Customizable Gesture-Based Assistive Control System

•••

Andrew Ho, Tu Yu Hsien, Jessica Bojorquez

Findings and Setup

Overall Project Goals and Specific Aims

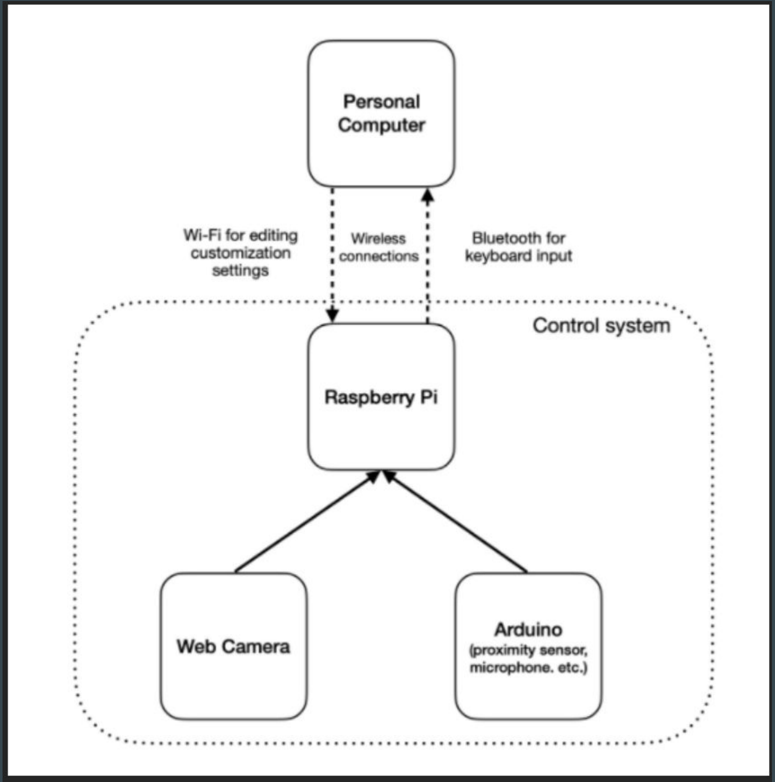
I. Create an alternative assistive method for gesture based input in personal computer
II. Collect sensor data from Arduino and USB camera to process in Raspberry Pi in real time
III. Train model to recognize specific hand gestures through web camera
IV. Send keystrokes of mapped gestures from Raspberry Pi to computer

Findings

- ARMv6 700 MHz single-core processor
  - Uses SD card to hold the Operating System
  - 32 bit Operating system Raspbian (Linux lightweight Operating System based on Debian)
- Raspberry Pi can be slow and cannot get fast results
- Our research advised to run a pre-built deep learning model
- Opted to train model in a more powerful laptop for higher speed and accuracy
- Python is a slow language for computer vision purposes
  - Due to its global interpreter lock. (but makes use of C libraries, which helps with speed)
  - It's friendly syntax and **large choice or libraries/algorithms** has made it very popular for this application
  - OpenCV offers more than 2500 optimized algorithms
- Some version of these libraries do not run on every OS due to the vast amount of hardware resources they use
  - Limited us on library versions

System

- Components
  - Raspberry Pi (the central control system)
  - Arduino Nano 33 BLE Sense
  - USB Web Camera connected to Raspberry Pi
  - BLE and Wifi Connectivity
- For user IO when working on the Raspberry Pi
  - Monitor
  - Keyboard
  - Mouse


Personal Computer
Wi-Fi for editing customization settings
Wireless connections
Bluetooth for keyboard input
Control system
Raspberry Pi
Web Camera
Arduino (proximity sensor, microphone. etc.)

Technical Approach

I. Arduino is a trigger

II. Camera takes video

III. Raspberry Pi takes inputs from these and runs model

IV. Raspberry Pi outputs predicted keystrokes to computer

V. Model is trained in personal computer

  A. Model structure saved in a json file

  B. Weights is saved in .h5 file

  C. Transferred to Raspberry Pi

Integration with Raspberry Pi

Different environments introduced some challenges.

Packages

Jessica	Andrew	Robert
Raspberry Pi	Raspberry Pi	Laptop
Python 3.7 OpenCV 4.1.0 Tensorflow 2.2.0 Keras 2.4.3 Scipy 1.4.1 Numpy 1.18.5	Python 3.7 Opencv-contrib-python 4.1.0.25 Tensorflow 1.14.0 Scipy 1.0.0 Numpy 1.13.3	Python 3.6 OpenCV 4.41 Tensorflow 2.3.1 Keras 2.4.3

Challenges

I. Different environments to run same software

  A. Required us to save and serialize our model so that Raspberry Pi would predict gestures. Libraries:

```
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import Model, Sequential
```

II. Different resources from our hardware

  A. Amount of cores

  B. Memory available

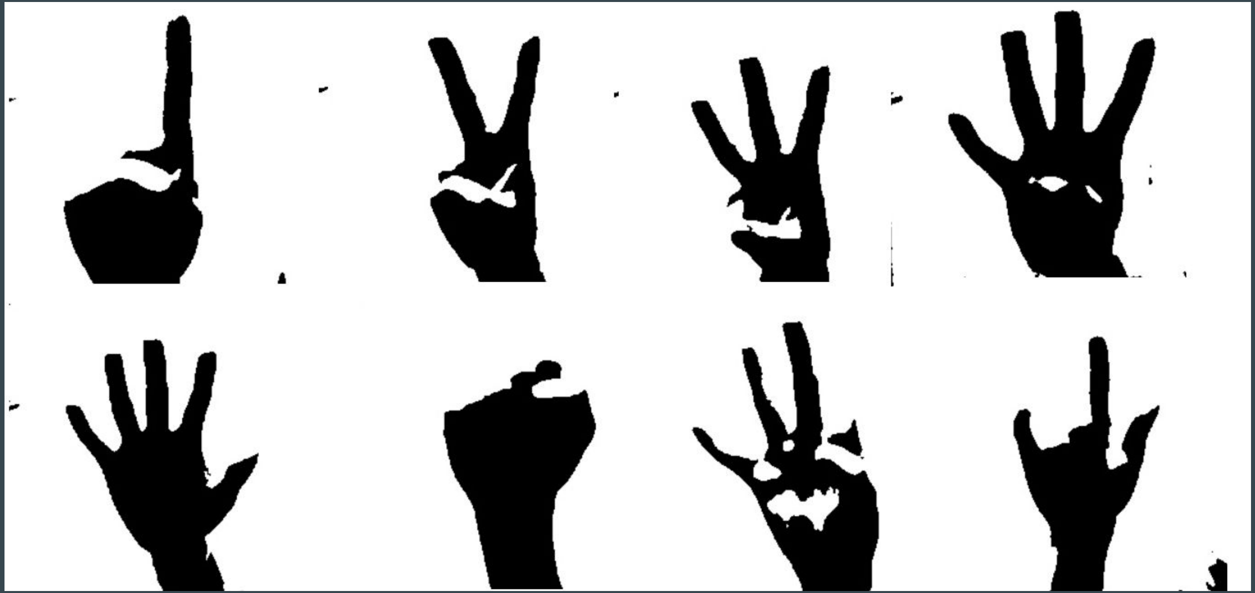
  C. Speed for installations

  D. Camera quality

Algorithm in training the model

1.Defining our hand gestures categories

For our project, we will take 8 gestures to recognize



2. Collect image from training

-> using webcam and Opencv library to perform the image

-> turning the picture into grayscale

-> store the trained image to “training_images” folder, 2500 each

3. Training the model

Some parameters we will use for training:

1. Nodes : The numbers of categories we are trying to predict.

2.Input Shape : The size here is 128 x 128

3.Number of Epochs : The number of time that the entire dataset go through the training model, here is 20

4.Convolution layer: Using the Conv2D here. It map the characteristic of different images and classify them for recognition

5.Pooling : The input is converted to a low resolution version and only keep the significant details to reduce the calculating time

3. Training the model cont.

6. Activation Function : Using the ReLU function here. It returns zero for negative inputs and the value itself for positive inputs

7. Dense Function : Classify the processed gesture signal to different folder

4. Testing the model

->Use webcam again to capture the image, and then load the trained model.

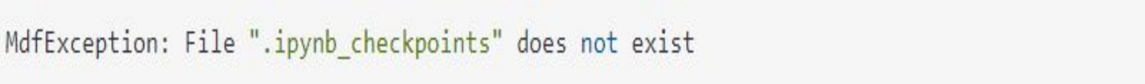
->See if the image is correctly recognized by the model.

-> Demo video:

https://youtu.be/z1YLB1_-mTo

Challenges

-> .ipynb_checkpoints in the hidden file when taking images



```
MdfException: File ".ipynb_checkpoints" does not exist
```

-> vulnerable to the environment

Integration and Future works

Arduino: System Trigger by Sound or Proximity

- The hand recognition is triggered by a snap or a wave near the camera
    - Prevents false positives in detection
- Using onboard sensors on the Arduino Nano 33 BLE Sense
    - MP34DT05 microphone sensor
    - APDS9960 gesture, light, and proximity sensor
- Trigger thresholds are set for both sensors
    - RMS of microphone reading: >120
    - Value of proximity reading: <220 (out of 256)
- Arduino prints to serial when threshold is met
    - Read by Raspberry Pi through serial

Raspberry Pi: Hand Gesture Recognition and Keystroke Sender

- Initialize web camera capturing, Bluetooth keyboard emulator, establish bluetooth connection, and identify the serial port that Arduino is using
- Run main looper:
  - Check Arduino sensor readings through serial connection
  - If gesture detection threshold is triggered, start image capture from web camera
  - Run captured image through hand gesture model
    - Patience: number of consecutive readings to obtain final returned gesture
    - Timeout: stop detection after a number of tries
    - Both parameters can be tuned
  - Identify keystroke to broadcast from gesture mapping
  - Send keystroke to personal computer through Bluetooth keyboard emulator

Future Works

- Automatic adjustments and dynamic parameter tuning during setup
    - Microphone: detect ambient noise level and set as baseline
    - Proximity: observe user distance and set distance to trigger
    - Web camera: adjust brightness and filter thresholds to ensure clear hand segmentation
    - Bluetooth: auto-connect to previous connection
- Improve hand gesture recognition
    - Additional hand gestures
    - Inclusion of skin segmentation
        - Non-black, noisy background hampers hand recognition accuracy