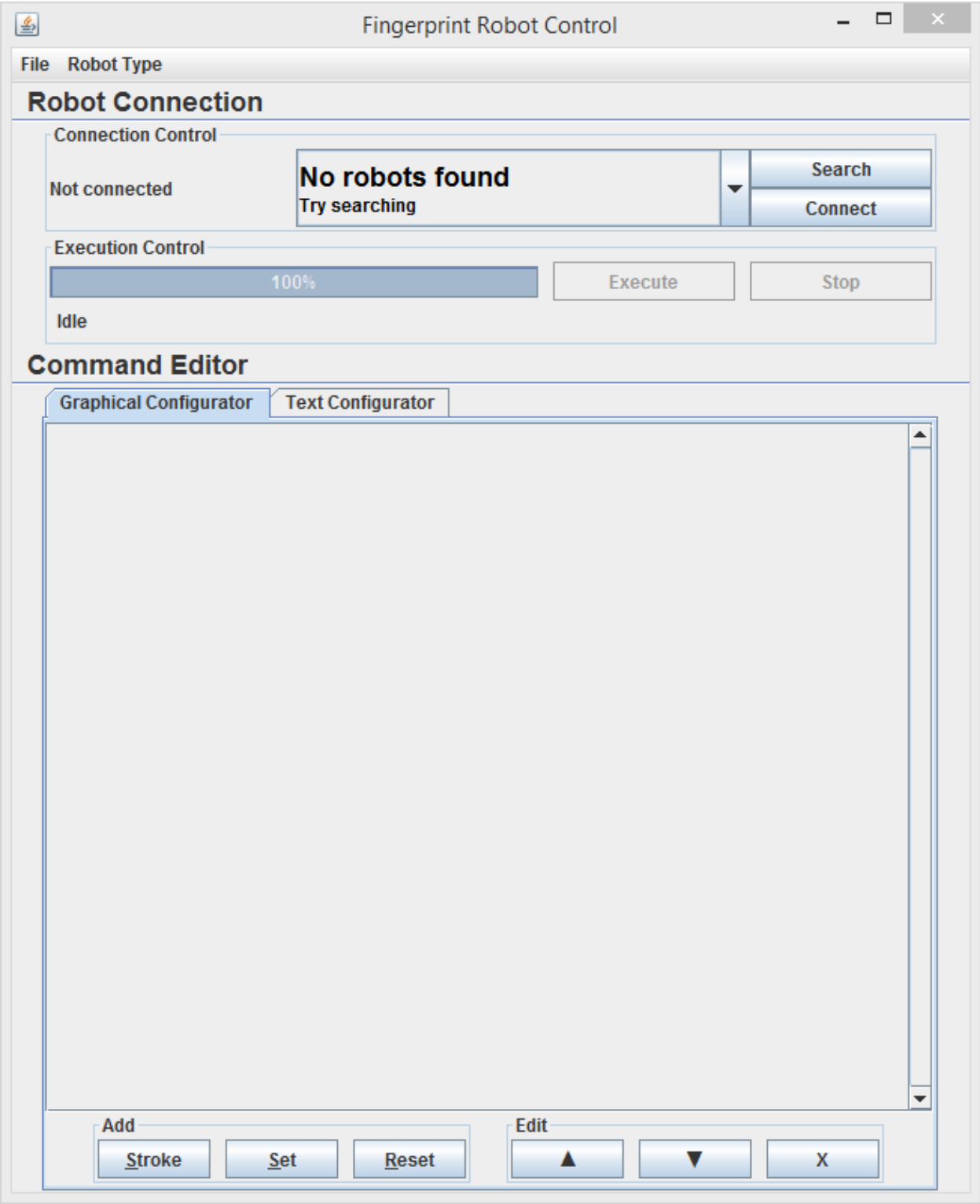
5.2 Control with `FingerpintRobotControl` Java Program

The program can be found in `Code/FingerprintRobotControl/FingerprintRobotControl-v1.1.jar` and allows for both GUI and command line operation. The following two sections describe both methods.

Note: the NAFSTR uses an older version of the software, found in `Code/FingerprintRobotControl/FingerprintRobotControl-v1.0.jar`. Most, but not all, of the instructions below apply to the older version.

5.2.1 GUI Control

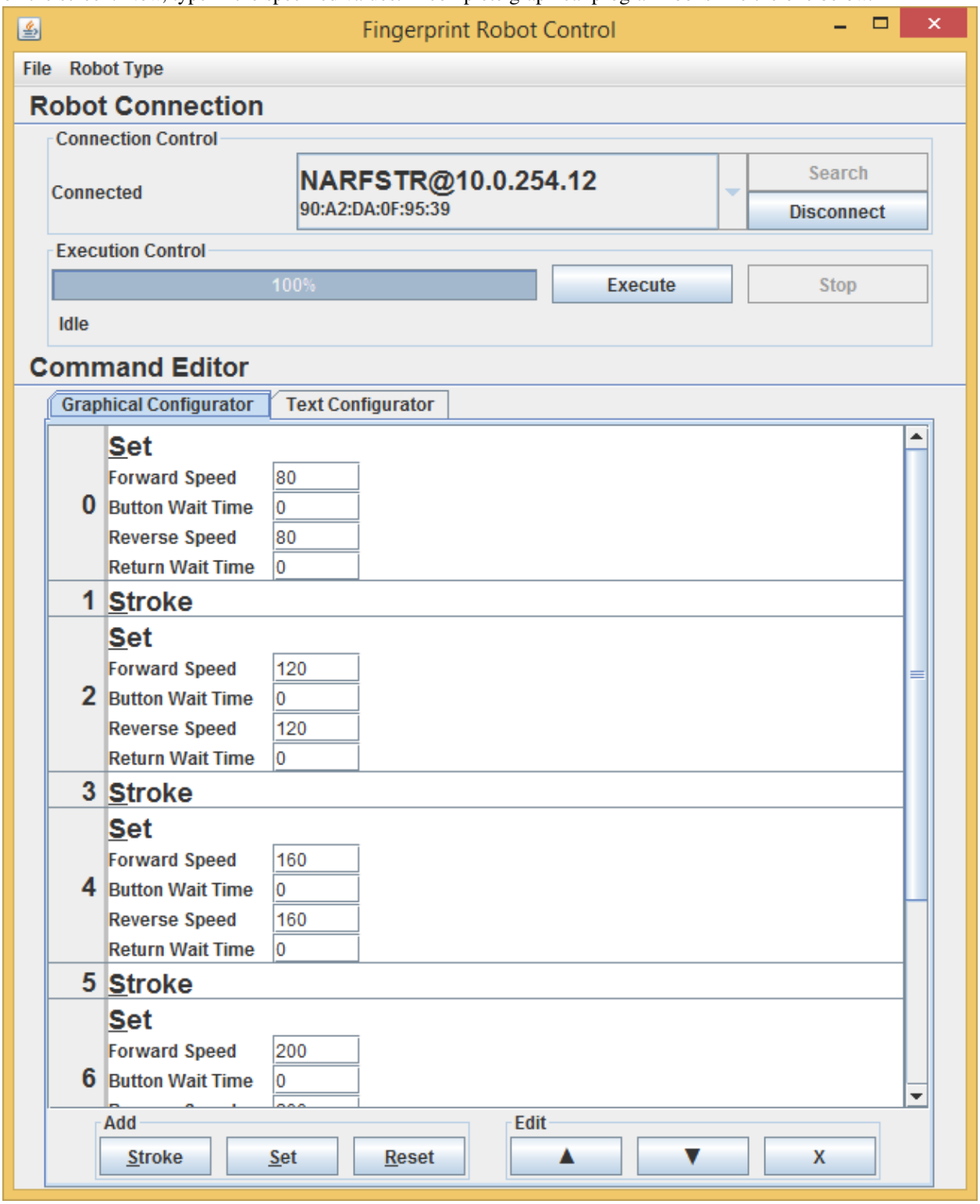
To launch the program in GUI mode, simply run the program. Or, launch the program from the command line with no arguments: `java -jar FingerprintRobotControl-v1.1.jar`. Once the program opens, you should see a window like the one below.



This is the starting screen. To use the program, first search for robots by clicking the "Search" button. After about three seconds, the combo box will be populated with the types, locations, and MAC addresses of all the robots the program found. To connect to a robot, select one from the list and press connect. If the connection was successful, the status field to the left of the combo box should say "Connected".

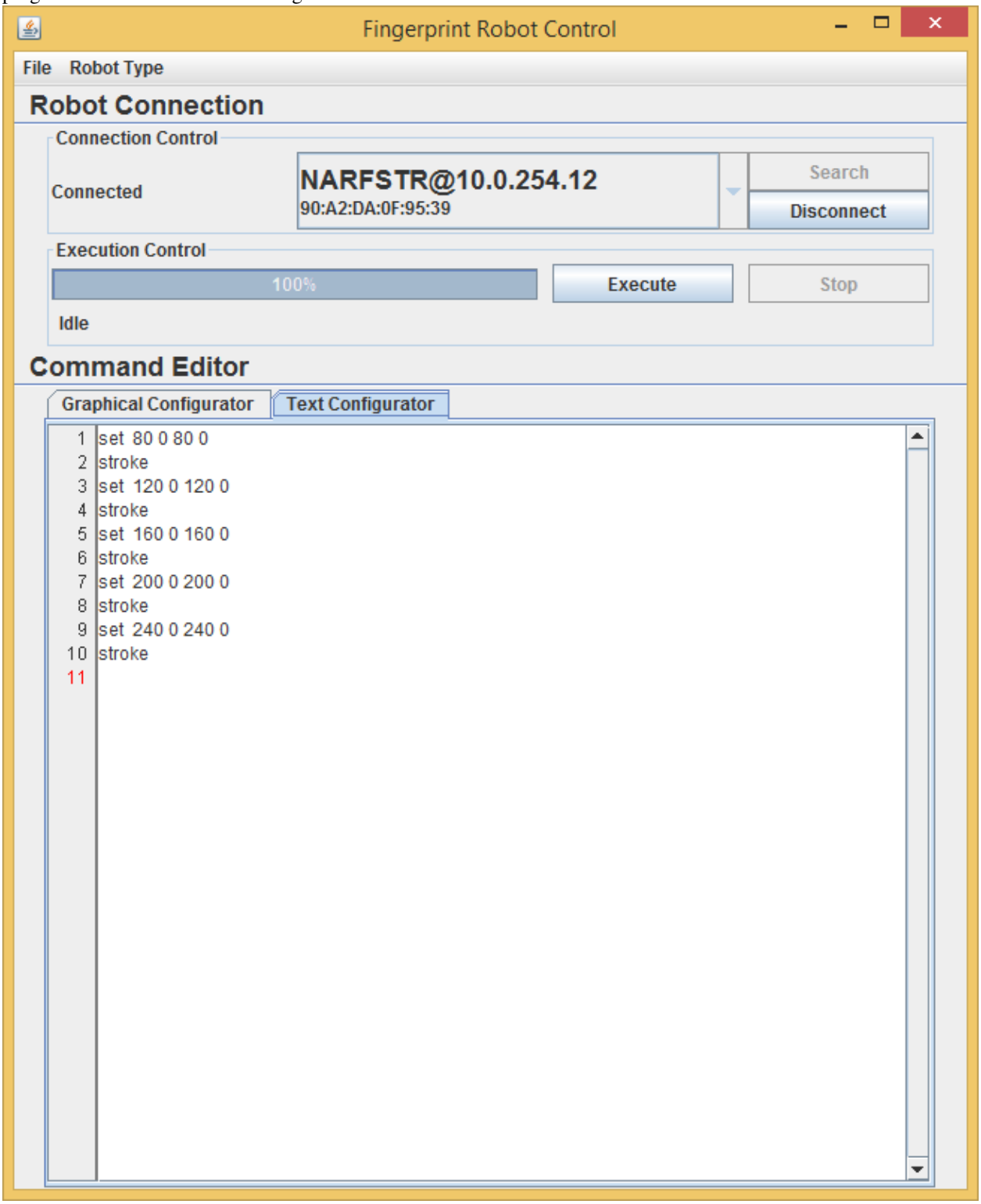
Now, you can begin to write the commands you want the robot to execute. There are two ways to do this: (1) using the

graphical configurator, which requires no knowledge of the communication protocol or, (2), using the text configurator, which allows you to simply type the commands you wish to send to the robot. To use the graphical configurator, press the button on the bottom of the screen with the name of the command you wish to add. A command should show up on the screen. Now, type in the specified values. A complete graphical program looks like the one below.



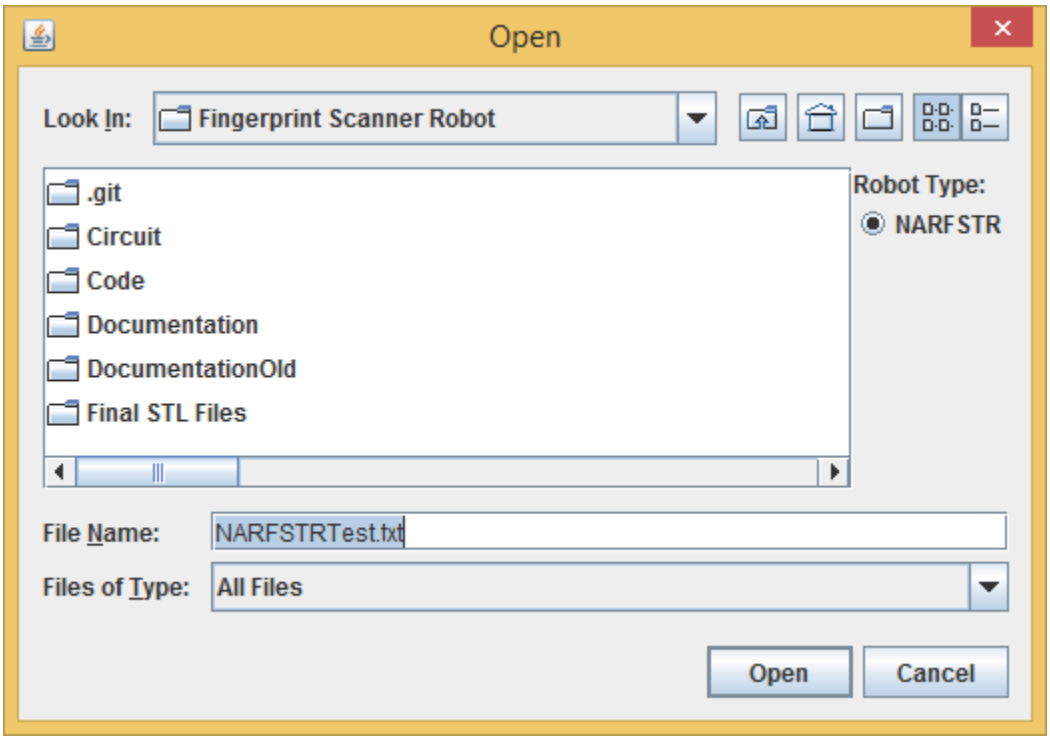
To use the text configurator, select the tab labeled “Text Configurator”. If you have a program currently open if the

graphical configurator, it will automatically be converted to text and displayed in the text configurator. This is a good way to learn the text format of the commands. Once you are in the text configurator, simply type commands. A program written with the text configurator is shown below:



Once you are done with you program, you can save it by going to “File” menu and clicking “Save”. This will bring up a dialog asking you where you would like to save your file. The saved files are plain text files, with the same content

that appears in the text configurator. To open a file, choose "Open" from the "File" menu. When you open a file, there will be radio buttons on the right side of the open dialog to allow you to choose the type of robot the opened file is for. (Currently, there is only one button for the NARFSTR. Support will be added for the NAFSTR in the future.) The picture below shows the open dialog.



To execute your programs, hit the "Execute" button. This will send the commands to the robot. You can see the progress in the progress bar. If you want to stop execution, hit the "Stop" button. This will stop execution after the currently running move is complete. There is no way to stop execution in the middle of a move. If you click the stop button, and the program still says "Stopping" under the progress bar even though the robot has stopped, you may need to click the "Force Stop" button. This will immediately disconnect the robot.

5.2.2 Command Line Control

To run the program from the command line, use the following command:

```
java -jar FingerprintRobotControl-v1.1.jar robotType fileName connectionMode
```

Parameters

`robotType` the type of robot to control, either NAFSTR or NARFSTR. (Currently only NARFSTR is supported.)

`fileName` an absolute or relative path to a file containing the commands to execute, 1 command per line.

`connectionMode`

`ethernet` searches for robots on the network and connects to the first one found.

`usb` searches for robots connected over USB and connects to the first one found.

IP address the IP address of the robot to connect to.

Serial port the name of a serial port the robot is plugged into.

5.3 Control Using Telnet, Putty, and other Terminals

The robot can easily be controlled using any terminal. With telnet, simply type `telnet ipAddress 2424`, where `ipAddress` is the IP address of your robot. With Putty, enter the IP address of your robot and the port 2424. Then, select telnet for the protocol and hit connect. For gtkterm, launch the terminal and select the right serial port. Once you have connected your terminal, just type commands, 1 per line, and the robot should execute them.