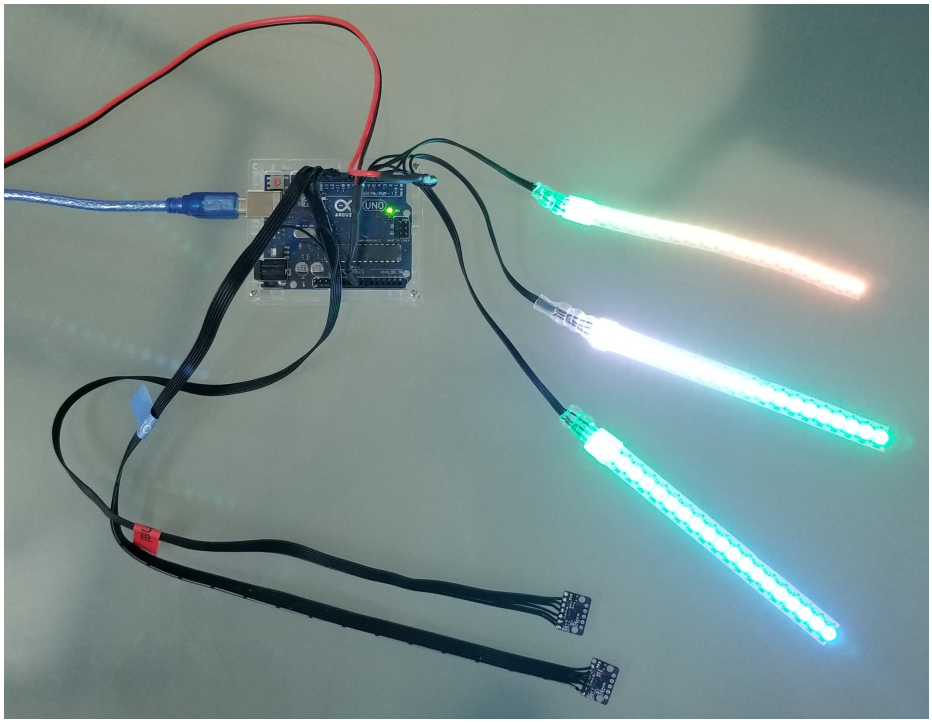
Elijah Specification

Joseph Rubin
josephmeirrubin@gmail.com

August 2, 2018

sixdof.space
ULTRA FAST OPTICAL TRACKING

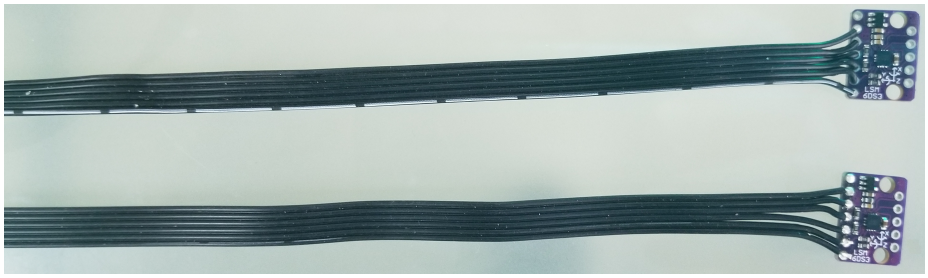


Contents

1 User Guide

1.1 Overview and Purpose

Elijah is a swallowing monitor that uses micro-electromechanical systems to observe the quality of infant or child ingestion. The device has two sensors that are placed on the outside of the patient's body. The first sensor is positioned beneath the jaw to track the motion of the tongue. The second sensor is placed on the front of the throat in the center of both axes, and its purpose is to measure movement of the throat while a bolus travels through the esophagus. The device operates during the pharyngeal and esophageal stages of swallowing while the patient is consuming solid, semi-solid, or liquid food.



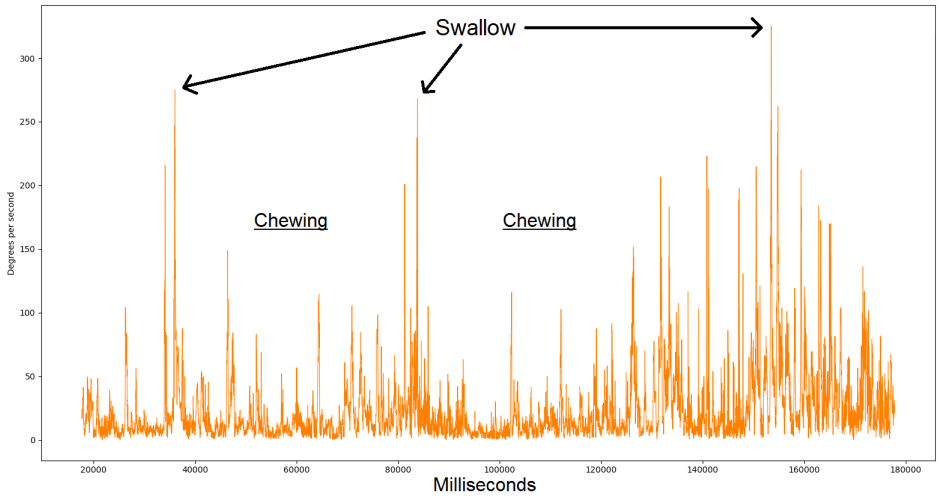
Existing procedures to measure swallowing health are lacking, either in convenience, ease-of-operation, or in the quality of information produced. In an X-ray imaging test such as a fluoroscopy, the patient is asked to swallow a barium solution or food coated in barium. Barium appears on an X-ray with high contrast, allowing its motion to be tracked, but this procedure may not be used freely because of the radiation involved and the specialists that must be present during the test.

An endoscopy necessitates instruments being used inside the patient's throat. This can be uncomfortable, and it is generally reserved for older children. Tools that are used exterior to the body such as stethoscopes and external cameras do not provide information of as much precision. Elijah aims to be unobtrusive to the patient while providing detailed information to medical professionals that they can use to diagnose dysphagia.

1.2 Technology and Architecture

The current version of Elijah has three main parts: sensors, the transmitter, and the receiver. The transmitter is responsible for capturing sensor data and providing it to the receiver, while the receiver's role is to analyze the data and provide a useful output.

The transmitter device is an Arduino Uno, and the receiver is a Lenovo laptop which runs Windows 7 and Python 3.6.2. The Arduino collects data



Output from the receiver overlaid with analysis.

from two MEMS sensors which each have a gyroscope and an accelerometer. While the main function of the Arduino is to collect data from the sensors and provide it to the laptop for analysis, three LED strips connected to the Arduino show a limited form of output. Presently, a connection with the laptop is necessary in order to collect and view data (the Arduino cannot act alone).

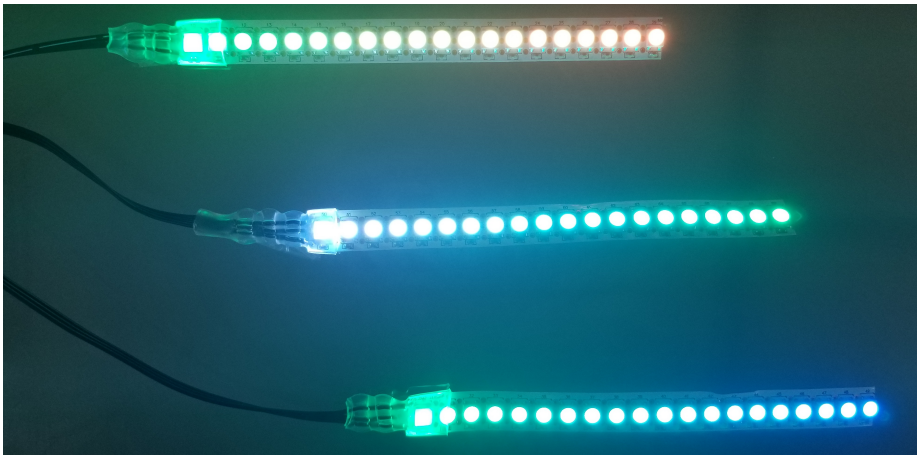
In the future, Elijah may use a single micro-controller with enough computational resources to both capture *and* analyze data. For output, this device could use an OLED screen, e-mail test results to doctors and technicians, or connect directly to a patient record system.

1.2.1 Input and Output Devices

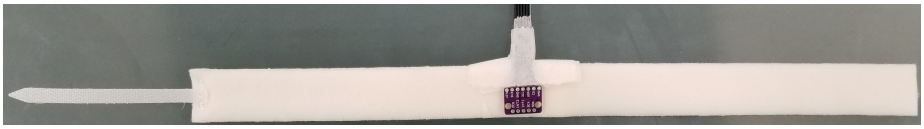
The MEMS sensor in use for Elijah is the LSM6DS3 iNEMO inertial module.[1] This device has a gyroscope and an accelerometer. The LED strips are the WS2812 Intelligent control LED integrated light source.[2] The following table outlines the purpose of each device that is connected to the Arduino:

[1] https://www.st.com/en/mems-and-sensors/lsm6ds3.html
[2] https://cdn-shop.adafruit.com/datasheets/WS2812.pdf

Device	**Purpose**
Magnitude LED strip (20 LEDs).	Display a measurement of the average strength of patient swallowing.
Frequency LED strip (20 LEDs).	Display a measurement of the general amount of time in between peaks in the gyroscope data.
Blindspot LED strip (20 LEDs).	Display a measurement of the prevalence of areas of low readings in the gyroscope data.
Limit switch	Click during a data capture to mark a location of particular interest. After the capture is done, the clicks may be shown as vertical lines on a plot. The limit switch may be disconnected from the Arduino when not desired.
Tongue MEMS	Monitor tongue movement during a swallow. This sensor is placed under the jaw. In instances where the child seems unwilling to wearing both sensors, the tongue sensor may be left unused.
Throat MEMS	Monitor throat movement during a swallow. This sensor is placed on the front of the neck (centered on both axes).



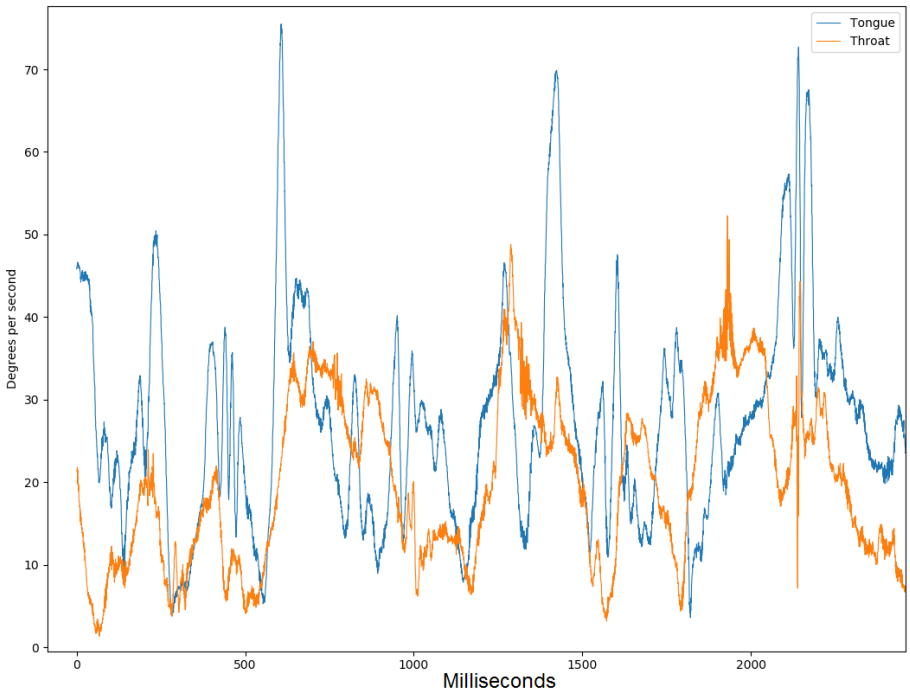
In an effort to make the system as comfortable as possible, we have explored different methods of attaching the sensors to the patient. Using an adhesive such as medical bandages or tape has sometimes proven to irritate the children, especially as they swallow. The current system uses a feeding tube holding band for the throat sensor which straps around the neck and locks to itself. A single sensor may be taped to the inside of the band where it will rest against the child's neck.



1.3 The Graphical User Interface

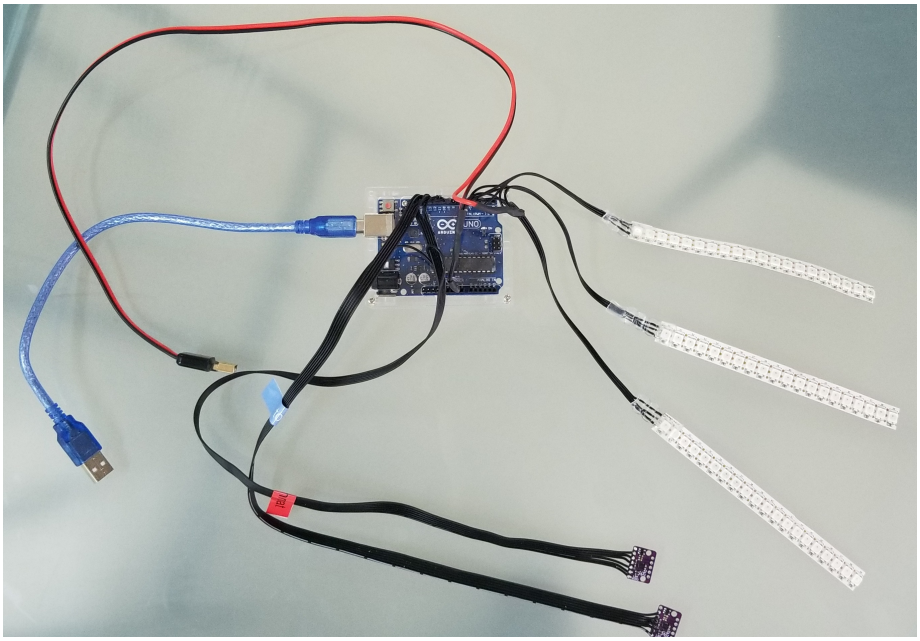
The `recv/dist/Elijah.exe` file launches a graphical window that simplifies capturing, processing, and visualizing the data. Make sure that Elijah is connected to the computer via USB, then double click the file to launch it. If the device is not connected, a dialog box will show prompting you to connect the device. Simply plug it in and click "Retry." Press "Start" to begin a capture and "Stop" to name and save it.

The left of the screen shows all of the saved captures. If there are many, use the scrollbar to advance to the one you would like to interact with. Click on the red 'X' next to a capture to delete it. Click on the capture's name to view its plot. The plot shows time in milliseconds on the x axis and the magnitude of the rotational motion of the sensors in degrees per second on the y axis.



In this example, the legend in the top-right corner tells us that the blue line represents the sensor that was placed beneath the jaw to measure tongue motion, while the orange line follows the motion of the throat sensor. The peaks in the graph represent times where the sensors detected relatively high amounts of motion. Generally, those areas are associated with a strong swallow.

2 Hardware



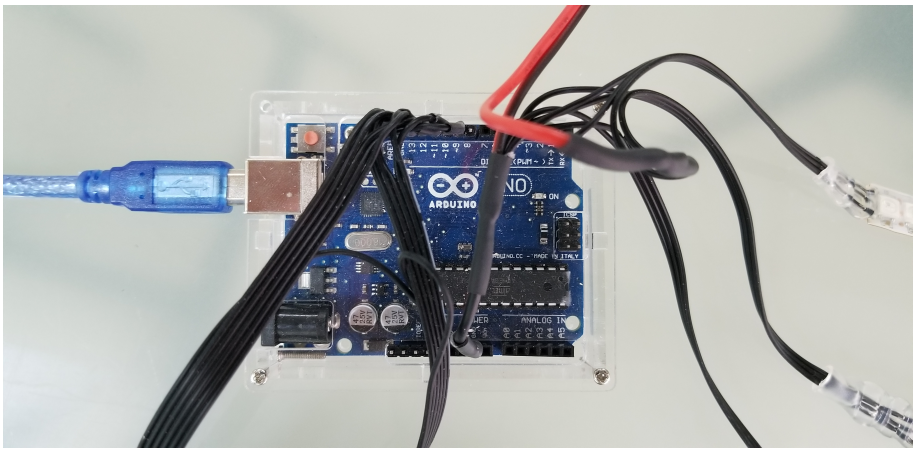
2.1 Pins

The following tables show which input and output pins are connected to each peripheral, as well as which pin mode is set on the Arduino. The pins listed in the first table are listed and modifiable in `trans/pins.h` while those in the second table are specified by the Arduino SPI library and thus cannot be changed.

Device	**Function**	**Pin**	**Arduino Mode**
Magnitude LED strip (20 LEDs)	Data	`3`	Output
Frequency LED strip (20 LEDs)	Data	`4`	Output
Blindspot LED strip (20 LEDs)	Data	`5`	Output
Limit switch[3]	Data	`7`	Input
Tongue MEMS	Slave select	`9`	Output
Throat MEMS	Slave select	`10`	Output

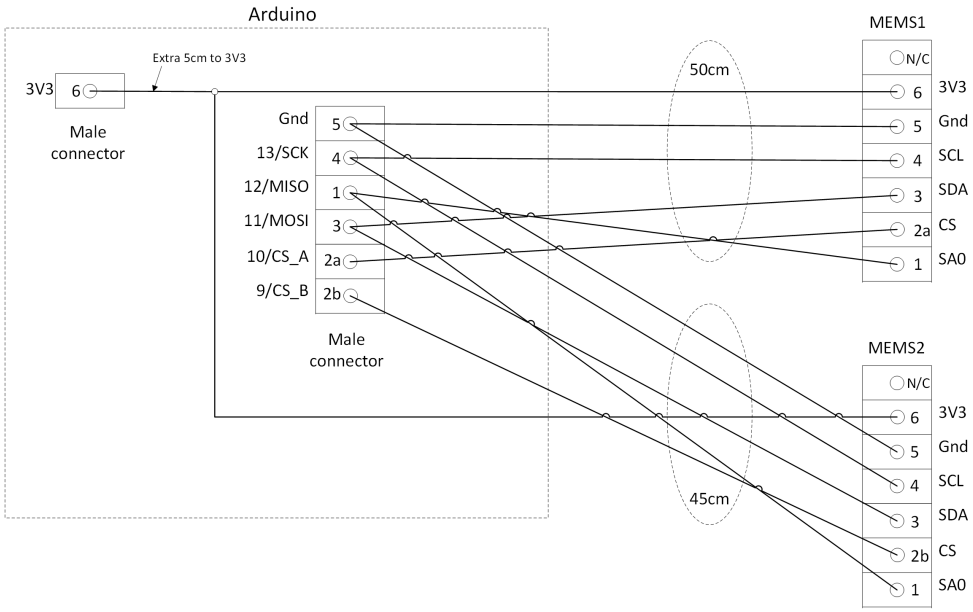
Device	**Function**	**Pin**	**Arduino Mode**
Tongue + throat MEMS	Master out slave in	`11`	Output
Tongue + throat MEMS	Master in slave out	`12`	Input
Tongue + throat MEMS	Serial clock	`13`	Output

[3] Clicking the limit switch during a capture will show vertical lines when it is plotted. This is useful for marking off periods in the capture for later analysis.



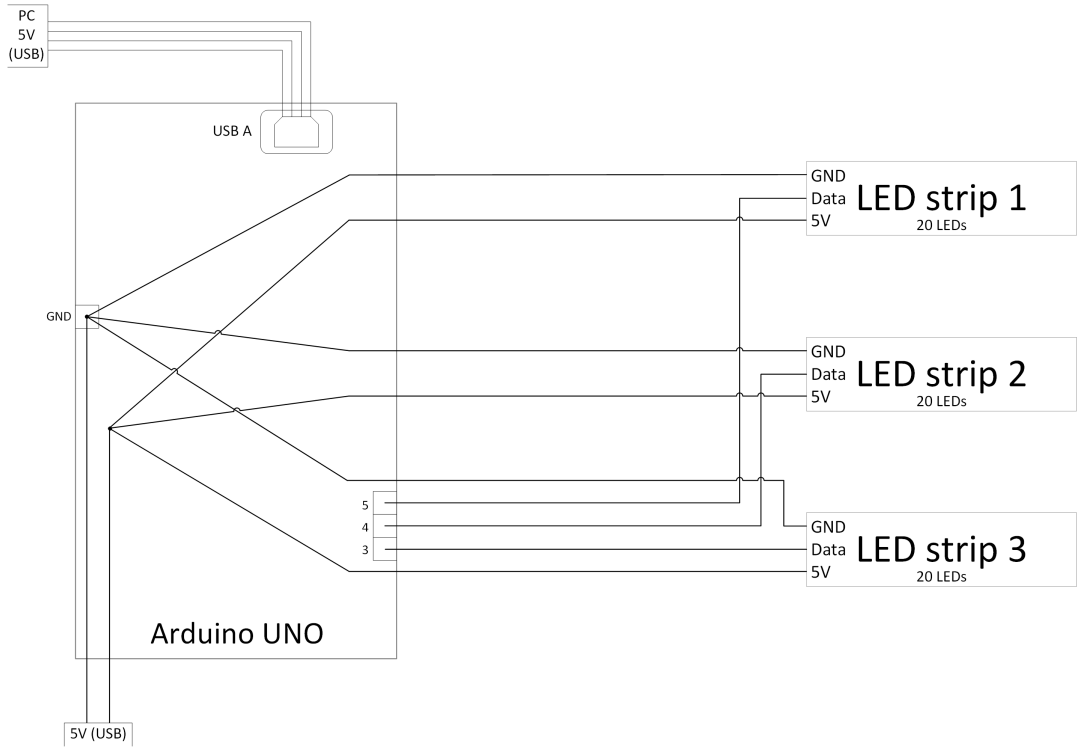
2.2 SPI and MEMS

Interfacing with the LSM6DS3 MEMS devices via SPI uses 12 wires in total. The same `3V3` and `GND` lines are split among the two devices, as are `MISO` (master in, slave out), `MOSI` (master out, slave in) and `SCK` (serial clock). The final two connections are the `SS` (slave select) wires, each MEMS having one of its own. The MEMS are rated at 3.3V for these signals. Since we are using 5V, there are concerns that we may be damaging the sensors, or establishing sub-par communication, but no negative consequences have been encountered.



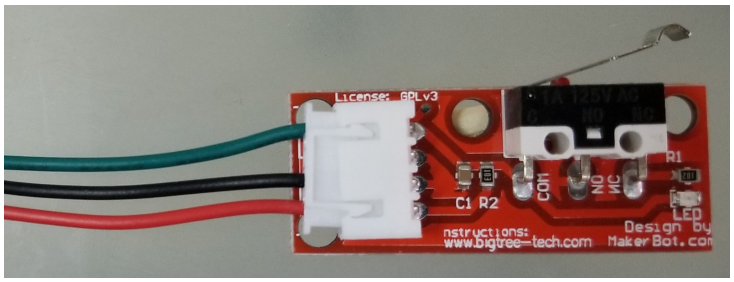
2.3 LED Strips

The three LED strips have 20 LEDs each, and they share a `GND` line to the Arduino. They also share a connection to a `5V` plug through the `GND`. The plug is a 5V1.0A power port of a USB power supply (higher ratings will work as well). Each one has a unique data wire which connects to an Arduino pin.



2.4 Limit Switch

The limit switch connects to the Arduino at `5V` and `GND`. It also has a data wire to the board. The limit switch may be disconnected when it is not desired.



3 Communications Protocol

The transmitter communicates over serial with the receiver using an A-B USB cable at a baud rate of 1,000,000 bits per second. The receiver must have the Arduino serial driver installed.[4] The receiver program is configured to automatically detect the serial port that is assigned to the Arduino Uno on Windows 7 as long as no other Arduino is connected to the computer at the same time. Numerical data is always transmitted **little-endian**, and signed values are transmitted in **two's complement** representation.

3.1 Capture Data Transmission Format

The main component of communication between the transmitter and the receiver is transmission of the capture data from the MEMS sensors. Care must be taken to ensure that the receiver can understand the readings. The following specification details how a capture is transmitted.

3.1.1 Synchronization

Start of data transmission is marked by `SIG_HEAD`. See the following section on high level "Signals" for details.

3.1.2 Configuration

Immediately after `Synchronization`, with no extra bytes or newlines in between, information regarding the configuration of the transmitter will be written. The configuration details are the MEMS capture rate in hertz (one rate which is shared by both the gyroscope and accelerometer), the full-scale range of the gyroscope in degrees per second, and the full scale range of the accelerometer in g's. These values are common to both the tongue and throat sensors. This data will be sent in the following format:

1. `capture rate` – 16 bit, unsigned. The capture rate in hertz that the MEMS devices are polled at.

2. `gyro full-scale` – 16 bit, unsigned. The magnitude of the positive half of the range of values that the gyroscope can read, in degrees per second, under the assumption that the negative range of values is of exactly the same magnitude. A value of 2000 therefore means that the gyroscope's output range of 16 bits spans a scale of -2000 to $+2000$ degrees per second.

3. `accl full-scale` – 8 bit, unsigned. The magnitude of the positive half of the range of values that the gyroscope can read, in g's, under the assumption that the negative range of values is of exactly the same magnitude. A value of 4 therefore means that the accelerometer's output range of 16 bits spans a scale of -4 g to $+4$ g.

[4] https://www.arduino.cc/en/Guide/Windows

3.1.3 Frames

Immediately after `Configuration`, with no extra bytes or newlines in between, the capture frames are written. They each represent a single round of polling a particular sensor (tongue/throat), and are represented in the following format:

1. `time` – 16 bit, unsigned. The number of milliseconds past the start of the capture that this particular frame was gathered.
2. `gyro_x` – 16 bit, signed. The x-axis output from the gyroscope.
3. `gyro_y` – 16 bit, signed. The y-axis output from the gyroscope.
4. `gyro_z` – 16 bit, signed. The z-axis output from the gyroscope.
5. `accl_x` – 16 bit, signed. The x-axis output from the accelerometer.
6. `accl_y` – 16 bit, signed. The y-axis output from the accelerometer.
7. `accl_z` – 16 bit, signed. The z-axis output from the accelerometer.
8. `flag` – 8 bit, unsigned. A series of bits which represent various parameters. From right to left:

   [0] `end` – a 1 indicates that this is the last frame that will be transmitted. The receiver should ignore `time` and sensor outputs from this frame, as well as any flag bits other than `end`, but `checksum` will be generated as normal so it should be resolved.

   [1-2] `sensor id` – a unique id for each sensor that identifies the source of this frame.

   00 – Tongue MEMS

   01 – Throat MEMS

   [3-6] unassigned – in the future, these bits may be used in conjunction with the preceding bits to express more sensor id values.

   [7] `button` – a 1 indicates that the limit switch was clicked during this frame. If the limit switch is disconnected from the transmitter, the value of this bit is undefined.

9. `checksum` - 8 bit, unsigned. Calculated as the exclusive or (XOR) of every raw octet in this frame other than this one.

There is no delimiter between frames. Each one is precisely 16 bytes, after which the next one begins.

3.1.4 Trailer

Following `Frames` may be any number of bytes sent from the transmitter. This section should not contain information that changes the resolution, scale, or meaning of any preceding data. The contents of this section should not be necessary for decoding the previous sections of the transmission, but it may be used for data analysis or debug output. A single `\x2E` (dot) `.` on a line of its own (terminated by either `CRLF` or `LF`) signals the conclusion of `Trailer` and the end of meaningful transmission. Even if the transmitter has nothing to say in `Trailer`, it will still write the dot and end the line.

3.1.5 Sample Frame Explained

During `Frames`, the following data was received from the transmitter:
`\x05\x00\x13\x01\xBE\xFF\xF1\xFE\xAC\x40\x80\x00\xA4\x02\x80\x11`

To understand this frame we have to break down these 16 bytes into component parts.

	Octet	Significance[5]	Meaning
`time`	`\x05` `\x00`	LSB MSB	This frame was captured at 5ms after the start.
`gyro_x`	`\x13` `\x01`	LSB MSB	The sensor's x axis gyroscope read 275.
`gyro_y`	`\xBE` `\xFF`	LSB MSB	The sensor's y axis gyroscope read -66.
`gyro_z`	`\xF1` `\xFE`	LSB MSB	The sensor's z axis gyroscope read -271.
`accl_x`	`\xAC` `\x40`	LSB MSB	The sensor's x axis accelerometer read 16556.
`accl_y`	`\x80` `\x00`	LSB MSB	The sensor's y axis accelerometer read 128.
`accl_z`	`\xA4` `\x02`	LSB MSB	The sensor's z axis accelerometer read 676.
`flag`	`\x82`		0 this is not the last frame. 1 0 this frame came from the throat MEMS. 0 no meaning (unassigned). 0 no meaning (unassigned). 0 no meaning (unassigned). 0 no meaning (unassigned). 1 the limit switch was clicked during this frame.
`checksum`	`\x11`		Since taking the XOR of every octet in this frame results in zero, this checksum is successfully resolved.

3.2 Signals

Reliable communication between transmitter and receiver is accomplished through the use of a handful of signals. Signals are a single unique byte or group of bytes that may be written to serial output to convey a specific meaning.

[5] Data is sent little-endian, so the least significant byte comes first.

3.2.1 Signal Definitions

Name	Value[6]	Meaning
SIG_READY	\x24\x25\x26	The transmitter is on and ready for a SIG_REQUEST.
SIG_REQUEST	\x3C	The receiver wants the transmitter to start a capture.
SIG_ENOUGH	\x3E	The receiver requires no further capture data.
SIG_HEAD	\x22	The transmitter is about to follow with capture data.
SIG_DENIED	\x21	The transmitter will not fulfill a SIG_REQUEST.

3.2.2 Signal Usage

SIG_READY, SIG_HEAD and SIG_DENIED are sent from the transmitter, while SIG_REQUEST and SIG_ENOUGH are sent from the receiver.

A SIG_REQUEST sent before SIG_READY or between SIG_HEAD and the end of capture data transmission may be ignored. Otherwise, a SIG_REQUEST always triggers a response of either SIG_DENIED or SIG_HEAD.

A SIG_HEAD is always followed by capture data with no extraneous bytes in between. The transmitter may choose to terminate its capture and stop transmitting data at any time, even if the receiver never sent a SIG_ENOUGH. In either case, the last frame of data is specially marked (see the preceding section on the transmission format for details). After a SIG_ENOUGH, the transmitter may continue to send capture data if it chooses. At the very least, it will complete Trailer.

3.2.3 Sample High Level Communication Flow with Signals

4 Programmer's Guide

4.1 Receiving Data

The receiver code (in /recv) contains command line scripts to pull data from the transmitter for analysis and visualization. The computer does not interact with the Arduino peripherals such as the MEMS sensors directly, but instead sends signals to the Arduino that instruct it to begin transmitting sensor output data. A graphical interface is also provided to simplify this task. Receiver code is written in Python 3.6.2 and recv/requirements.txt[8] lists the required packages and their versions.

[6] Signal values are represented in C notation - the characters \x followed by two hexadecimal digits.

[7] Notice that capture data continues for a short time even after SIG_ENOUGH; this is to be expected.

[8] Run pip install -r requirements.txt to ensure that you have the correct packages.

4.1.1 Command Line Scripts

The file `calibrate.py` is used to calibrate the MEMS gyroscopes and accelerometers. The script gathers at-rest readings from the sensors and uses it to apply a constant bias to all future readings. This form of calibration accounts for linear error. To calibrate the sensors, orient the boards with the black boxes (the sensors themselves) facing upwards. The boards should be completely still, but do not hold them down by hand or touch the sensors with your skin. Now run the script. After a few seconds, the script will stop, and the calibration data will be saved.

Use `gyro_test.py` and `accl_test.py` to ensure that the calibration was successful. When the boards are at rest and facing upwards, the gyroscope outputs should be 0 for each axis, and the accelerometer outputs should be 1 for the z axis and 0 for the x and y axes.

Capture data from the transmitter using `capture.py`. Simply run the script, and it will terminate after a few seconds. The resultant capture data will be stored under `/recv/raw`. Note that use of this file to capture data is not recommended; it is far easier to use the graphical user interface, which allows starting and stopping the capture on-demand.

The purpose of `process.py` is to prepare the raw data for plotting. In the strictest sense, this is not a command line script because it takes no action upon invocation. The plotting script imports this file to processes the data that it needs, so there is no need to manually process your captures before trying to plot them.

The script `plot_mag.py` generates plots of the data. By default, it simply graphs the vector magnitude of the three component axes of the gyroscope data from each sensor. The first command line argument specifies the number of the capture to be plotted. For most use, it is not recommended to invoke this file directly because the graphical interface allows any capture to be plotted by clicking on it.

The code in `gui.py` runs the GUI which was discussed in the User Guide. You may build the GUI into an executable file by invoking `build_script.bat`. The executable may be distributed to and run on Windows computers that do not have Python or any of its packages installed. the executable will be saved into `recv/dist/`. The batch file does not need elevated privileges to run, but it may be necessary to remove the old executable from `recv/dist/` before running the build script, if it exists.

4.2 Capturing Data

The Arduino communicates with the MEMS sensors using 4-wire SPI in mode 0 at a speed of 10 MHz. At most one slave select pin is brought low at any time to choose which MEMS device to interact with. The other three critical SPI communication pins – `MOSI`, `MISO` and `SCK` – are handled by the Arduino SPI library and should not be used or configured manually. The program for capturing data from the sensors and transmitting to the receiver is stored in `trans/`. This code

is written as an Arduino Uno sketch. The file `trans/config.ino` contains the constant `CAPTURE_RATE` for configuring the speed at which the Arduino polls the MEMS sensors. In the same file, the constants `GYRO_SCALE` and `ACCL_SCALE` define the full-scale range of the gyroscope and the accelerometer for each of the two MEMS devices.

If the limit switch is connected to the Arduino, clicking it will allow the Arduino to include the time it was pressed in the capture data. The receiver will be able to plot these clicks as vertical lines, allowing the limit switch to mark off sections of the capture for later analysis.

The file `trans/pins.h` has three constants for configuring the operating mode of the transmitter. These modes are not exclusive. When `DEBUG_MODE` is on, additional information will be sent over the serial connection. Using this mode will make it impossible for the receiver to capture data, because the transmission protocol will be broken by the extra output. `TRANSMIT_MODE` controls whether or not capture data will be transmitted over serial. Finally, when `DEMO_MODE` is on, the three LED strips will light up with a limited view of the captured data. The number of LEDs that are illuminated on a particular strip indicate the measure of its associated metric.