American Sign Language Reader

Final Report

Group: T4
Alyssa Mathias
James Baak
Maaike Gooderham

Github:
https://github.com/JamesBaak/ASLR

SYSC3010A

Table of Contents

1.0 Project Description

1.1 Motivation

Communication is important to our society. However, it can be harder for those who rely on sign language to communicate with others as it is not a skill everyone possesses. This can make even the most simple articulation (like asking for directions) much more difficult.

With 5% of Canadians 15 or older reporting a hearing limitation and American Sign Language (ASL) being one of the two legitimate sign languages in Canada, there is a need for a product that overcomes these communication barriers [1][2]. The team hopes that using the American Sign Language Reader (ASLR) will remove these social barriers by providing ASL Sign Language users a way to translate their gestures into readable text for those who may not understand ASL. By integrating the ASLR with an Android app, the ASLR can monitor gestures and output the corresponding letters on a phone screen for people to read.

1.2 Scope

With the amount of time that had been allotted for this project, the team was aware that the entire alphabet was not feasible. To combat this issue, the team limited its goals to the integration of the following:

- The integration of a system that can read hand gestures
- The integration of a system that can read the data from these gestures and identify them correctly
- The integration of a system that can output the result of the data onto a phone screen in text form
- The integration of a system that can correctly identify 6 distinct American Sign Language (ASL) characters: B, I, L O, Y and 1

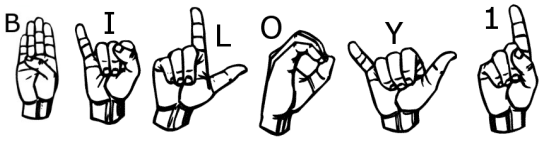


Figure 1: Chosen ASL Signs for Project [3]

The ASL letters B, I, L, O, Y, 1 were selected as their respective symbols are quite different from one another. These letters provide a base goal that will be more feasible in the time permitting than the entire alphabet. The team handled these goals in such a way that subsequent phases could be created for the project should it continue being developed. (i.e. the implementation of the rest of the alphabet, words, etc.)

2.0 System Architecture Design

2.1 Deployment Diagram

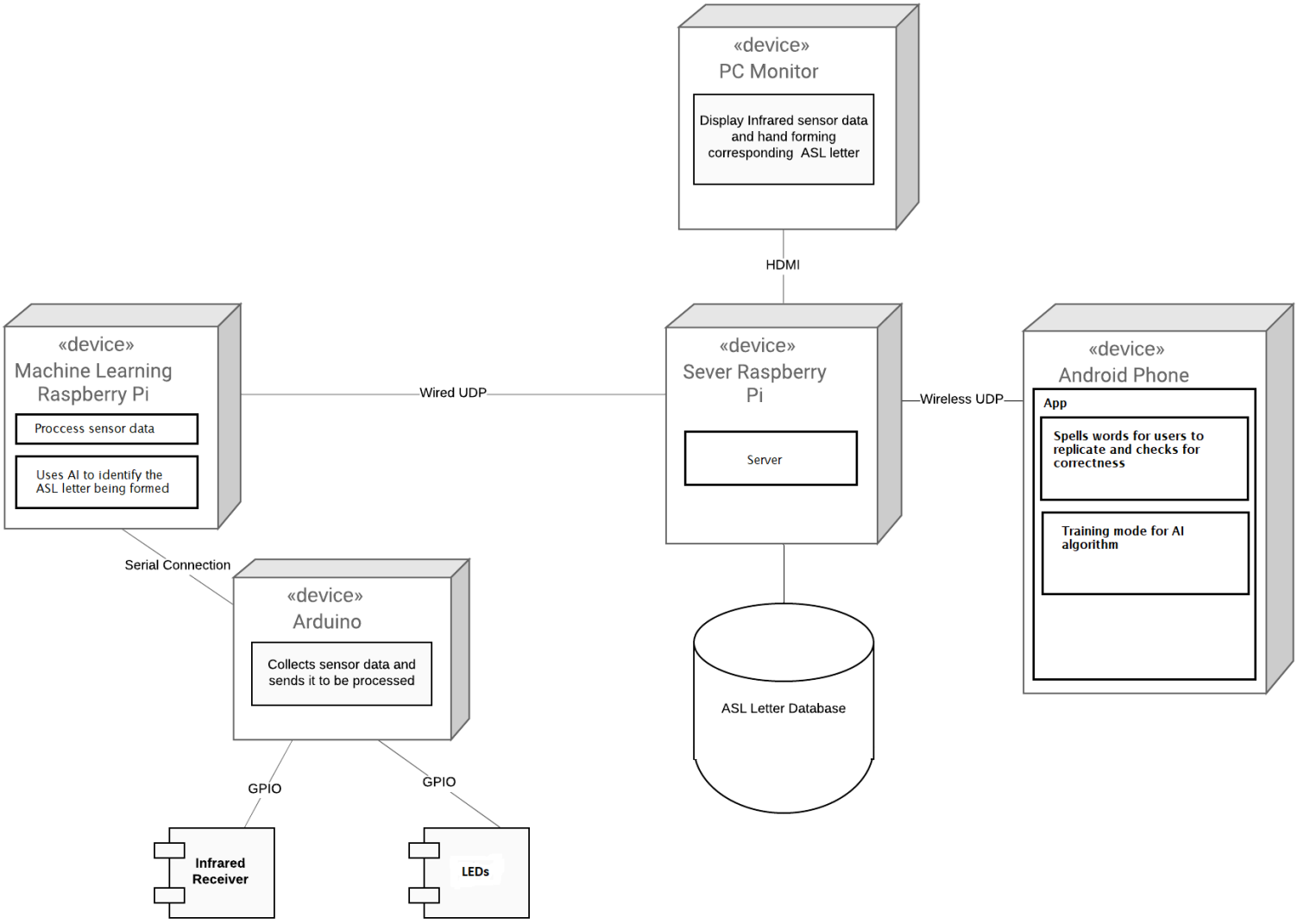


Figure 2: ASLR Deployment Diagram

Figure 2 describes the final deployment for the ASLR. Due to hardware difficulties, the array of infrared sensors and receivers have been replaced with with six infrared receivers and a series of 14 LEDs. Instead of reading a hand gesture with a bracelet array of infrared transmitters and receivers, the user selects the gesture they wish to display by shining an infrared light from a remote on one of the six infrared receivers. The LEDs indicate when the Aurduino is sending data; they will not be flashing when

this is the case. They also demonstrate the multiplexor circuit created for the infrared emitters. Updating the hardware allowed the functionality of the system to be tested even though the design did not follow the original plan.

The rest of the system operates as indicated in the design document. The machine learning (ML) Pi uses the gathered sensor data to detect what "gesture" is being made. The result is then sent to the server, which stores this information in the database for use in training the ML algorithm. The server displays the predicted letter and the gathered sensor data. It also sends the prediction to the Android application, which displays the results. The Android application is the method by which the user interacts with the system by allowing the user to issue a sample request, train the ML algorithm and display the predicted results.

2.2 Message Protocol Table

Table 1: Communication Protocol descriptions and formats
Note: (ml pi) - Machine Learning Raspberry Pi = ml pi, Server Pi = Server

Sender	Receiver	Message Description	Channel	Format
Arduino	Pi (ml pi)	The array of sensor values from the GPIO of arduino in Machine Learning Pi's buffer.	Serial	Float[196] (The single array will be divided) Float values are transmitted over serial through comma separated strings end with the new-line character ('\n') Eg. "1.2,1.3,1.4, …. ,1.6\n… repeat"
Pi (Server) Android	Pi (ml pi) Pi (Server)	Sample buffer (Request)	UDP	JSON: { type : "sample", "payload" : c } Where c is an int representing a character in "BILOY1", if there is no char c, then only the prediction will be retrieved and it won't be added to the training set.

Pi (ml pi)	Pi (Server)	Save result; payload is a json (Response)	UDP	The JSON returned by the ML Pi after a sample request:
				{
				"type" : "save",
				"payload" : {
				"prediction" : p: int,
				"class" : c: int,
				"input" : i: int[14][14]
				}
				}

Pi (Server) Pi (ml pi)	Android Pi (Server)	Acknowledgement that the sample, load, and create_user request was received.	UDP	The common ACK packet used to notify components of the system that their request has been received. JSON: { type : "ack", payload: "" }

Sender	**Receiver**	**Message Description**	**Channel**	**Format**
Android	Pi (Server)	Request to load the specific user's event records on ML Pi for fitting the ML model.	UDP	The JSON of the load request to the server: { type: "load", payload: username: string } Where username is a string representing the username of a user or is a blank string. If the username is defined and exists, the saved ML results of that user will be loaded into the ML Pi. If the username is blank, all user's results will be loaded into the ML Pi.

Pi (Server)	Pi (ml pi)	The data records from the database used to train the MLP neural network. These records are loaded into the ML Pi after a load request to the server.	UDP	{
				type : "records",
				payload: {
				records: r : **Record**[n],
				remaining: r : int
				}
				}
				A **Record** is the same type as the payload in the message above. *remaining* is the number of record messages waiting to be transmitted. This is needed because UDP has a limit of 65,536 without accounting for headers. In case there are many records, the server should be able to supply any number of records to the ML Pi. [6]

Pi (Server)	Android	Return of labelled data. Notification from ML Pi.	UDP	The JSON return to the Android App to notify user of prediction:
				{
				type: "prediction",
				payload: i :int
				}

Android	Pi (Server)	Request to create a new user in the ASLR system. The server should send an ACK to notify Android App that the user has been created.	UDP	The JSON below represents a request to create a new user:
				{
				type: "create_user",
				payload: {
				username: u : string,
				saltValue: s : string,
				password: p : string,
				developer: d : int
				} }

Sender	Receiver	Message Description	Channel	Format

Android	Pi (Server)	Request to get a user's details for login.	UDP	The JSON to get a user from the database:
				{
				type: "get_user",
				payload: username: string
				}

Pi (ml pi) Pi (Server)	Pi (Server) Android	If there are errors in processing requests, error responses will be returned to the requestor to notify them of the error.	UDP	The returned JSON on error in processing request: { type: "error", payload: errorMsg: string }

2.3 Sequence Diagrams

Figures X and X describe the sequence diagram of the system. In order to use the ASLR, the user must first login to the system. The entered password is compared with the password in the database. If the passwords match. If not the user is able to try and login again. If the user has successfully logged in the ML Pi loads that user's training data and they can sample the sensor data. As seen in the diagram, the user sends a sample request to the application which forwards it to the server which then sends it to the ML Pi. The ML Pi then samples the sensor data from the Arduino and makes a prediction. The data and the prediction are sent to the database to be stored, and the prediction is sent to the server. The server sends it to the Android application, which displays the results on the screen. If the user is a developer, there is another option to train the ML Pi. This follows the same process as sampling the sensor data with the addition of a character indicating the created "gesture".

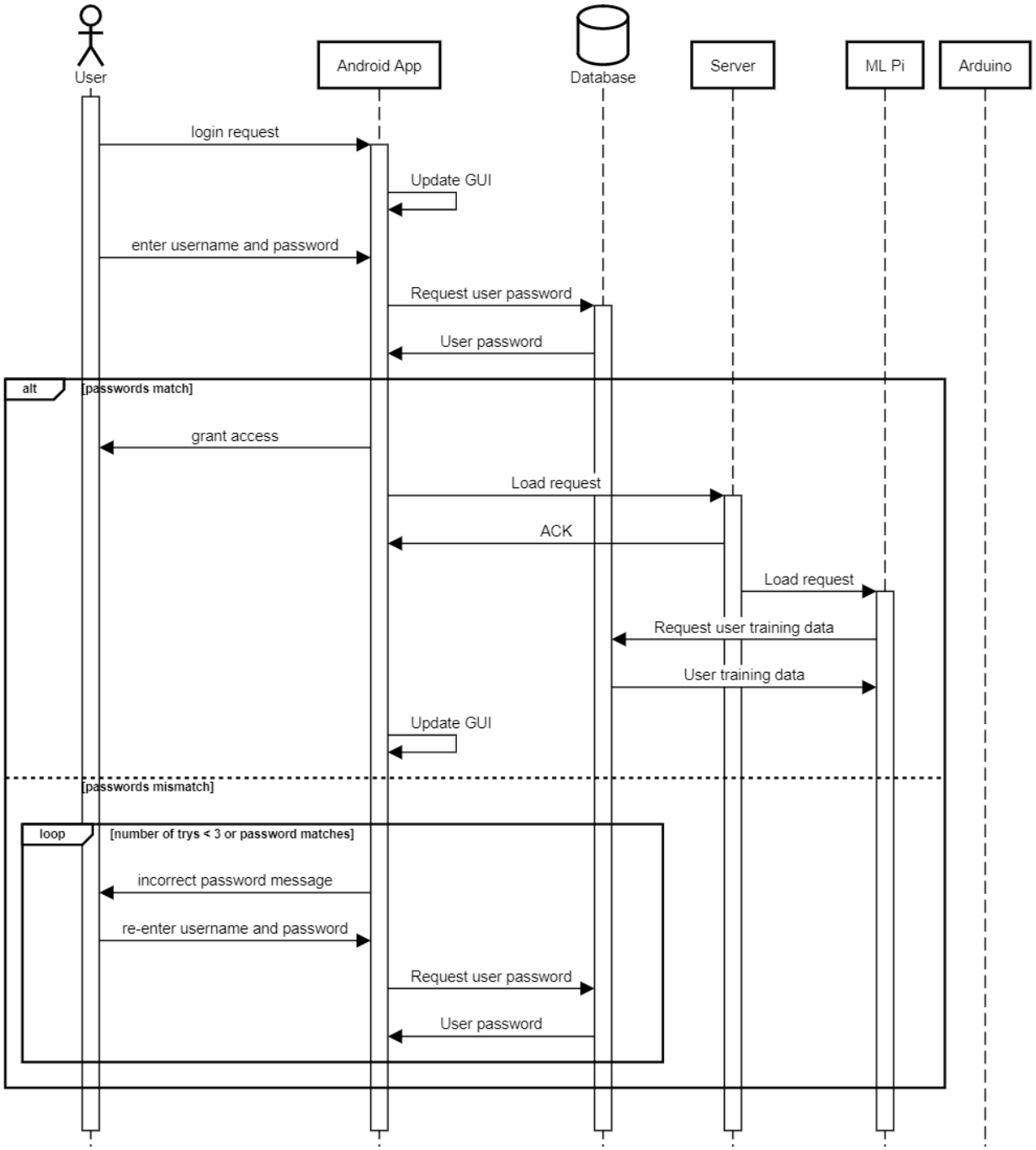


Figure 3: Sequence Diagram for ASLR System Part 1

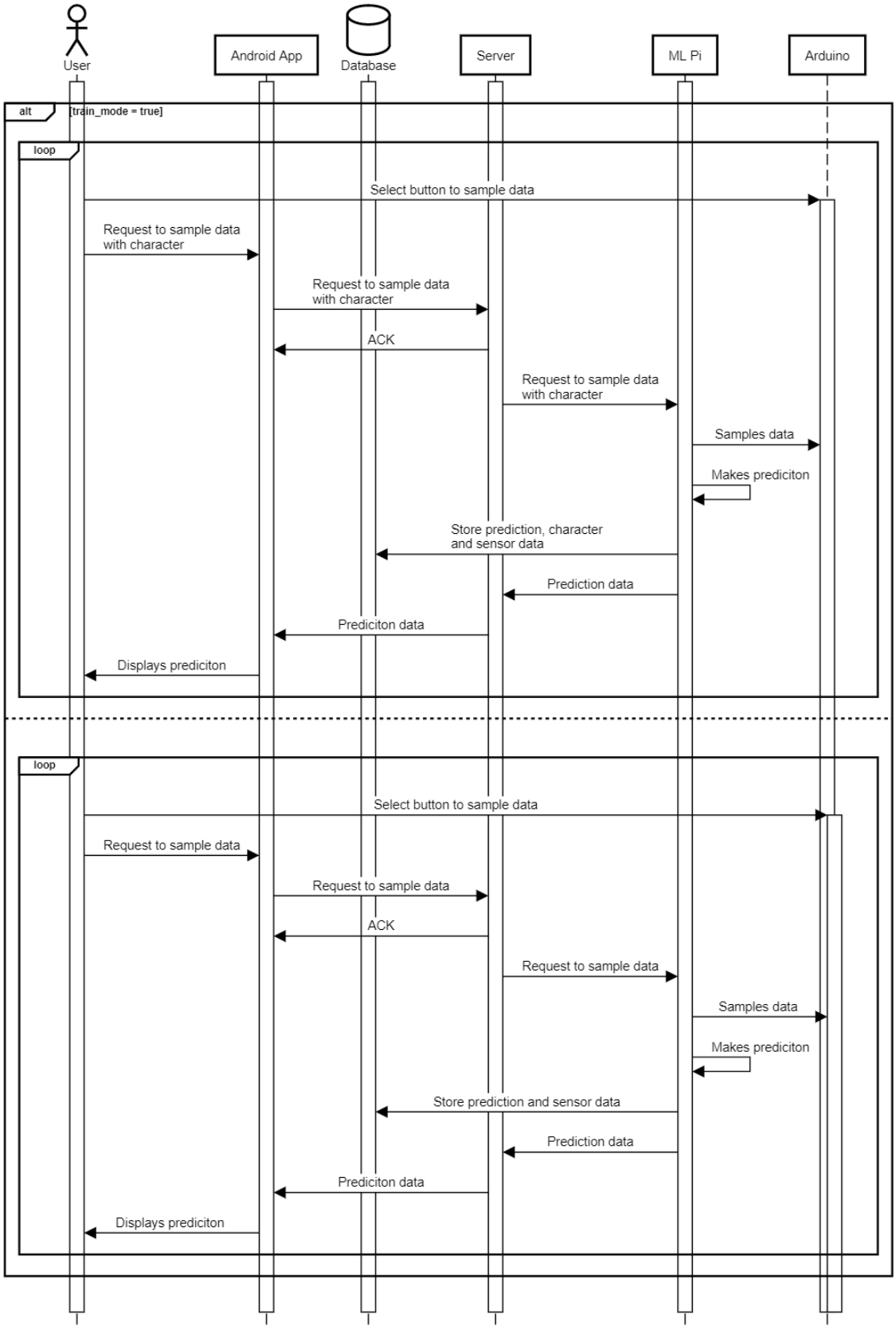


Figure 4: Sequence Diagram for ASLR System Part 2

3.0 Discussion of Results

3.1 Hardware

The hardware included the set up of two different circuits; one for the IR receivers and one for the IR Emitters. As can be seen in Appendix B, Figures 8 through 11, the circuits went through a few changes before we settled on a final design.

3.1.1 Circuit Changes

The immediate difference that can be seen between the past and present circuits involves the multiplexing and demultiplexing of the signals. While the original circuits boasted two 8:1 and one 2:1 muxes, the circuits were simplified to the use of two 16:1 muxes. Initially, research concluded that 16:1 microchips were not available for breadboards. This caused issue, as many of the team's components did not belong to us personally which made chips involving soldering impossible for use. However, after the completion of the design document, the team happened upon 16-channel analog multiplexers that were integrated on breakout boards. Though breakout boards themselves require soldering (if considering the addition of pins), the pinouts on the break out board lined up perfectly with those on the breadboard, allowing us to attach it to the breadboard without the use of soldering.

3.1.2 The Power Issue

The next primary issues came from the amount of power the circuits needed, and the IR emitters themselves. In the case of power, the Arduino UNO R3 does not have enough power to run heavy duty circuits such as the ones the team had implemented. The absolute maximum ratings for any general purpose IO pin (GPIO) coming from the Arduino is 40 mA. [4] Using the 5V pin supplied by the Arduino was out of the question as well, since the maximum the pin could draw safely was 200 mA. [4] With 35 connections between the emitters, receivers, multiplexers and the operational amplifiers, this was not enough power to support our circuits. Granted the amount of inputs was greatly reduced (due to the open circuits created by the multiplexer since it can only connect one channel at a time), but the team still had to find a better way to provide power to the remaining connections. The solution to the team's power issue was the power module provided in the Arduino kit. It is a power module that converts 7 to 12V into either 5V or 3.3V through the use of two voltage regulators and the maximum amount of current that it outputs is 700mA. [5] By connecting a 9V battery to this power module, the team now had more than enough power to run the circuits (and the Arduino) in a proper fashion.

3.1.3 The Emitter Issue

As previously mentioned, the IR emitters caused many issues for the team. It was initially thought that the IR emitter circuit was poorly assembled and required tweaking before actually setting up. However, after testing multiple parts of the circuit individually (the code, the mux selector pins, the mux itself), it became clear that the problem was not the circuit itself. Further research was conducted, hoping to find a resolution to the issue. The next assumption made was that the emitters were not working due to

a lack of power. This was only concluded not to be the case later by using the power module mentioned in 3.1.2. After triple checking the datasheets and making sure the specifications of the emitters were understood, the team redoubled their efforts in an attempt to make them work. Several methods were attempted to make the emitters function. These methods included:

- Setting an IR emitter in series with a resistor (for current limiting) using a range of resistors from 100 - 1k Ohms
- Testing the circuits with a DC signal and an analog/PWM signal (repeating resistor values for each) (PWM duty cycles: 33% and 50%)
- Setting up a transistor circuit to amplify the current running through each emitter, using the resistor values and signals mentioned above
- Changing the arduino's clocks to supply a higher carrier frequency (34-38kHz) for the PWM signals by going into the arduino's registers and repeating everything above

Unfortunately, these methods did not resolve the issue. Deeming the emitters unuseable, the team decided to settle for an alternative method to prove that the rest of the system was functioning as it should. The solution was to set up 6 receivers separately from the receiver multiplexer (one for each ASL letter we were using) and flash an IR LED at them coming from a tv remote. This way, the hardware code would still have data to send along to the Raspberry Pis for use in testing the rest of the system.

3.1.4 Infrared Transmitter and Receiver Bracelet

Before the hardware was changed due to the previously mentioned issues, a bracelet was created to hold the transmitters and receivers. The bracelet followed the design described in the *SensIR* paper and contains holes for the transmitters are receivers [5]. It was cut from a piece of laser rubber using a laser cutter. Buckles printed using a 3D printer were used to fasten the bracelet. Figure X shows the completed bracelet.

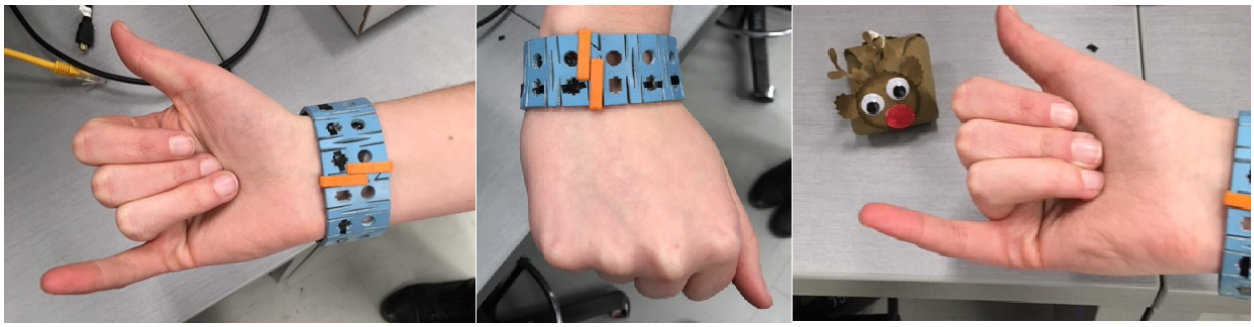


Figure 5: Wearing the Infrared Sensor Bracelet while Signing a Y

3.2 Software

3.2.1 Android Application

The Android application allows users to interact with the ASLR through a graphical user interface. The opening menu has two options: logging in and creating a new user. There are two access permissions for user accounts: user and developer. These permissions affect how the user can interact with the ASLR.

Creating a new user is the first mode of operation. To create a user a username and password must be entered. The user must then indicate if the new account will have developer permissions. By default new accounts will only be able to access user modes. Only a developer account can be used to give developer access to a new user account to keep developer permissions secure. This information is then sent to the server Raspberry Pi to be stored in the database. The password will be encrypted before it is sent to the database so the encrypted password is stored.

Because a developer account is needed to give developer access to future accounts, the first account created will automatically have developer access. This is a special case and does not follow the usual process of user creation.

Current users must first login in order to access the options available to them. First, the user must enter their username and password. The username is then sent to the database and the hashed password for that username and the salt value are retrieved. The Android application then hashes the user entered password using the retrieved salt value and compares the results. Comparing the hashed passwords means the password hash function does not have to be reversible. If the passwords match, the user is then logged in. If the passwords do not match, the user has two more tries after which the application will be locked for five minutes. The limited amount of password retries and timed lockout will discourage an attacker from brute force guessing a user's password.

With a regular user account, the user can create a new user with regular user permissions or identify a formed letter. The user clicks a button indicating that the sensors should start collecting data. They then create an ASL letter. The sensor data is processed and sent to the neural network to identify the formed letter. The results are stored in the database and the predicted letter is displayed on the application screen. Users then have the option of repeating the process with another letter.

The developer user account has the same permissions as the regular user account with the added option of providing feedback to the neural network for training. The developer clicks a button indicating they wish to train the neural network. The Android application then sends a request to start sampling sensor data and the developer creates an ASL letter and indicates the letter they created. The sensor data is processed and sent to the neural network to identify the formed letter. The results are stored in the database and the predicted letter is displayed on the application screen. The developer then has the option to repeat the process with another ASL letter.

3.2.2 Neural Network

The neural network is the component within the Machine Learning Pi that samples the buffer of the Arduino and predicts what gesture is being formed. The network is a Multilayer Perceptron model with one hidden layer using a One-vs-Rest strategy to classify the multiple gestures of the ASLR system. For a more detailed description of the definitions, configuration, usage of the neural network used in the ASLR system please see the design report for the project.

When implementing the machine learning model in python, it was originally planned to use 5-fold cross-validation with the model against all data to avoid overfitting and tune hyperparameters. [6] Since we are also using "online" learning to train the model before a prediction during training mode, the model's training and validation time must be short to return a prediction to the user promptly. With this in mind, it was determined that using the train-test split method in scikit learn would be effective in testing bias in the model using the accuracy metric because a small portion of the data would be saved for testing

the accuracy of the trained model. The train-test split method still satisfies the requirements of the original design by ensuring that the machine learning model does not overfit to the data, which is a concern in very small datasets. The regularization defined in the SensIR paper was also used to counter the bias of the dataset.

Figure 6: Output of ML Pi after sampling and training of ~20 samples.

The issues in hardware and the time constraints of the project made it difficult to experiment and tune the hyperparameters of the model. Therefore, the hyperparameters and tools of the project used the values outlined in the SensIR paper. The paper claims to have reached an accuracy of 93.3% using 14 segments with a training data set of 1200 samples. As shown in figure x, our machine learning model reached an accuracy of 33% after about 20 samples. This is still far from the goal of 80%, but is expected after the few samples that are used for the training and testing of the machine learning model. With more time, a balanced number of gesture samples could be obtained to further train the model and improve its accuracy. Using the 6 infrared receivers from the Arduino and an IR remote to select the desired gesture, the simple neural network would learn which receiver maps to which gesture after a large and balanced dataset is produced. Further hyperparameter tuning could be employed in future iterations to decrease the bias and increase performance.

4.0 Contributions

Included in the following tables are the respective parts each member worked on for the project and the final proposal. A brief description is provided in Table 2 to explain what each file was used for.

Table 2. Code/Hardware contributions

File Name/Code/Hardware	Author	Description
program.ino	Alyssa	Arduino code for running the hardware/circuits
hardware/circuits	Alyssa	The circuits for the IR emitters and IR receivers, including their connections with the Arduino
CreateNewUserActivity.java	Maaike	The Android activity that controls the creation of new users
DeveloperActivity.java	Maaike	The Android activity that controls developer options
LoginActivity.java	Maaike	The Android activity that controls logging in users
MainActivity.java	Maaike	The main user screen for the Android application
SampleActivity.java	Maaike	The Android activity that controls sending training and sample requests

File Name/Code/Hardware	Author	Description
main.py	James	The main functional loop program for the Machine Learning Raspberry Pi.
serial_buffer.py	James	The python thread class to populate and control the buffer of data from the Arduino.
serial_test.py	James	A test program to ensure proper communication over the serial channel between the Machine Learning Pi and Arduino
udp_client.py	James	Simple UDP client to test the functionality of the Server.
udp_server.py	James	The server of the ASLR system. Handles the requests from the Android application and uses the Machine Learning Pi to make predictions on sample requests.
server_test.py	James	Contains the server unit tests to ensure proper implementation of design.

Table 3. Final Report contributions

Section Name	Author
1.1 Motivation	Alyssa
1.2 Scope	Alyssa
2.1 Deployment Diagram	Maaike
2.2 Message Protocol Table	James
2.3 Sequence Diagrams	Maaike
3.1.1 Circuit Changes	Alyssa
3.1.2 The Power Issue	Alyssa
3.1.3 The Emitter Issue	Alyssa
3.1.4 Infrared Transmitter and Receiver Bracelet	Maaike
3.2.1 Android Application	Maaike
3.2.2 Neural Network	James
4.0 Contribution Tables	Alyssa, James, Maaike

Section Name	**Author**
Appendix A	Alyssa, James, Maaike
Appendix B: Circuit Designs	Alyssa
Appendix C: Android Application Class Diagram	Maaike

5.0 References

[1] “Facts on Hearing Limitations,” *Statistics Canada: Canada's national statistical agency / Statistique Canada : Organisme statistique national du Canada*, 26-Feb-2009. [Online]. Available: https://www150.statcan.gc.ca/n1/pub/89-628-x/2009012/fs-fi/fs-fi-eng.htm.
[2] “Language,” *Canadian Association of the Deaf - Association des Sourds du Canada*. [Online]. Available: http://cad.ca/issues-positions/language/.
[3] Lifeprint, Lifeprint Wallpapers, https://lifeprint.com/asl101/topics/wallpaper1.htm
[4] "Arduino Playground - ArduinoPinCurrentLimitations", *Playground.arduino.cc*, 2019. [Online]. Available: https://playground.arduino.cc/Main/ArduinoPinCurrentLimitations/. [Accessed: 10- Nov- 2019].
[5] *MB102 Breadboard 3.3V/5V Power Supply*. Handson Technology, 2019.
[6] “5.7.1. Cross-Validation” The Hundred-Page Machine Learning Book, Andriy Burkov., 2019, pp. 60.
[7] Jess McIntosh *et al.,* “SensIR: Detecting Hand Gestures with a Wearable Bracelet using Infrared Transmission and Reflection”, UIST 2017, Quebec City, Canada, Oct. 2017

Appendix A

Android Application README

Android Application

The Android Application contains the following files:
In ASLR\Android\ASLRApp\app\src\main\java\com\example\aslrapp:
1) CreateNewUserActivity.java Controls the creation of a new user
2) DeveloperActivity.java Contains the opening developer screen
3) LoginActivity.java Controls logging in a user
4) MainActivity.java The opening screen of the app
5) SampleActivity.java Controls sampling the sensors, displaying the prediction and training the ML Pi
In ASLR\Android\ASLRApp\app\src\main\res\layout:
1) activity_create_new_user.xml Layout for CreateNewUserActivity
2) activity_developer.xml Layout for DeveloperActivity
3) activity_login.xml Layout for LoginActivity
4) activity_main.xml Layout for MainActivity
5) activity_sample.xml Layout for SampleActivity
In ASLR\Android\ASLRApp\app\src\main\res\values
1) strings.xml Contains string values for app layouts

ASLR\Android\ASLRApp\app\src\main\res\drawable contains the images displayed in the app

Tests are located in the following directory:
ASLR\Android\ASLRApp\app\src\test\java\com\example\aslrapp

An APK to install the application can be found in the directory:
ASLR\Android\ASLRApp\app\build\outputs\apk\debug\app-debug.apk

For testing, the app the following user has been created:
1) username: james1 password: baak

Arduino + Hardware README

ASLR - Arduino

This is an Arduino program for sequentially turning on IR emitters and recording the amount of current generated by them
through IR receivers/photodiodes. The data collected is then sent to Serial for transmission to a Raspberry Pi 4.

This is a third year university project for undergraduate studies in computer engineering.
It was an experiment to test the feasiblity of creating an American sign language to text reader by referencing the University of
Bristols's SensIR project.

https://www.youtube.com/watch?v=hCahI7ZRbOA
https://dl.acm.org/citation.cfm?id=3126594.3126604

Set Up

What you'll need to run this program:
- x1 Arduino Uno
- x1 USB cable type A/B
- x2 CD74HC4067 (16:1) Multiplexers (Robojax)
- x4 Operational Amplifiers (LM324N - Texas Instruments)
- x14 IR Transmitters (SFH4556P - Osram Opto Semiconductors)
- x14 IR Receivers/Photodiodes (BPW34FA - Osram Opto Semiconductors)
- x14 100k Ohm Resistors
- x14 20 Ohm Resistors
- x14 1k Ohm Resistors
- x14 22pF capacitors
- X14 2N2222A transitors (or 2N3904)

IR Receiver Circuit

Using all 14 receivers and all 4 op amp chips will result in four identical subcircuits. Figure 1 below shows one of those four identical subcircuits. It is four transimpedance circuits (using one IR Receiver, one resistor, one capacitor each) all connected to an LM324N. The resistance values here are all 100k Ohms. The capacitance values here are all 22pF.

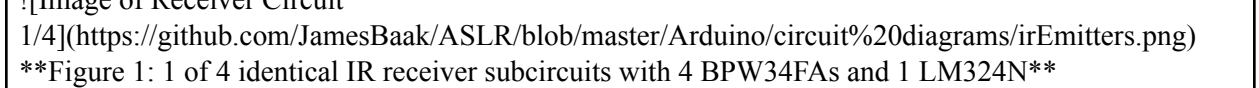


Figure 1: 1 of 4 identical IR receiver subcircuits with 4 BPW34FAs and 1 LM324N

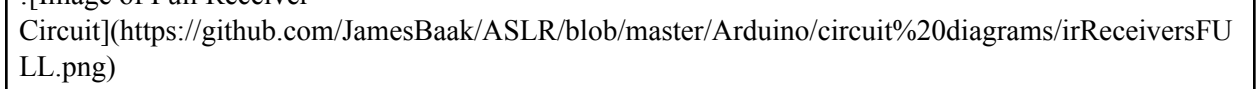
Figure 2 below shows the full IR Receiver Circuit once all four subcircuits have been combined.


Figure 2: Full IR Receiver Circuit with 4 subcircuits, 1 mux and 1 Arduino Uno

IR Emitter Circuit

Each emitter is part of its own transistor circuit involving a 2N2222A transitor, a X resistor connected to the transistor base and a X resitor in series with the LED at the transitor collector. This is to amplify the signal that is put out by the IR LEDs. Each transistor circuit is connected to one channel of the MUX. There are 14 of these transistor circuits.

)
Figure 3: The IR Emitter circuit

Steps
1. Set up the circuits shown above
2. Open the program.io sketch in the Arduino IDE
3. Set up Serial.print() lines or use the sendVoltage method to send out on the TX pin
4. Verify the sketch
5. Connect the Arduino UNO
6. Upload the sketch to the Arduino Uno

Server README

Running the Server:

1. Navigate to the Server folder
2. The addresses and ports of the server and machine learning Pi's can be modified in the `udp_server.py` file. Maybe move this to a file later to read in address values.
3. Run the UDP server using Python > 3

```
python udp_server.py
```

The server should connect to the database with the Database folder. If it does not exist it will be created, but will not function as the tables do not exist. If the server is unable to connect to the Machine Learning (ML) Pi or if there is an issue, the server will return an error to sample requestor indicating the issue with the ML Pi.

Interfacing with the server:

The server takes UDP requests with a string representing a JSON in the content. The JSON should be encoded as a string with UTF-8. The JSON has the keys `type` and `payload`. The keys/fields are required to communicate with the server, otherwise an error message will be returned. Table of supported types and their respective payloads:

Type	Payload	Response
sample	empty string to get prediction from ML PI or an int representing one of the gestures	{ "type": "prediction", "payload": int }
load	The username of the user's data that the machine learning algorithm should train with or empty string to load all records	{ type: "ack", payload: "" }
create_user	[json: { username: string, saltValue: string, password: string, developer: int }]	{ type: "ack", payload: "" }
get_user	The username of the user's record to return	{ type: "user", payload: { "username" : string, "salt" : string, "password" : string, "developer": int }}

Machine Learning Pi README

Machine Learning Raspberry Pi

Description:

The machine learning Pi is the component in the ASLR system that samples the buffer between the Arduino and the Pi buffer thread, and predicts the current user's gesture. The prediction is formed by passing the standardized 196 sized float buffer into a simple neural networks using the One-Vs-Rest multiclass classification strategy and hyperparameters discussed in the SensIR paper.

SensIR paper: Jess McIntosh et al., “SensIR: Detecting Hand Gestures with a Wearable Bracelet using Infrared Transmission and Reflection”, UIST 2017, Quebec City, Canada, Oct. 2017

Starting the program:

1. Connect the Arduino and ensure that it is running the program to output Infrared receiver values over serial using the communication protocol described in the design and final report document.
2. Use command `ls /dev/tty*` to search for serial port on a Pi terminal. The serial port should be `/dev/ttyACM0`, but may vary over Pi updates.
3. Connect the Pi over ethernet to the Server Pi and update the Server with the address of the Machine Learning Pi on the ethernet interface. The address of the Pi can be found using the command `ip a`.
4. Start the `main.py` program using Python 3: `python main.py`. The program should connect to the Pi over serial, otherwise it will throw an error describing to the user that the Arduino is not setup. The buffer data from the Arduino should be printed to the console, but the `serial_test.py` program can be run to validate connection and proper communication.
5. Follow the instructions in the other system components README.md files to use the entire ASLR system.

Appendix B: Circuit Diagrams

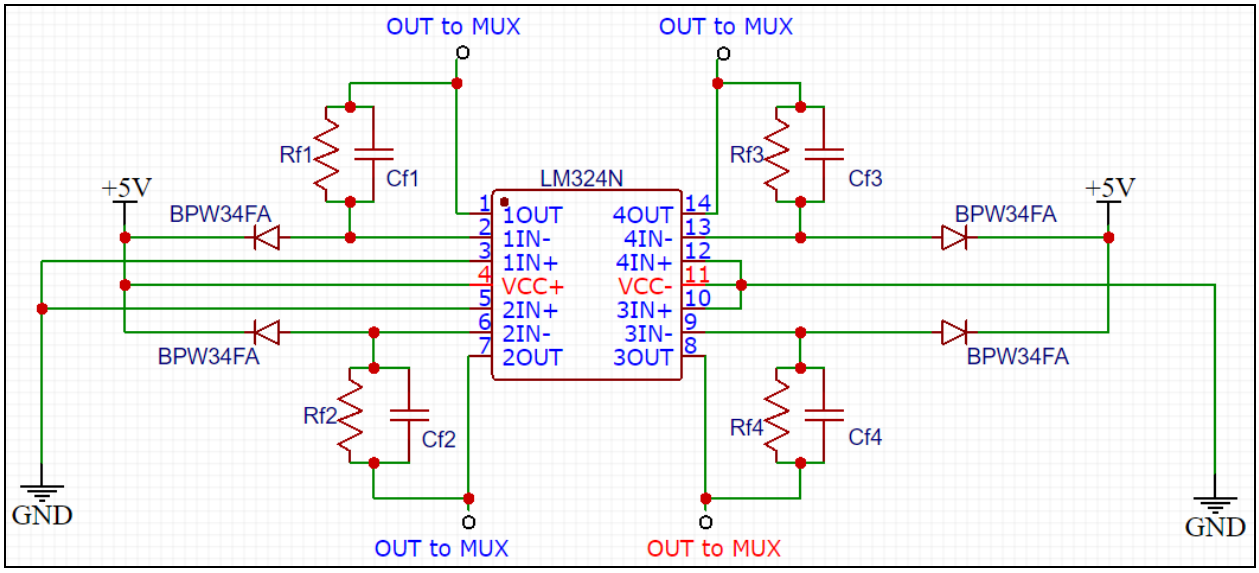


Figure 7. 1 of the 4 subcircuits that compose the IR Receiver circuit

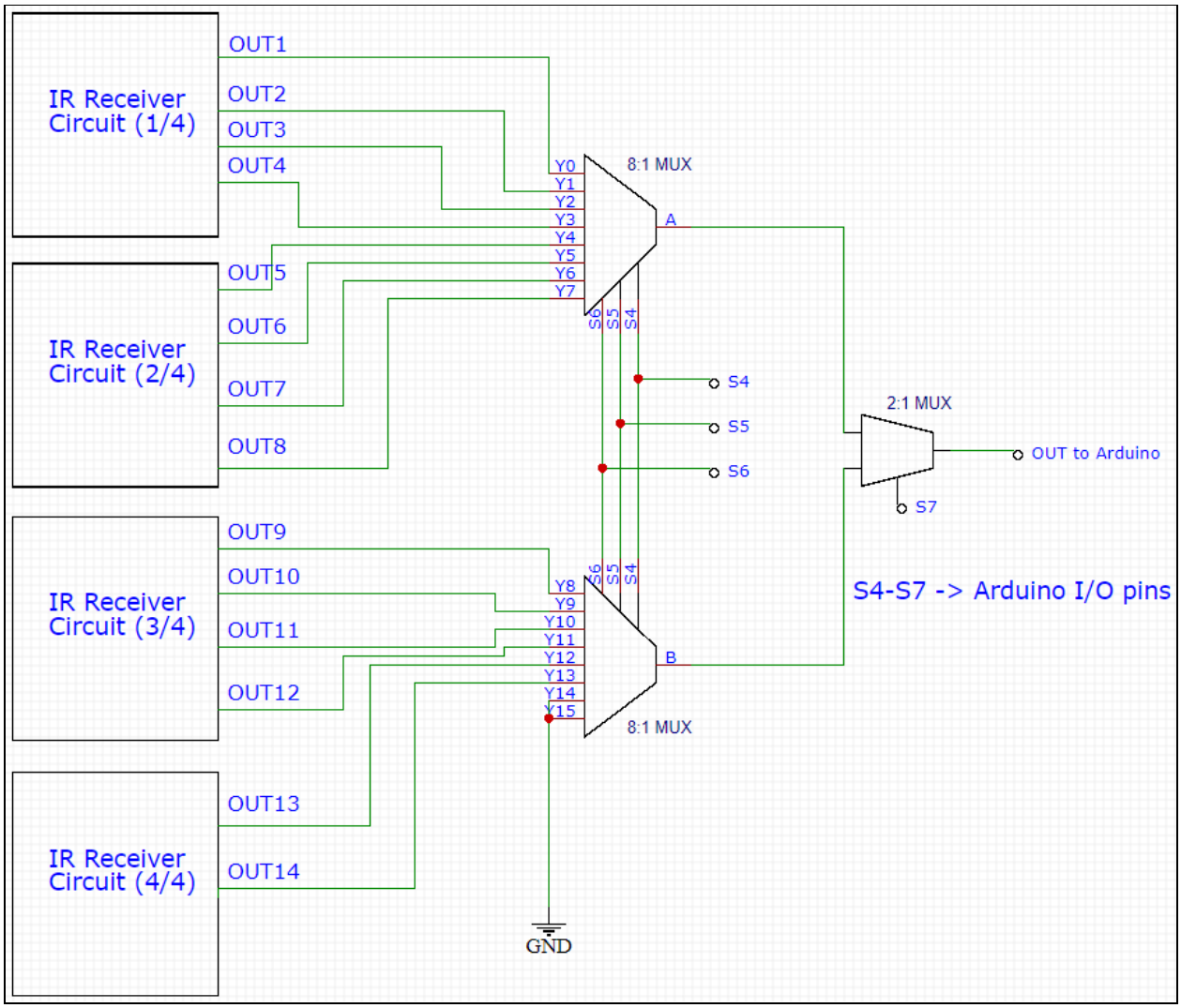


Figure 8: The Full IR Receiver circuit before design changes were implemented

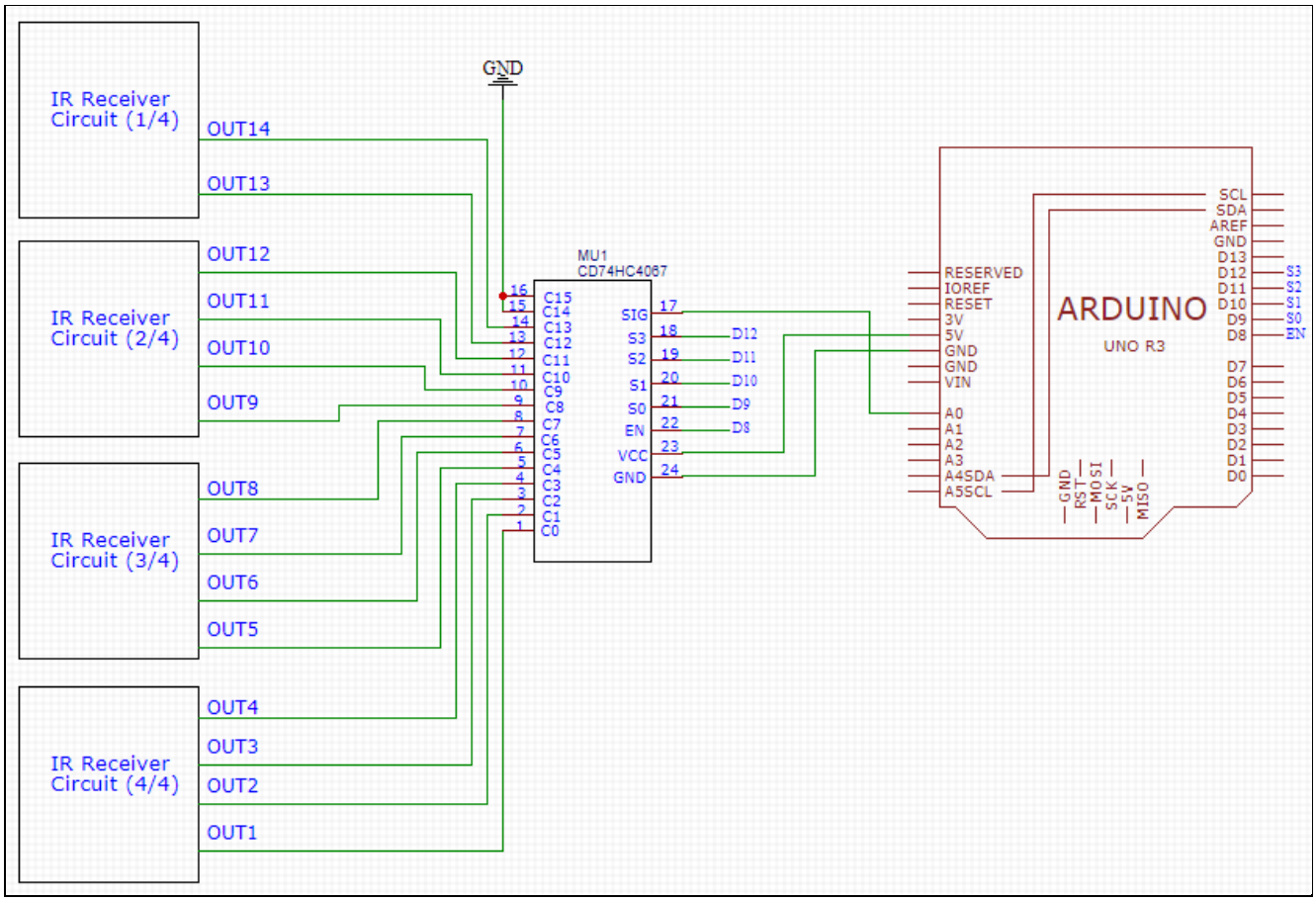


Figure 9. The entire IR Receiver circuit after design changes were implemented

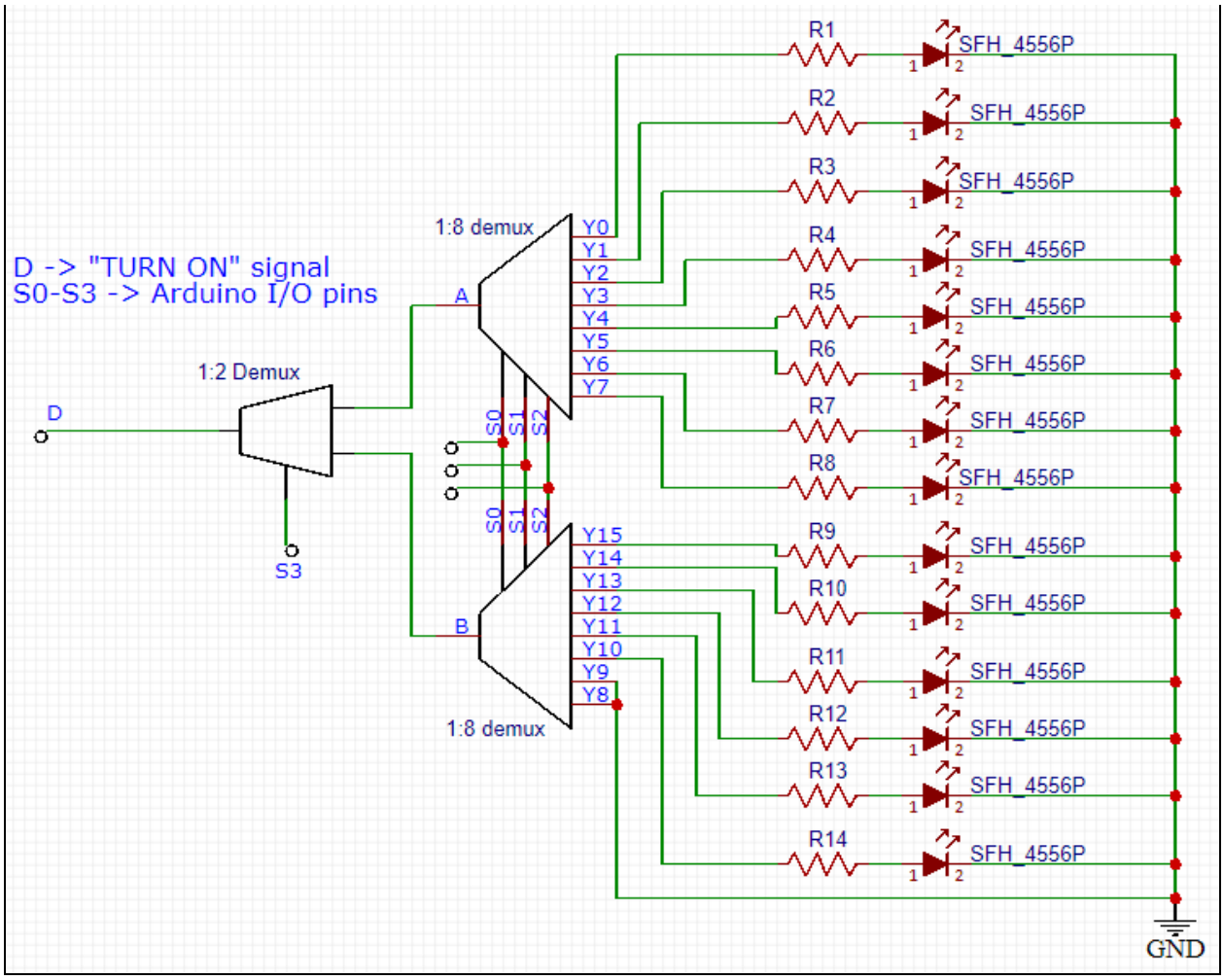


Figure 10. The IR Emitter circuit before design changes were implemented

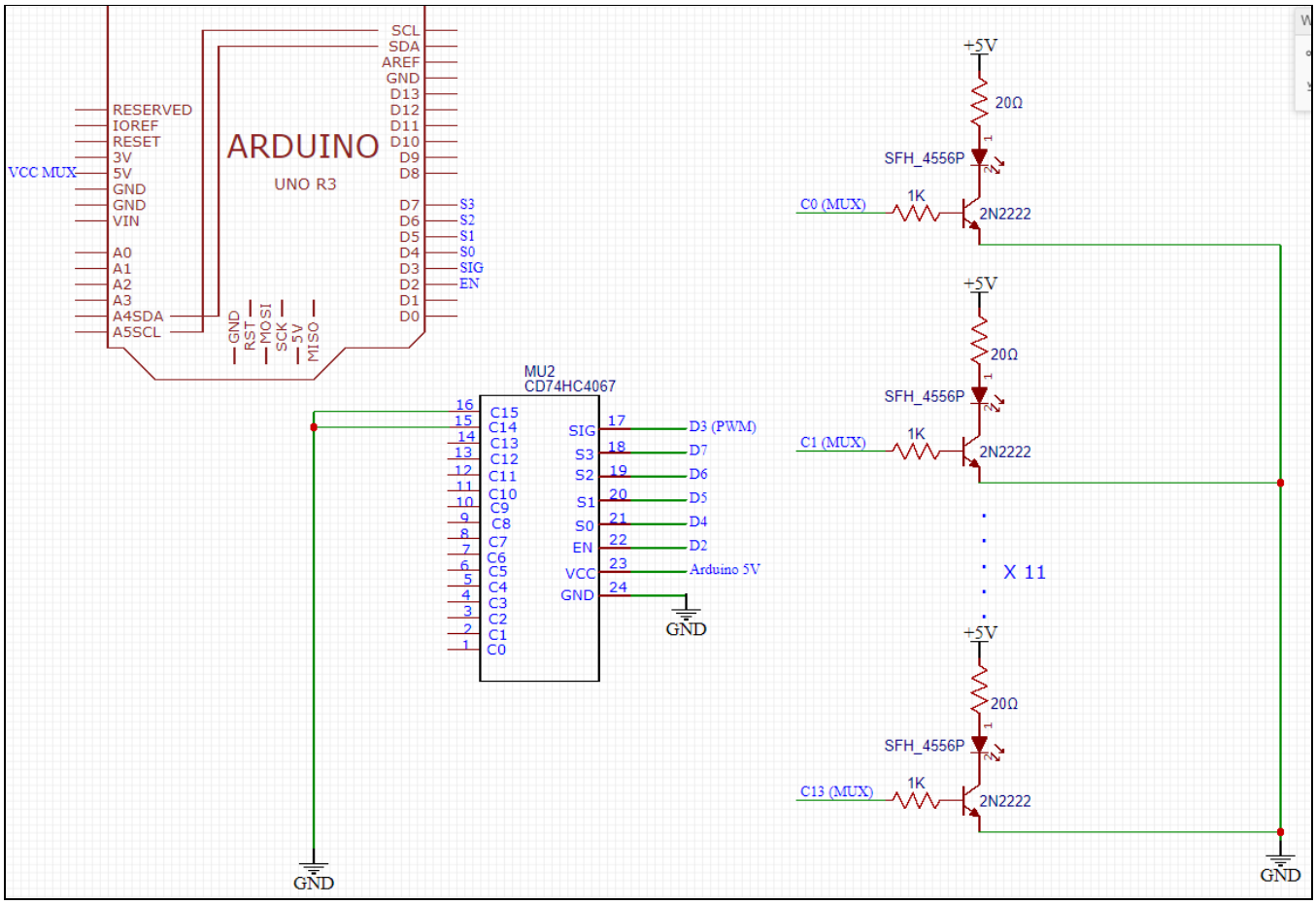


Figure 11. The IR Emitter circuit after design changes were implemented

Appendix C: Software Diagrams

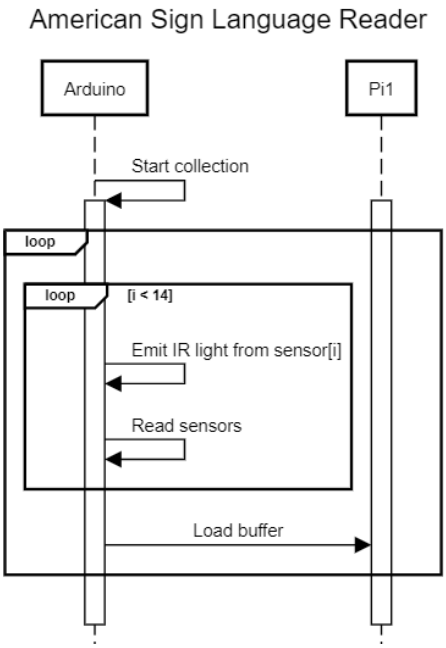


Figure 12: Arduino and ML Pi's buffer loop sequence diagram

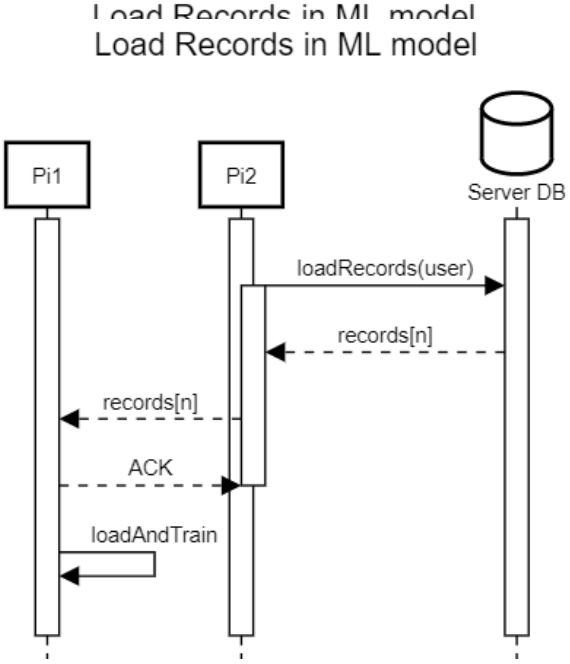


Figure 13: Loading labelled data into the machine learning model for training

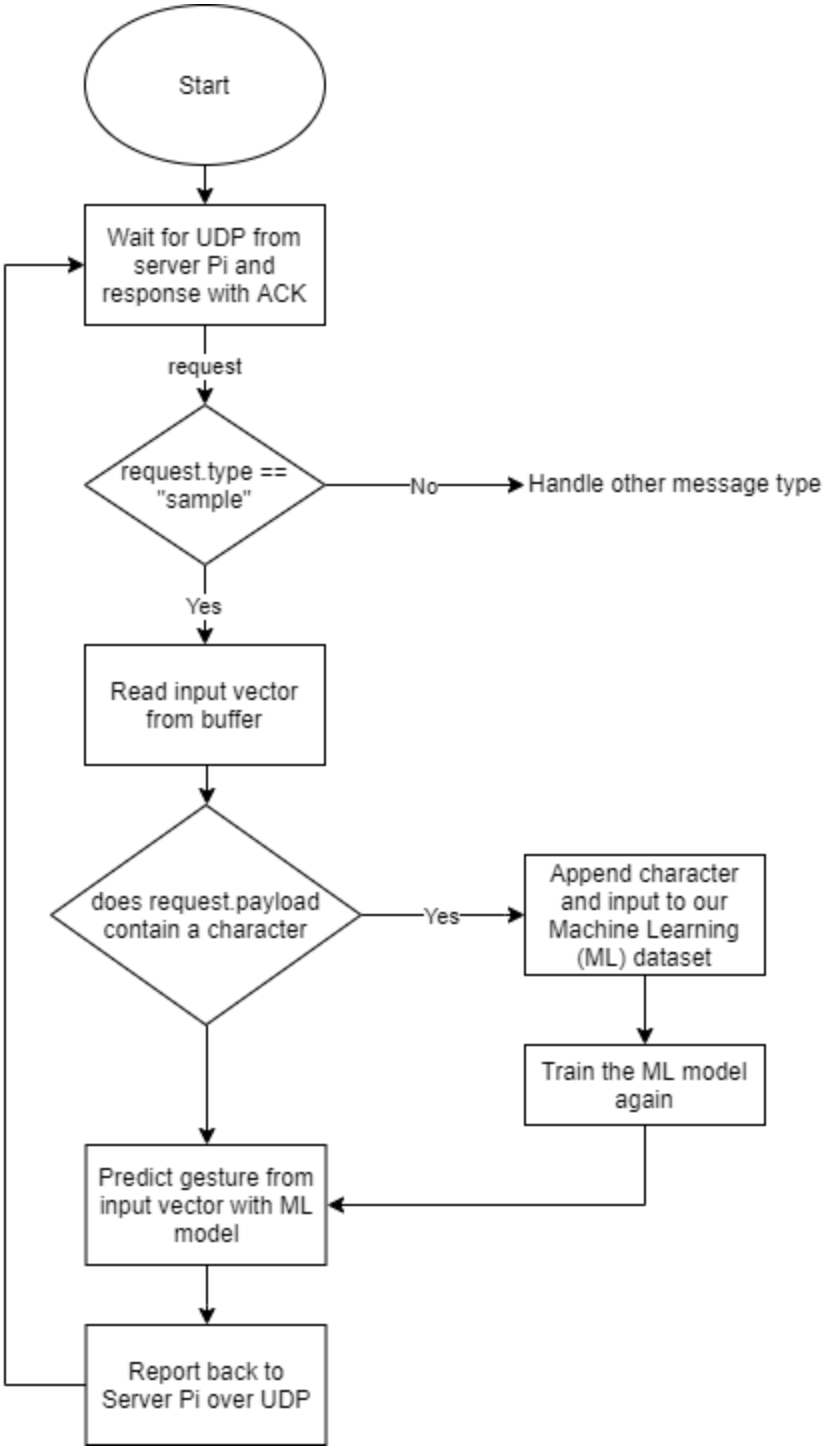


Figure 14: Machine Learning Pi Functional Sample Algorithm with Sample Message Type