Lumos: Smart Wand for Home Automation

Ashwin Kumar*
ashwinkumar@wustl.edu
Washington University in St. Louis

Sayali Patil*
sayali.patil@wustl.edu
Washington University in St. Louis

Rajagopal Venkatesaramani*
rajagopal@wustl.edu
Washington University in St. Louis

ABSTRACT

Modern smart home assistants (Alexa, Google Assistant, Siri) aim at providing increased comfort to humans through automated operations. However, they rely heavily on voice commands for real-time smart home control. Unfortunately, being restricted to using voice commands or complicated user interfaces to trigger such devices makes such interactions difficult for people with speech impairments, elderly people who find it hard to remember commands, or novice users. Through Lumos, we propose a gesture-based input device in a portable form factor to aid elderly and disabled people. Using a combination of inertial measurement with an accelerometer and gyroscope and motion tracking using an infrared (IR) camera, we demonstrate the design of a system that can accept gestures as inputs.

Using the AWS IoT core interface, we link identified gestures to smart home devices to allow for a more intuitive and seamless control. Our solution is quicker in nature as people won't have to wait for smart assistant's response to a vocal trigger and more intuitive than using tedious smartphone applications. Most importantly, this enables simple smart- home interactivity for people with speech-impediments, or disabilities that otherwise prevent them from interacting with smart-assistants conventionally.

CCS CONCEPTS

• **Computer systems organization → Embedded systems**; Sensor networks; • **Human centered computing** → Accessibility.

KEYWORDS

Gesture Recognition, IMU, IR Camera, Neural Network, AWS

ACM Reference Format:
Ashwin Kumar, Sayali Patil, and Rajagopal Venkatesaramani. 2021. Lumos: Smart Wand for Home Automation. In *Proceedings of* . ACM, New York, NY, USA, 6 pages. https://doi.org/

1 INTRODUCTION

With the advent of WiFi-connected smart homes and IoT devices, people have increasingly started relying upon smart assistants like Google Assistant, Alexa and Siri for everyday automation and control. These assistants are built around the use-case of voice activation and interaction. This may be a problem for users for several reasons - persons with speech impediments, users in noisy environments and non-native speakers often struggle with getting a command accurately recognized by smart assistants. The alternative is to use smartphone applications to carry out the same task, but these suffer from the lack of the ease-of-access that single commands would provide, as well as being non-intuitive and hard to navigate.

To this end, we propose Lumos: a smart wand for home automation - a device in a portable form factor which allows users to trigger home-automation events with simple and intuitive gestures. Using a combination of motion detection using an on-board Inertial Measurement Unit (IMU) and motion tracking with an InfraRed (IR) camera, we integrate two separate approaches to gesture recognition into a single form factor, while recognizing the advantages of each approach over the other.

2 GOALS

For creating the gesture recognition based home-automation device, we had the following goals:

- It should be in a portable form-factor.
- Gestures should be intuitive and easy to perform.
- Power consumption must allow for reasonable runtime.
- The implementation must be cost-effective.
- Must allow a large number of discrete as well as continuous inputs.

Armed with these goals, we proceed to the design aspects of the smart wand.

3 DESIGN

In designing Lumos, we approached the problem of gesture-based smart-home control using two independent methods.

- Using an on-board Inertial Measurement Unit (IMU) to recognize motion, translate that into gestures and transmit that information to the cloud.
- Using an InfraRed (IR) camera to track the motion of the wand and parse it into known gestures before transmitting to the cloud.

Both methods have their pros and cons, which are discussed in the following sections. These methods would eventually communicate recognized gestures to the cloud, which can then be used to control smart devices.

3.1 IMU-based gesture recognition: Flick

The first method used for gesture recognition is based on directly sensing the motion of the wand and mapping that data to a set of gestures, henceforth called *Flick*. This involves reading accelerometer and gyroscope information from an IMU located directly on the smart-wand. Using such an input system, we can create a self-contained gesture input system that is also wireless.

This necessitates the following design considerations:

- The wand should have an on-board IMU and communications module to transmit acceleration and gyroscopic information.

*All authors contributed equally to this research.

- The wand should have an on-board power supply to power the IMU and the communication module.
- Because wireless communication is very expensive in terms of power draw, the wand should also have some on-board processing to allow parsing of gestures, so that minimum packets are sent wirelessly.

For Flick, we identify the following pros and cons:

+ The solution is completely wireless and self-contained
+ It allows smart device control without being in proximity of any smart assistant or even any camera.
+ It preserves privacy, as voice or video are not captured or transmitted
- It requires an on-board power supply, hence has limited life
- IMU input can be noisy, which may lead to accidental triggers or missed inputs.

We address the limitations by doing the following. We use a rechargeable battery as a power supply, with an off/on switch to further save power. For limiting accidental triggers, we provide a control to signal the start and end of input, and for reducing noise, we use moving average filters.

3.2 Motion tacking using IR camera: Flow

The second proposed method for gesture-recognition, the implementation of which is completely independent of Flick, uses a fixed InfraRed camera, and a portable device (merely any object, called the wand) with a reflective tip which is tracked using computer vision methods. We call this method *Flow*. As opposed to the linear motions captured by Flick, Flow allows us to capture arbitrarily complex 2-dimensional gestures, and in theory, an unlimited number of such gestures. Flow necessitates the following design considerations:

- The wand must reflect IR light back to the camera, as a narrow beam, to be detected as a point by tracking techniques.
- The module running computer vision algorithms must have sufficient processing power, to avoid streaming video over the internet.
- An IR illuminator must emit IR light directly away from the camera lens, to be reflected back.
- Visible light must not be captured by the system, to avoid privacy concerns. This must be a hardware solution to avoid potential breaches.

For Flow, we identify the following pros and cons:

+ Support for a large number of complex gestures.
+ Camera and processing module are hardwired, wand requires no power, and therefore minimizes the hassle of charging.
+ Relatively robust to noise in inputs.
- Computer Vision algorithms are computationally taxing.
- Restricted to line-of-sight, as opposed to Flick, which uses on-board power and sensors.
- Sensitivity to ambient light.

To address the privacy concern of having a camera that is always on in an indoor space, we install a physical visible-light blocking filter (IR-pass filter, 780nm) in front of the camera. Only IR wavelengths pass through this filter, which is not enough to produce a full video feed. This solution is robust to any manipulation by external actors who would not have physical access to the device.

3.3 Communication and control

After gesture input is processed, the communication and IoT aspect for both Flick and Flow remains the same. Processed data is sent via WiFi to an IoT web client. Once received, it is passed through a gateway or an API that can interface with the required smart devices. Since such smart devices typically operate over WiFi, an internet based system should be able to efficiently convert inputs to mapped manipulations of various devices connected to the smart home.

For the communications and control system, we had the following requirements:

- It should use WiFi and internet protocols to communicate information
- The raw input should be parsed locally without transmitting it over the internet for both privacy and power concerns.
- There should be a convenient interface for smart device control that can be mapped from inputs.
- There should be minimum processing done online, to reduce lag.

Combining all of these design considerations, we propose Lumos, a smart wand capable of recognizing gestures through both camera-based and IMU based motion tracking and controlling smart devices wirelessly and wordlessly. The implementation for the same is discussed in the following section.

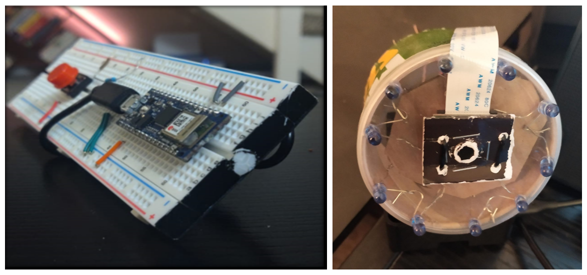

Figure 1: Lumos: Final Form Factor; IR camera and Illuminator

4 IMPLEMENTATION

Figure 1 shows the final form factor, the IR camera and the illuminator circuit. Below, we discuss the technical aspects of the implementation.

4.1 Flick

For implementing the Flick design, we required a small form factor micro-controller with both an IMU for detecting motion and a WiFi module for communicating with AWS IoT. The Arduino Nano 33 IoT board fit this criteria perfectly. We configured the Nano to connect to an AWS IoT shadow we created, and to connect to our WiFi network.

For the physical circuit, we used the circuit as outlined in Figure 2. A push button was connected to digital pin 2 in a normally off

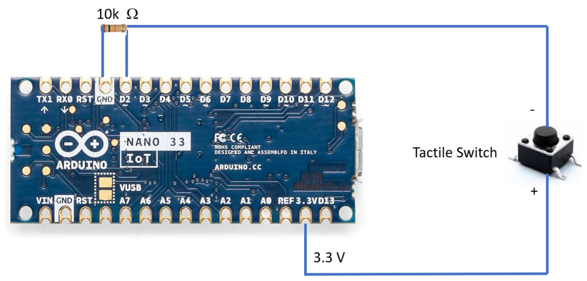

Figure 2: Wiring diagram for the Flick circuit

Table 1: Current draw for various active components

Component	MCU	IMU	WiFi module	Total
Current	4-6 mA	1.25 mA	30 mA	~38mA

state, through a pull down resistor. For the power supply, we used a rechargeable 18650 Li-Ion battery (2850mAh) with a USB module to connect it to the Nano. This allows for easy removal and recharging of the battery as and when required.

The current draws for various components are shown in Table 1, which gives us a run-time of around 75 hours on a single charge of the selected battery, without any active power-saving techniques.

The circuit is mounted on a long prototype board (the wand), which can be held in hand. The Nano board is placed farthest from where the hand will be, to ensure the IMU is able to sense the maximum range of motions when the tip is moved.

On receiving power, the board first looks for a WiFi signal, connecting to the SSID saved in its configuration files. Once connected, the onboard LED starts flashing while it tries to pair with the MQTT Client at AWS. A solid LED signals a successful connection, after which the wand is ready to accept input. The startup routine typically takes between 5 and 15 seconds.

When the user presses the button, it signals the board to start listening for IMU events. While pressed, the accelerometer and gyroscope readings are saved into global variables that keep track of the moving average for each of the 6 possible inputs, with a weight specified for the incoming value (10% by default). This allows us to filter some noise out of the IMU readings and smooth the input.

The period during which the button is kept pressed is called the "active" period, where motions made by the user can be detected. When the button is released, the collected information is parsed and relayed through the MQTT Client to the shadow at AWS IoT.

There are two possible motions that can be detected by Flick:

4.1.1 Flicks. : As the name suggests, we can detect quick motions made using the wand. For detecting gestures, we use threshold based triggers on the gyroscope data. When the input crosses the threshold (i.e., when the wand is rotated fast enough on one of the axes), the specified gesture is recognized and added to a queue. The moving average filtering ensures that we only detect a spike if it is of a certain minimum duration or intensity, getting rid of momentary threshold crossings. Since there are three axes of rotation (Figure 6) and each can be triggered in two opposing directions in addition to the neutral position, we use a trio of tribits to encode which

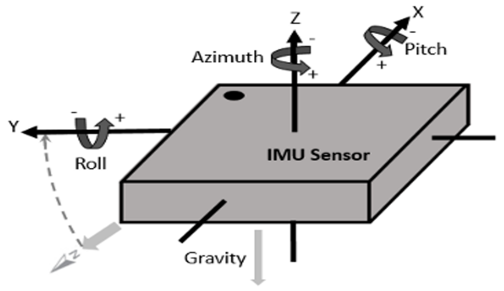

Figure 3: 6 Degrees of Freedom for the IMU

flick was detected into an integer. Each tribit can hold a value of 0 (negative threshold crossed), 1 (neutral) or 2 (positive threshold crossed). The ordered trio of tribits can then be converted to an integer in a similar fashion as binary numbers, except using powers of three, to give us a number between 0 and 26. A detection of no flicks would give a value of 13, and (12, 14), (10, 16) and (4,22) are the pairs of opposing flicks for each of the three axes of rotation.

While it is possible to cross the threshold in more than one axes simultaneously, such motions are not very intuitive to perform. Thus, to account for the lost number of possible inputs, we enable chaining of up to three such flicks as part of the same gesture. This gives us 6^3 possible inputs, which can be mapped to any smart home control.

4.1.2 Continuous Input. : Sometimes, it is also necessary to have a continuous input variable to control events such as the brightness of lights or the speed of a fan. We used the accelerometer data to enable one such input for this implementation. Due to gravity, the IMU is always under acceleration, and gives us readings even when the wand is perfectly still. We use this ability to determine the orientation of the board with respect to the ground, using the detected acceleration about each axis to extract the angle. This is then mapped into a percentage value, 0% when the wand is pointing straight down, and 100% when it is pointing up.

Once the button is released, the data collected is parsed to send to the shadow created on AWS IoT. The data is parsed into a message buffer string first, before it is published. If flicks are detected, the message begins with "flicks_" followed by the parsed tribit integers separated by underscores. If no flicks have been detected, the Nano uses the collected accelerometer information to get the angle. The message begins with "percent_", followed by the percent value. We use this underscore separated format to separate the data and detect the type of input on the AWS side.

Using the ArduinoJSON library, the information about the device, sensor and the data is serialized and published as a JSON object to the "arduino/outgoing-gyroscope" topic. The message is passed as a string in the "data" variable of the JSON object.

4.2 Flow

To implement Flow, we required a compute-device, which was both powerful enough to run computer vision techniques, as well as not draw too much power. We found a Raspberry Pi 3 Model B a perfect fit, with its 64-bit quad-core CPU and 1GB of RAM sufficient for

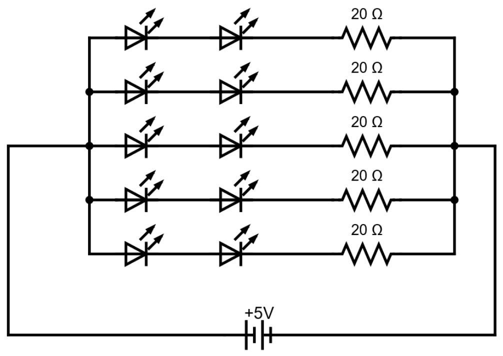

Figure 4: Wiring diagram for the illuminator circuit

the most part. Additionally, the Raspberry Pi 3B has an on-board wireless LAN module, so no network hardwiring was necessary. We configured the Raspberry Pi to connect to our WiFi network at startup, and launch the python script that runs the back-end computations. We also registered the Raspberry Pi as a thing on AWS IoT, and installed the AWS IoT Device SDK on the Pi.

The camera used for our implementation was a Raspberry PiCam NoIR - which is simply a Raspberry Pi Camera with its IR filter removed, allowing IR wavelengths to pass through. In conjunction with the visible light filter installed externally, the camera only captures IR light. The camera's field of view is also sensitive to the frames-per-second (fps) refresh rate set by the user, which we use to tune the trade-off between a wider field of view and computational performance.

For the illuminator, we built a circuit comprising of 10 high-intensity IR LEDs. The circuit and the resistances used are illustrated in Figure 4. The LEDs were arranged in a ring around and facing away from the camera lens. Each LED has a 20° field of light emission; arranging the LEDs in a ring ensures uniform illumination across the camera's field of view. The IR illuminator was powered by a 5V power-supply, with a total current draw of 500mA.

Finally, to make the wand-tip reflect light straight back at the camera, we use retro-reflective tape, cut out in the form of a circle. We also ensured that the front facing surface of the wand was otherwise dark (black) in color, to avoid mis-detection by the camera.

We use Python 3, OpenCV [2] and TensorFlow [1] to implement the software for the pipeline. The method can be broken down into the following steps:

- Detect the presence of reflected tip in the frame.
- While tip is present in frame, append current position to a list of 2-D coordinates where the tip has previously been.
- Once the tip is not in view or has not moved for a set number of frames, terminate tracking. Draw the shape of the pattern by interpolation between points as lines.
- Segment the image with the drawn shape to only include the regions where movement was present, and resize to appropriate shape.
- Pass this drawn shape to a previously trained image-classifier.
- Send output of classifier to AWS, which translates it into a trigger for a pre-set action.

Video frames are first converted to grayscale. To detect and track the motion of the wand tip, we use OpenCV's *SimpleBlobDetector* method, which can filter by color, intensity, size, circularity, area, etc. The detector is also supplied with parameters to adjust the thresholds of light intensity to retain, accounting for noise from ambient light sources. As the wand moves through the field of view, 15 frames each second are captured and processed by the detector, appending the position of the tip to a running list. A line is also drawn between each subsequent point in this list using OpenCV's *line* method, to trace the shape of the motion. After a set number of frames where the wand is either not in view, or not moving, an image with the shape of the motion traced is sent for further processing.

In the processing step, the image is first segmented to only the region where the shape is present using a simple parse of the matrix for non-zero values. A padding of ten pixels on each side is added to this segmented image. Next, the image is sent to a previously trained neural network model, and a classification label is obtained. This shallow neural network was implemented in TensorFlow, comprising a Flatten layer, followed by a Dense layer of size 512 with ReLU activation, followed by a Dropout step with frequency 0.2 for regularization, and finally a softmax activation layer for 10-class classification. Because of lack of training data, we used the MNIST handwritten digits dataset [4], but in theory, one is only limited by the size of the training data in implementing more gestures. With more and more complicated gestures, the use of deep convolutional neural networks would be worth investigating. Figure 5 shows examples from the MNIST dataset.

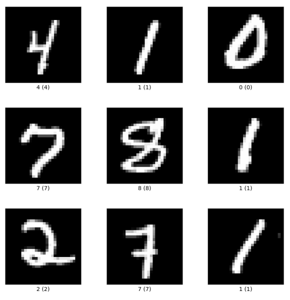

Figure 5: MNIST digits

As the Raspberry Pi 3B is not powerful enough to train neural networks locally, we trained a model on a separate machine running Ubuntu (Ryzen 7 3800X, 64GB RAM), which took a few minutes to conduct 10 training epochs over the dataset. This gave us test accuracy of about 98%. The trained model is then transferred to the Raspberry Pi, and is used only for inference. The classification label obtained is then published to an AWS Topic, after which server-side processing eventually triggers an action, as elaborated upon in the following section.

4.3 Cloud Implementation

For Flick and Flow methods to work, we need to send messages to smart devices in the house to perform corresponding tasks. This communication method is based on sending mapped gestures data by Flick and Flow to AWS IoT, parsing the data, and using the

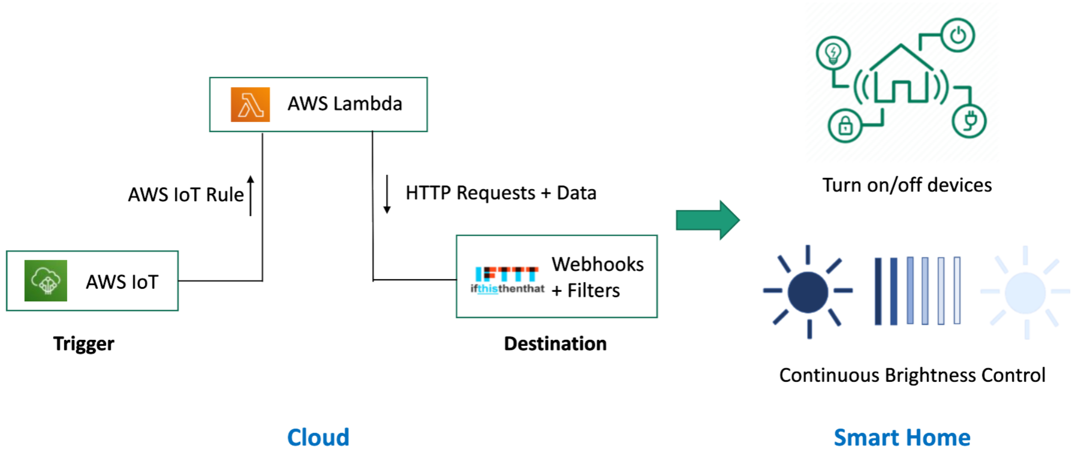

Figure 6: Cloud Implementation

parsed data to trigger smart devices. This method enforces wireless communication between the wand and smart devices.

AWS IoT implementation consists of following steps:

4.3.1 Adding devices as AWS IoT things . : For Arduino and RPi to communicate with AWS IoT we need to establish a secure communication link between them. Additionally, AWS IoT should be able to receive data in an appropriate format from Flick and Flow once the connection is established. Adding Arduino nano to AWS IoT as a thing did not need to install any additional device Software Development Kit (SDK), RPi on the other hand needed us to install AWS IoT Device SDK as there is no on-board IoT communication module on RPi.

Adding a device as an AWS thing involved creating an AWS thing with a specific name for the device. To restrict the functionality of the device, we created a policy to attach with the device. This allows only IoT specific actions to be performed by the device. To authenticate the device to securely communicate with AWS IoT, we created Transport Level Security (TLS) certificate outputting a public key, a private key, and a certificate. We attached the corresponding thing and policy of the device to the certificate. The last step here is to activate the certificate. These certificates and generated iot-endpoints for both the things were then saved on the respective devices. Authentications and endpoints were used to send data to the MQTT client in the AWS IoT console.

4.3.2 Creating Lambda functions. : We created a lambda function for both Arduino and RPi implementation which get triggered by AWS IoT rules associated with Arduino and RPi implementation respectively. Whenever the devices send any messages to the MQTT client, the associated rules get invoked and they send all the data received from the devices to the corresponding lambda functions, by triggering them.

4.3.3 Creating AWS IoT Rules. : We created rules to specify what AWS IoT should do once it receives data from any of the devices. Rules for Arduino and RPi are created to send the received data to corresponding lambda function by invoking them.

4.3.4 Data Parsing. : Once the lambda functions receive a data in Jason format, we parse the data string of interest to separate out keywords (action words). These keywords are then sent to the webhook service hosted on If This Then That (IFTTT) platform.

4.3.5 IFTTT setup. : IFTTT is a web-based service aimed at providing users the capability of executing a chain of conditional statements triggered by other web services. To trigger smart devices based on the retrieved keywords from the data sent by Arduino and RPi to AWS IoT, we make use of the webhooks service on the IFTTT platform. Webhooks are generally used to connect two different web services. In our implementation, we are using AWS Lambda as the triggering end for a webhook and smartLife app that controls smart devices as the receiving end. When an AWS IoT rules invokes the associated lambda function, the function serializes the event data and sends it to the webhook URL. To control smart devices, we created multiple applets using webhooks service and the smartLife app. When the webhook URL receives data, it runs the respective applet based on the data value, which in turn performs specific actions on smart devices at home. To add more degrees of control, such as controlling the brightness of a light bulb, we are using IFTTT filters, which is a bit of a JavaScript code. We are adding these filters while creating applets for controlling different devices. IFTTT filters allow us to define diverse range of actions based on the received event data.

Together, these services help us control our smart devices through gesture mapping. This approach allows for remote management of all the connected devices in an easy and secure way. AWS IoT provides fine-grained information on device logs which helps in keeping track of the devices. It supports the transmission of trillions of messages over the established communication link to process in real-time. Cloud implementation in our case requires minimum online data processing. All the gesture data is processed on physical devices before sending the final data to AWS IoT. One limitation of this implementation is that IoT cannot be directly used to interface with smart devices. We cannot use smart home assistants as endpoints, they can only be used as a trigger.

5 LIMITATIONS

Here we discuss some limitations of our work. For the Arduino based solution, even though there exists a continuous input, there is just one variable for control. In a real system, people might need to control multiple smart devices that can accept continuous input. Further, it was not possible to extensively test the system with a large number of devices because of logistical constraints. The implementation does work in theory, and testing can further confirm the reliability of the system.

We also note that currently, IFTTT acts as an extra intermediary between Lumos and the smart devices. It should be possible to directly interface the smart devices through AWS IoT if their API's can be accessed. Similarly, the SmartLife app used to control the smart devices is very slow to respond to complex routines. For power users, it might be necessary to create gestures that do multiple things, and that is bottlenecked by the app right now.

Finally, for the Flow approach, we note the obvious limitation of line of sight. Multiple cameras might solve this issue, at an overhead to infrastructure cost. The IR camera is also highly sensitive to lighting conditions, and while we have noise removal techniques available, the hardware at hand was not powerful enough to run them simultaneously.

6 RELATED WORK

Recent advances in gesture recognition using inertial measurement units (IMUs) focus extensively on wearable devices, essentially placing the IMU on the wearer's fingertips, or wrist. In [3], the authors propose a hand-gesture recognition algorithm using a restricted column energy (RCE) neural network (a neural network with a relatively straightforward learning scheme, with activation upon demand). The authors tweak the RCE learning metric to achieve superior recognition performance. In [5], the authors use the same Arduino board, the Nano 33, as used in our work, to recognize finer gestures at the fingertip level using several machine learning approaches. Chiefly, they achieve over 95% accuracy using Recurrent Neural Networks.

Prior literature on the use of InfraRed cameras for gesture recognition is limited in the academic domain, but several hobbyists have implemented similar gesture recognition methods [6, 7] using the Raspberry Pi NoIR camera. We used some of this existing code as a starting point, but had to make extensive changes. Our approach differs from these methods in that we conduct the OpenCV processing locally, instead of streaming the video to a more powerful server machine. Additionally, we use a neural network approach for image classification, as opposed to classical machine learning methods like SVM and k-nearest neighbors employed in these projects. Our logic for detecting the start and end of a gesture also differs significantly from prior work, where often start and end regions are defined, which are impractical to use without active feedback from a screen.

7 CONCLUSIONS AND FUTURE WORK

We showed the design and implementation of Lumos, a smart wand for gesture based control of smart devices and homes, to aid people with vocal disabilities and elderly. Using Flick and Flow, we provide users with the ability to use gesture based input from anywhere in the home (using IMU based gestures) and with an un-powered device in certain locations (using an IR camera approach). We demonstrate a proof of concept design of such a product, combining both forms of input into a wand-like form factor. We discussed the pros and cons of each implementation, and suggested solutions to overcome them. Finally, using AWS IoT and Lambda functions, we showed how such devices can be used to control smart devices using WebHooks and IFTTT as brokers.

There is a lot of scope for improvement of this design to make it a marketable product. Due to manufacturing constraints, we were unable to create a compact and ergonomic housing for the components, but with the use of additive manufacturing and 3D modelling, it is possible to pack all the components into a single long cylindrical form with an ergonomic switch. For mass manufacturing, it might be better to not use the entire Arduino board and just use the required modules to save power and space. Further, the battery can be designed to be charged wirelessly or with a standard smartphone charger for increased convenience.

It is also of interest to see how Lumos can directly interface with the smart devices instead of going through a broker like IFTTT. Since most of the API's for smart devices are proprietary, this might also require some monetary investment or partnerships. Using a more powerful computer or RaspberryPi, it is also possible to increase the speed of detection and to remove the static background noise without overloading the processor, making the IR-based solution more feasible. Finally, it is also possible to train a small model to read IMU data and classify motions into recognized types, perhaps using cloud computation. This will come with its own power trade-offs, but can further open up the space for motion based inputs. The presented design for Lumos functions as a working proof-of-concept for development of similar gesture based input devices for the mass market, using IoT concepts and readily available hardware.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.

[2] G. Bradski. 2000. The OpenCV Library. *Dr. Dobb's Journal of Software Tools* (2000).

[3] Minwoo Kim, Jaechan Cho, Seongjoo Lee, and Yunho Jung. 2019. Imu sensor-based hand gesture recognition for human-machine interfaces. *Sensors* 19, 18 (2019), 3827.

[4] Yann LeCun and Corinna Cortes. 2010. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/. (2010). http://yann.lecun.com/exdb/mnist/

[5] Oleg Makaussov, Mikhail Krassavin, Maxim Zhabinets, and Siamac Fazli. 2020. A Low-Cost, IMU-Based Real-Time On Device Gesture Recognition Glove. In *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 3346–3351.

[6] Sanniti Pimpley. 2017. Using Harry Potter interactive wand with OpenCV to create magic: Learn OpenCV. https://www.learnopencv.com/using-harry-potter-interactive-wand-with-opencv-to-create-magic/

[7] Jasmeet Singh. 2019. Real Harry Potter Wand with Computer Vision. https://www.hackster.io/jasmeet-singh/real-harry-potter-wand-with-computer-vision-520e3b