Mobile Robot Trained to Recognize Commands in American Sign Language

Christian Allen
Undergraduate, Utah State University
chin.allen9@hotmail.com

Jessyca Allen
Undergraduate, Utah State University
jessycalynn9@gmail.com

Thomas Baskin
Undergraduate, Utah State University
tomcbaskin2@gmail.com

Gary Fehlauer
Graduate, Utah State University
garyfehlauer@gmail.com

Tyler Kunz
Undergraduate, Utah State University
kunztr18@gmail.com

Pablo Moreno
Undergraduate, Utah State University
pablomoreno555@gmail.com

***Abstract*—The following robot is a mobile, recognition based skid-steer tank. This robot can be used to perform simple movements based on its recognition of an alphabetic sign from American Sign Language. It currently recognizes 24 letters and performs six actions.**

I. Introduction

This report will outline the design construction and testing for our ASL Recognition Robot. ASL is used as a way of controlling and interfacing with the robot while providing insight into the practical limitations of communicating with a mobile robot in real time using nonverbal methods. The desired outcome for this project is to present ideas regarding how to best communicate with robots using nonverbal commands that are not limited to the use of an application.

A. Nomenclature

- ASL - American Sign Language
- 4wd Breakout Board - Provides access of the electrical components to the Raspberry Pi
- ROS - Robot Operating System
- VNC - Virtual Network Computing
- PWM - Pulse Width Modulation
- OOP - Object Oriented Programming

II. Objective/Methodology

A. This Robot:

- Recognizes ASL Alphabet Signs
- Moves based on a hardcoded command

B. Project Restrictions:

- Alphabetic ASL Signs Only (Excluding J and Z)
- User has to reorient themselves in front of the robot
- Robot needs to be in a large area as it doesn't have edge or collision detection

C. Aim

The objective of this project is to better understand robot intelligence and gain experience in the field of robotics. The different disciplines of robotics have an expansive depth of content in terms of algorithms, technology, frameworks, and applications. For the ASL robot, the goal was to implement computer vision, machine learning, and basic mobility into a ROS architecture and effectively create a mobile robot to recognize and react to the ASL alphabet. With so many emerging technologies centered around voice recognition, the ASL robot project explores the possible avenues of gesture recognition, not only in a mobile robot, but as an inclusive alternative to any technology that caters to vocal recognition and interaction.

III. Machine Learning

A. Objective

The objective of the machine learning model is to classify an ASL hand gesture with the correct letter it represents. Since the robot would need to recognize hand gestures in a variety of environments, a properly-trained machine learning model should work well to generalize the environments in which it can recognize particular hand gestures. However, the breadth of situations that the model can recognize an ASL hand gesture is limited by the dataset.

B. Tools

The Keras sublibrary of Tensorflow was utilized for the creation and training of the model. This library allows for faster training and testing through GPU integration. Further, the Keras library has convenient tools for dataset augmentation such as mirroring and random sampling. Mirroring in particular is a useful training tool with recognizing ASL gestures, since left-handed people will sign with their dominant hand.

The dataset was provided through Kaggle [1], containing 87,000 images across 29 classes. These classes include the letters A-Z plus gestures for 'space', 'delete', and a negative class 'nothing' without any hand in frame. For this project, however, the extra three classes as well as letters 'j' and 'z' were omitted. For this projects purposes, the extra classes were unnecessary. The ASL gestures for 'j' and 'z' both require hand movement, which doesn't work well with a static image.

The quality of the dataset raised some concerns. Many of the images in the training dataset were taken in poor lighting, where the hand is devoid of any detail. All of the training

images were taken against the same background, giving no assistance to generalization.

C. Architecture

Rather than experiment with a novel model architecture, training was done with a known image classification architecture. It was found to be highly successful at the time as ResNet provided excellent classification ability while attempting to reduce the number of parameters over its competitors. ResNet was an ideal candidate for ASL classification.

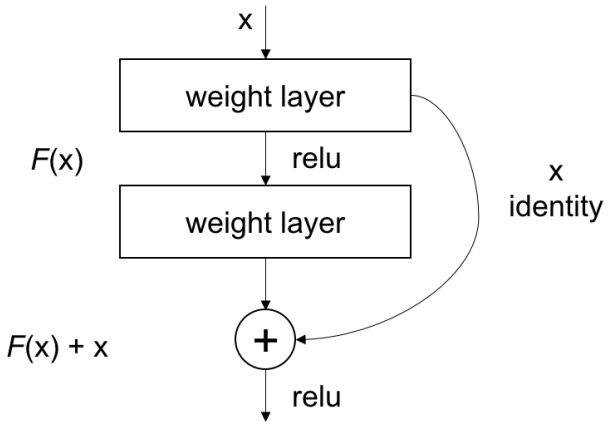

Fig. 1. ResNet Residual Layer

The name ResNet refers to the innovation of the architecture: the residual layer. Shown in Figure 1, the residual layer feeds forward with both the weighted input and the unmodified input. The residual connection allows the model to learn more information at each layer without adding too many extra parameters that would otherwise be required.

In practice, the residual block is composed of 3 convolutional layers. The ResNet architecture has multiple variants, each with a different number of internal residual blocks. For this project, experimentation was done with 50, 101, and 151-layer variants of ResNet.

D. Results

The model was very successful at learning the training dataset. With a learning rate of $.005$, the model was trained on $\frac{1}{10}$ of the dataset per epoch for a total of 30 epochs. Using categorical crossentropy loss, the model achieved a training loss below 0.1 after about $20 - 25$ epochs.

To test if the model could recognize ASL gestures with a webcam the testing dataset was composed of a few dozen webcam images where ASL gestures were made in the center of the frame.

The model was not able to generalize from the training data, however, and had little over 5% accuracy on the test images. The low accuracy can likely be accounted for by the quality of the dataset. The accuracy did not improve with different ResNet architectures nor with increased training epochs. Given more time the models would have had more training on different datasets. Unfortunately for this particular area in robotics the datasets found have proven to be disappointing.

IV. Mobility

Fig. 2. Yahboom G1 AI Vision Smart Tank

A. Hardware

The Yahboom Tank, shown in Figure 2, is a skid-steer style robot with tracks controlled by four motors, two for each side of the tank where one controls the forward motion and one controls the reverse motion. The electrical current to the motors is controlled by a 4wd breakout board. This is connected to a Raspberry Pi with a 40 pin cable to interface with the software.

The Raspberry Pi is a mini computer of varying degrees of power. A model 4 was used for initial development and testing of the mobility processes. The model comes with a 1.5 GHz quad core 64-bit processor and 2, 4, or 8 GB of ram. The model 3B+ was used for development of ROS[1] and integration testing. This model comes with a 1.4 GHz quad core 64 bit processor and 1 GB of ram. There is a noticeable difference in power when running full desktop OS between the two Pi boards. The model 3B+ ran slower when a connection to the internet was made as well as running VNC and controlling the robot.

B. Software

The mobility was developed and tested using Raspberry Pi OS. This is a full desktop OS provided by the Raspberry Pi corporation. VNC software was used to connect to the Raspberry Pi from a laptop for mobile testing. After initial movement tests were successful, the mobility program was integrated into the ROS architecture which used Ubuntu Server 16.04.

The mobility program was written in Python 3. It used the package RPi.GPIO to communicate with the motors. It is a

[1]See Controller Section

high-level library that can attach to a designated set of General Purpose Input Output(GPIO) pins on the Raspberry Pi and control the outputs or inputs to those pins. For the Yahboom tank, there are six pins for the wheel motors:

TABLE I
YAHBOOM TANK MOTOR CONTROL PINS

Pin Number	Purpose
20	Left forward motor
21	Left reverse motor
16	Left motor velocity
19	Right forward motor
26	Right reverse motor
13	Right motor velocity

All of the pins were set to output and thus sent out a voltage to the motors. The forward and reverse pins are turned on or off by giving it an attribute GPIO.HIGH or GPIO.LOW, respectively. The velocity pins are passed into the class GPIO.PWM along with a frequency. PWM stands for Pulse Width Modulation. It works by sending a signal at the given frequency, turning the motor on and off. This controls the velocity of the forward or reverse motor. To change the velocity, the duty cycle of the PWM object can be changed to reflect the percentage of the frequency being used to control the velocity of the motor.

The program was written with an Object Oriented Programming(OOP) approach. A class named `Mobility` was created and takes six parameters, all of them being a pin number that correlates to the different motors. The class has built in functions that move the robot:

TABLE II
MOBILITY COMMAND TABLE

Function	Action
forward	Moves the robot forward
backward	Moves the robot backward
turn_right	Turns the robot right 90 degrees
turn_left	Turns the robot left 90 degrees
go_right	Turns the robot right, moves it forward, and reorients
go_left	Turns the robot left, moves it forward, and reorients
square	Moves in a square pattern
spin	Spins in place
motor_test	Tests each motor sequentially
clean_up	Performs the GPIO.cleanup function for shutting down the program
reset_pins	Resets the pin designation to new pins

The class can be instantiated into an object and the movement functions can be called for usage. During initial testing, `mobility_test.py` was used. It takes a letter input from the console and, given an appropriate letter, the robot performs the coded action.

C. Issues

The following obstacles were encountered while interfacing with the tank during development of the mobility portion of the ASL Recognition Robot.

1) Remote connection with the image provided by Yahboom: The image provided by Yahboom was difficult to work with. It is based off of the Raspberry Pi OS with preloaded code and programs used for the Yahboom tank. The network configuration was broken, as such was the ability to connect with the robot via VNC. The SD card that was used was flashed and reflashed about a dozen times, each time the network configuration files were changed. Setting up Ethernet and wireless connections as well as a hotspot connection was attempted to test that the Pi's connectivity or home internet wasn't the issue. A new SD card was purchased as a test to find if the old card was fundamentally broken. Both SD cards were in working order. The solution was to use the Raspberry Pi OS image and work without the preloaded code.

Remote connection with the fixed Pi was also a challenge. Utah State University's WiFi requires connected devices to have registered MAC addresses on their networking website. To remotely connect to the Pi after the MAC addresses (eth0 and wlan0) have been provided to USU the following steps should be taken:

```
nmcli d wifi list –rescan yes
nmcli radio wifi on
nmcli –a d wifi connect BLUEZONE
```

Then from the device remotely accessing the Pi:

```
sudo ssh ubuntu@[IP ADDRESS]
```

It was important to connect to the robot remotely so that during testing the robot could be controlled and shutdown from another device.

2) Loose wheels on the motors: Once connected to the Yahboom tank, the tracks were not mobile. Instead of moving in their intended fashion, they would jitter or not move at all. The troubleshooting steps that were taken were checking to see if the motors were actually working by taking off the tracks and running the code from the `mobility_test.py`. The motors worked properly after the test. Next was testing the connection between the wheels and the tracks. It was found that the wheel that moves the tracks was not tightened down to the motor, causing slippage of the tracks whenever the motor was running and thus causing the tracks to fail.

3) Track Slippage: Due to the nature of the movement, the tracks performed to a varying degree based on the surface it is performing on. The robot performed better on carpeted areas over areas covered in linoleum or tile. However, heavily carpeted areas caused the robot tracks to grip the carpet and get stuck. This is a working problem.

V. CONTROLLER

A. Objective

The controller is the software responsible for connecting the various pieces of the robot together. The controller needs to deliver information from the hardware to the software and back again. This robot uses ROS as a controller, chosen for its featured systems for communicating between each part of the project.

ROS uses a publisher-subscriber model which conveniently abstracts much of the necessary networking and hardware knowledge that may otherwise be required. ROS allowed for the development of the mobility and vision modules in separate python scripts which are then loaded when ready for execution by the controller.

B. Mobility

The mobility module of the controller is responsible for receiving a prediction from the vision model and executing some sequence of instructions on the robot hardware. The majority of the complexity within the mobility module is abstracted to a mobility object which does all the communicating with the tank's hardware. Thus, the mobility module of the controller receives a prediction from the vision module and executes the corresponding sequence of mobility instructions.

C. Vision

Using the robot's webcam, the vision module is responsible for identifying ASL gestures. The implementation of the vision module promised to be simple, only requiring a pretrained machine-learning model to make predictions on each webcam frame as they are delivered by the controller. The robot was successfully passing images between publisher and subscriber nodes. The vision module is dependent on Tensorflow to load the pretrained model for prediction on the webcam images.

D. Issues

1) Tensorflow not supported by Python2: The decision to use ROS was made rather early on in the life of the project. This had some consequences that did not become clear until efforts were made to fully integrate each piece of the project. One of the consequences was the limitation on what environment was usable. Python 2 was used in order to maintain compatibility between the operating system of the Raspberry Pi and ROS. Unfortunately, it was found during integration that Tensorflow no longer supports Python 2. Since the machine learning model was dependent on Tensorflow to load and predict, an impasse was reached for implementing the vision module.

Attempts were made to bypass the Python requirements in the weeks preceding the deadline, however they were unsuccessful in fully implementing the vision node; However, as a proof of concept, simulations of dummy predictions from the vision node allowed for instruction of the robot to execute certain commands.

2) Attempted Solutions to bypass Tensorflow: The initial project design used three nodes: the webcam publisher node, the Tensorflow image recognition publisher/subscriber node, and the mobility subscriber node. Images taken in the first node would be passed through the nodes and decisions would be made on those results. An idea to fix the vision node (i.e. getting Tensorflow running) included creating a blocking subprocess in the vision node and sending the images received from the webcam publisher node to a virtual environment containing Python3 and Tensorflow but Tensorflow would not properly download onto the Pi. Wheels are Python installation tools that help with updating, installing, and downloading Python packages. A common error encountered when working through this project was the following: "ERROR: grpcio−...−...−...−.....−...−x86_64.whl is not a supported wheel on this platform." This indicates that either another wheel is needed or this problem needs to be solved in another way. Other wheels were installed – none worked. Tensorflow Lite would likewise not install in the virtual environment. After these failed attempts to run Tensorflow on the Pi time to solve this issue had run out. A different possible solution would have been to make a websocket and send the images over to another computer and then respond with the corresponding ASL command. The components worked independently on different machines but ran into issues on the robot itself.

3) Lighter Operating System: Visualizing the webcam output was a challenge. Ubuntu 16.04 was downloaded on the Raspberry Pi. This version of Ubuntu can run ROS Melodic. Additionally, the Raspberry Pi 3 architecture requires an older version of Ubuntu because newer versions (20.04) are not supported and will cause the Raspberry Pi 3 to fail on boot up. To save space on the SD card a lighter version of Ubuntu 16.04 was downloaded. This presented some challenges. It took some discovery and exploration to find out how to use more than one terminal at a time. To add a new terminal or access an existing one, type `Ctrl-Alt-Fn-{F#}`. To access images captured by the webcam, GNOME was installed along with other dependencies so that the `eog` command could be used. When the `eog` command was running, the `Ctrl-Alt-Fn-{F2}` window would show the webcam output – assuming the window had not previously been made into a new instance of a terminal. Before `eog` could run, the DISPLAY environment variable had to be set. By default, DISPLAY had no value. It was empty. After some discovery, the display was set to the correct value using the command `export DISPLAY=:1`. After altering the webcam publisher node – with GNOME now installed – the output of the publisher could be seen in the `F2` environment on Ubuntu.

VI. Program

Here is the GitHub repository with the code developed for the project:
https://github.com/pablomoreno555/robotics-project-ros

This link takes you to a demo video uploaded to YouTube:
https://youtu.be/XdM5NlTFyPE

VII. Conclusion

This report has shown that while challenging, non-verbal interaction with a robot is possible but does require more time and testing than what could be committed this semester. While a number of different preexisting datasets for ASL exist in competition with each other, one was not found to be more effective than another. Instead they proved to be difficult to work with and their promised results were unachievable. Work

needs to be done on alphabetic ASL recognition that shows both successful and repeatable results. Working with these different datasets is enlightening to those who use them to just how far behind inclusivity is in the world of technology and robotics. Despite all the setbacks the work done on this project shows that this idea is possible and worth working on. Even without all of the necessary experience, knowledge or tools and a strict time constraint progress was still made by showcasing what already exists and how it works towards and against this objective while highlighting areas where improvements can be made.

REFERENCES

[1] Akash, ASL Alphabet, https://www.kaggle.com/grassknoted/asl-alphabet