Remote Free TV

A Gesture Based Natural User Interface for Roku Smart TVs

Brent George
bdgeorge@umich.edu
University of Michigan
Ann Arbor, Michigan, USA

Samuel Gonzalez
samgonza@umich.edu
University of Michigan
Ann Arbor, Michigan, USA

Tyson Lin
tysonlin@umich.edu
University of Michigan
Ann Arbor, Michigan, USA

ABSTRACT

Traditional television (TV) sets are predominately controlled by physical remotes which are easily misplaced and can confuse users with excess options. Yet, the use of more natural TV control interfaces like hand gestures are largely disregarded due to past failures resulting from poor recognition in complex environments. However, recent advancements in artificial intelligence have dramatically improved the accuracy and speed of gesture classifies, thus prompting a reinvestigation of the topic. In this paper, we present *Remote Free TV* - a computer vision and hand gesture interface for Roku streaming devices with a median miss-classification rate of 3.7%. From our user testing we find quantitative and qualitative evidence affirming that people enjoy using Remote Free TV when compared to a traditional remote control; demonstrating the viability and attractiveness of hand gesture TV control interfaces. Although just a prototype, Remote Free TV should inspire future work in deploying gesture and other natural user interfaces as an alternative or auxiliary means of controlling smart TVs.

KEYWORDS

Natural User Interfaces, Gestures, Machine Learning

ACM Reference Format:
Brent George, Samuel Gonzalez, and Tyson Lin. 2023. Remote Free TV: A Gesture Based Natural User Interface for Roku Smart TVs. In *Proceedings of FCHI'23.* ACM, New York, NY, USA, 5 pages.

This project is open source. A complete copy of the source code is available at https://github.com/samrg123/Remote-Free-TV

1 INTRODUCTION

TV remotes are not the most natural interface, and also come with a lot of problems. From an interface perspective, there are a wide variety of controls, especially on older remotes. Part of the appeal of streaming sticks is the reduction in the number of buttons; many users find that traditional remotes have too many controls and only learn and use a small subset in normal use [3] [8]. Additionally, controlling a TV traditionally has required a remote in your hand. If you have lost the remote (a common occurrence), or your hands are preoccupied or dirty, you cannot control your TV.

FCHI'23, 2023 Fake Conference on Human Factors in Computing Systems, April 17 2023, Ann Arbor, MI

Gestures have had their time in the spotlight as an interface over the past 20 years. They have fallen out of favor, but mainly because of the domains where they were applied. If you have a complex interface, gestures are difficult to use because there are only so many natural gestures you can use - swiping left or right, rotating, sticking up fingers, etc. We don't want to require users to learn a subset of sign language in order to send hundreds of commands.

Modern smart TVs only have a limited number of commands to start with, as referenced above. Combined with voice search, a list of commands such as "up", "down", "left", "right", "volumeUp", "volumeDown", "volumeMute", "power", "fast forward", "rewind", and "play/pause" would capture nearly all of the functionality required for a streaming service. Hence, even limited gesture actions may be enough to effectively control a TV.

This is our project - we want to control our TV using gestures. So for example, swiping your hand can relate to directional commands, moving your hand in a circle may control volume, and other ideas for commands may arise. We want commands to be both intuitive and accurate, so that users rarely feel the urge to reach for a remote. Therefore our approach to the project focuses on the Human Computer Interface (HCI) aspect of gestures involving intense user testing and feedback to find a truly appropriate set of gestures for a TV.

1.1 Previous Work

The use of hand gestures as a means of TV control has been attempted in the past by large TV manufactures such as Samsung[10]. However, these attempts were largely unsuccessful as they often frustrated users when they failed to detect gestures and required large arm movements to move a cursor around the screen making them exhausting to use for more than a few minutes[5]. Despite this, reviewers praised the novelty and natural intuitiveness of using hand gestures to replace remote controls even though limitations in gesture detection and inefficient UI design inevitably caused them to fail[1].

2 EXPERIMENTAL SETUP

2.1 Hardware

The hardware for our entire system centered around our choice of camera. A typical TV-watching environment usually involves seated individuals a few arms length distance away from a television. Our evaluation criteria for a camera included a large field of view, a variable range that matches our target environment, and depth camera capabilities to allow us to use supplementary depth data if it increased the accuracy relative to an RGB-only system. These criteria led us to use the Intel Realsense D435 Depth and RGB camera. With a nearly 90° field of view, 1-10ft range, an available

depth camera, and 90Hz refresh rate that limits frame latency, it met each of our criteria and ensured we wouldn't be limited by our imaging hardware.

In addition to the imaging hardware, we used a standard TV with HDMI input, a Roku Premiere 4K streaming stick device, and a laptop. In our setup the TV acts as a dumb display medium while the Roku was used to smart TV interface and features an internal library of streaming apps where the user can select and play digital content from. There were other streaming stick platforms available, such as Amazon Firestick, AppleTV, and Chromecast, but the Roku provided the simplest API to control externally via HTTP POST requests and thus was choosen for the project. Finally, a consumer-grade laptop running Linux was used process the video stream from camera, detect gestures, and forward the appropriate commands to the Roku. Dedicated hardware could alleviate I/O and compute bottlenecks for the gesture recognition process, but budget and time limitations of the project did not allow for this.

2.2 TV Interface

We opted to use the Roku as our smart TV interface because of its simple API. The Roku External Control Protocol (ECP) enables a Roku device to be controlled over a local area network by providing a number of external control services [9]. By finding the IP address of the Roku device on the LAN, we can send arbitrary commands through HTTP REST API calls to exposed endpoints on port 8060 of the device. We limited ourselves to a subset of functionalities; specifically, we only used commands that were available on a standard Roku remote. In total, we had directional commands (left, right, up, down), selection commands (ok, back, home) and playback commands (play/pause, fast-forward, rewind).

2.3 Gesture Framework

There are many different types of machine learning (ML) algorithms that we could have used to build our classifier, such as K-nearest-neighbors, linear regression, and random forest. Each algorithm has its pros and cons, but from our research convolutional neural networks (CNN's) tend to outperform other classifiers when detecting hand gestures with some models yielding accuracy above 90% as shown in figure 1. Therefore, in hopes increasing our chances at successfully detecting gestures, we decided to proceed with a CNN classifier for the project despite the additional overhead need to train it.

We simultaneously attempted two different approaches for gesture recognition during the pilot phase. The first approach was training our own computer vision machine learning model to detect gestures. This real-time model would feed its output into a simple python script that would send HTTP commands to a Roku device on a local area network. However, the time constraints of the project limited our confidence that we could produce a working product using this method as training a model from scratch is computationally expensive and isn't guaranteed to yield good results.

We then pivoted to the second approach, an off-the-shelf model. A online search for free options led us to MediaPipe[6], which is backed by Google and came recommended by Dr. Alanson Sample. Under the hood MediaPipe uses CNN's that offers solutions that work with our hardware constraints and allow for fine tuning. Additionally, MediaPipe integrates easily with Python and preliminary results showed we could deploy fast, accurate models with this framework.

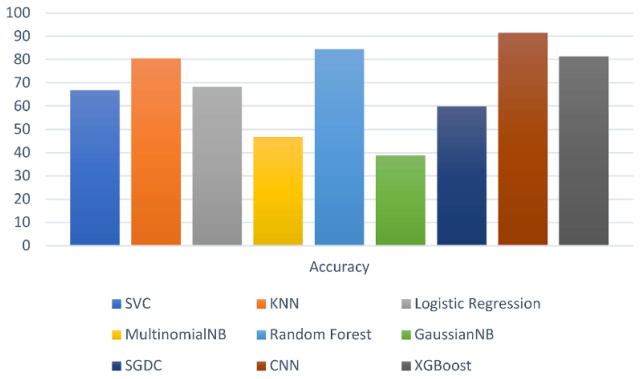

Figure 1: Accuracy of different machine learning algorithms when classifying hand gestures [2]. The MediaPipe model we use in this project relies on an underlying CNN for classification.

2.4 Gesture Selection

Gesture recognition traditionally incorporates two main components - hand/arm position and motion. Hand positions, like a thumbs up, are annotated by our MediaPipe model, and motion detection is done by a post-processing step as described in section 3.

To train our MediaPipe model to detect hand positions, a large dataset is needed to obtain high accuracy. We did not have the time or resources to curate a custom dataset that accounted for various depths and backgrounds, skin tones, hand sizes, and other confounding factors. This led us to explore open source hand position datasets. We only had 10 unique Roku commands, and some of them were natural candidates for multiplexing with motion (for example, one hand position for directional movements with motion differentiating between up/down/left/right). This led us to deprioritize the quantity of hand positions in a candidate dataset, and instead focus on the natural mapping from the gestures to Roku commands. In fact, gesture mapping would be a main focus of subsequent user studies. We made a final decision to use the HAnd Gesture Recognition Image Dataset (HaGRID) [4]. This provided a modern, large hand gesture database that had 18 positions; enough to choose natural mappings, but not so many that different gestures would be easily confounded with each other.

Our initial system used the following gesture-command mappings: an open palm plus a direction for the directional command, a peace sign for play/pause, a call sign for home, a reverse peace sign for back, an ok sign for select, and four fingers plus left/right direction for rewind and fast-forward. We later refined this mapping to be more intuitive in response to user feedback. The final gesture mapping can be seen in figure 2.

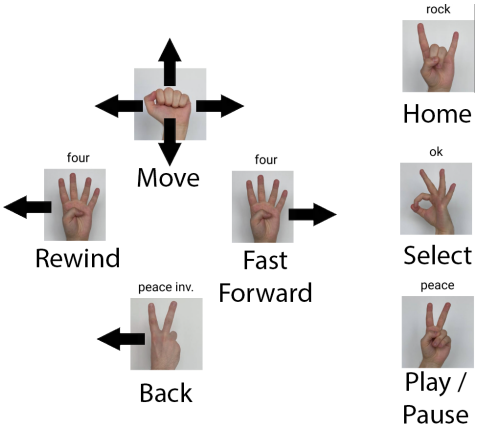

Figure 2: The final gesture-command mapping used for the project. Static gestures are shown on the right while motion gestures are shown on the left.

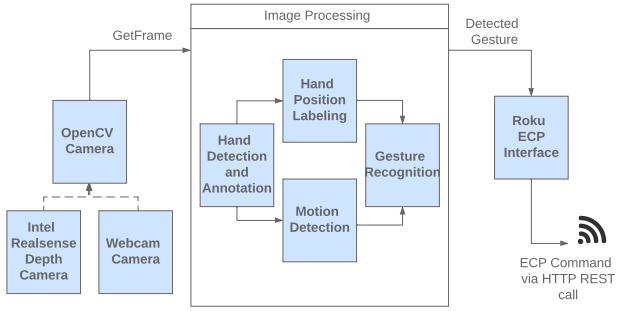

Figure 3: Software Architecture of Remote Free TV.

3 IMPLEMENTATION

3.1 Software Architecture

Figure 3 displays the flow of information in our system, from the raw camera input to commands sent to the Roku. An OpenCV video camera, with frame rate and resolution metadata, serves as the input to the system. We implemented two options for this camera, either the Intel Realsense D435, or a standard webcam. The image processing main loop will fetch frames from this camera and run analysis. This includes passing the frame through a MediaPipe model that identifies hand position coordinates and annotates where hands are present on the captured image. These hand annotations are then passed to a hand position labeling model trained on the HaGRID dataset, which identifies hand positions like "fist" or "rock" using the gestureTrainer.py script we wrote. The hand annotations are also passed to separate motion detection logic that keeps a rolling history buffer to detect motion of a hand across the frame. The hand position labels and motion data are combined to ascertain whether a full gesture is detected. If a valid gesture is output by the image processing, it is send to the Roku ECP interface, which sends an HTTP REST call to a pre-configured IP address reserved by the Roku streaming stick on the LAN.

3.2 Optimization

From our initial testing it became clear that although our system was able to consistently detect static gestures, it struggled with detecting motion gestures which frustrated users. Digging deeper we realized that most failed detentions stemmed from a combination of MediaPipe not detecting blurry moving hands (dropout) and the high latency associated with process video frames. To mitigate these problems and increase usability we implemented two main optimizations.

First, to increase robustness against dropout we implemented a finite state machine that transitions between four gesture states: IDLE, WAITING, ACTIVATED, and COOLDOWN. While IDLE, gestures wait to see their desired hand classification from MediaPipe and then transition to WAITING. While WAITING, gestures wait to see the desired motion before transitioning to ACTIVATED or transition back to IDLE if MediaPipe fails to detect the desired classification in a 0.25 second window which acts as a buffer that prevents dropout from resetting the gesture. Finally, ACTIVATED gestures send a their corresponding command to the Roku and transition to COOLDOWN state where they stay for 0.25 seconds before resetting to IDLE.

Second, to reduce latency we profiled our code and saw that MediaPipe was often only able to process 10-15 frames per second (FPS). This resulted in the builtin OpenCV video capture buffer filling up, significantly increasing the latency of the video stream we received from the camera. To get around this we refactored our code to continually pool frames from the camera on a separate thread and have the main thread poll frames from the camera thread. Doing so ensured that the OpenCV buffer never filled up and that the latest frame that we polled from the main thread was at most 16ms old as determined by our camera's frame rate.

4 EVALUATION

4.1 Training

Due to the large size of the HaGRID dataset (719 GB) we were unable to train MediaPipe using the raw images and instead used a subset of 500K images down sampled to a 512×512 resolution (14 GB) [7]. An 80/20 training/validation split was then used to train classifier with a batch size of 2, learning rate of 0.001, and 5% dropout for 1 epoch. Despite only training for 1 epoch due to limited time, the classifier converged on a 77.15% training accuracy with 0.5789 loss and 93.39% validation accuracy with 0.1858 loss.

4.2 Testing

To validate that viability of our interface we collected both qualitative and quantitative metrics from a series of user tests conducted by our friends and family members.

To start, users were provided with information on the study that outlined the goals of the project and what we hoped to learn by having them test the TV interface. They were then given a list of "objectives" to complete, designed to guide them user to use every gesture. Below is an example objective list given to a user:

(1) Navigate to the Amazon Prime app
(2) Select the third show in the second row

(3) Fast forward 15 minutes
(4) Play the video
(5) Go back to the start of the video
(6) Go back to the app screen
(7) Return to the home screen
(8) Browse and explore on your own

We then had users fill out a Google Form with the following questions to get a feeling of what each user thought about the TV interface and where we could improve:

(1) What do you like about the gesture control for the TV?
(2) How would you improve the gesture control for the TV?
(3) Are there any gestures you had difficulty with? If yes, why?
(4) What is your major?
(5) On average, how easy was it to use the TV, on a scale of 1-5? (1 = Horrible, 5 = Great)

We also collected the following quantitative data: the percent of gestures the system got correct, the percent of gestures the system missed, and the percent of gestures the system got wrong.

We continually performed users testing throughout development of the project, but the most recent iteration had the largest testing population. Here are the results for that iteration, with sample size $n = 7$.

	Median	Mean	Std. Deviation
Rating (1-5)	3	2.85	1.07
Correct	65.21%	59.39%	15.20%
Missed	34.78%	34.56%	12.45%
Incorrect	3.70%	6.03%	6.68%

From our qualitative feedback, users noticed that the system was a bit slow, and didn't register a lot of gestures. This lines up with our numeric data, with about 1 in every 3 gestures being missed. They also did not like how unnatural some of the gestures were, particularly the four fingers plus direction for rewind and fast forward, and the two fingers plus direction for back. These unintuitive gestures also were the gestures that the system had most difficulty classifying correctly.

Many users also noted a learning curve for the system as a whole. They had to figure out the optimal region to place their hands, and keep them in that region. They also had to figure out the optimal speed at which to perform motion gestures so that the system would register them. Once users figured this out, however, they reported that the system was quite fun to use.

5 LIMITATIONS

Our system's limitations can be broadly divided into two categories: ML model biases, and usability bias towards technical users.

Using the HaGRID dataset helped us recruit a much more diverse population than we could have recruited ourselves, but it still has limitations. We noticed that our model was highly affected by time of day and natural versus indoor lighting conditions. The classifier had a hard time recognizing gestures when there was sunlight in the background. Additionally, we do not have a large enough testing population to reliably measure bias based on skin tone, but a visual inspection suggests that the HaGRID dataset is skewed towards lighter skin tones. This could unfairly and adversely affect our system's accuracy for those with darker skin tones. Future work should be done to contribute to the HaGRID dataset to improve its diversity.

From user testing, we also found qualitative evidence that individuals with a technical professional background were faster in learning the mapping between gestures and commands. This may reflect a higher comfort level with technology and prior practice with digital interfaces. However, a true natural interface should be just as accessible and intuitive to non-technical users, and this suggests that further improvement to the gesture mapping can be explored.

6 FUTURE WORK

Many of the limitations of Remote Free TV can likely be resolved with additional research and refinement of the system. An example of this includes further user testing with custom made gestures, perhaps not currently found in the HaGRID dataset, which could isolate more natural gesture-command mappings and improve the overall usability of the system.

Similarly, streaming stick interfaces are limited to controlling their own software, and can't directly affect the TV settings. This includes the power and volume settings for a TV. These commands can often be communicated by identifying the IR frequency used by the TV and interpolating IR signals from the remote to directly control the TV, as power and volume cannot always be controlled through an HDMI port. Our current system is limited to commands sent through the Roku device over the HDMI port, but could be easily extended with additional hardware to send power and volume commands over IR.

Likewise, search interfaces are often controlled by on-screen keyboards navigated by directional controls. These are slow and being phased out by many services in favor of voice search. When combined with a microphone, a gesture could be added to the system to activate voice search. By adding power, volume, and voice search capabilities, future works could completely replicate all functionality of current streaming stick remotes.

Finally, TVs are not the only potential use of this system. The gesture recognition is independent of the media it is controlling, so our gesture classifier doesn't necessarily need to send commands to a TV. Out of curiosity, we decided to have the classifier send keyboard commands within the computer. Gestures were used to page up and page down, navigate tabs, control volume, and take screenshots. One application of this was that one of our team members was able to navigate a recipe with dirty hands.

7 CONCLUSION

Remote Free TV successfully demonstrates how static hand gestures can reliably be used instead of a remote control to performing simple tasks such as pausing and playing the TV while across the room. However, additional work is needed to decrease frame processing time and develop more intuitive gesture mappings before motion gestures are robust enough to be used in place of a remote. Nonetheless, we believe that as AI classifies and processing power improve over time, these limitations will largely be mitigated and the combination of intuitive voice and a natural universal gesture system will be capable of permanently replacing TV remotes on a large scale.

A ACKNOWLEDGMENTS:

Special thanks to Professor Alan Sample, Sumukh Marathe, and, Yatian Liu for the great learning experience!

REFERENCES

[1] John Archer. 2012. Samsung Smart TV voice and gesture control systems. https://www.trustedreviews.com/reviews/samsung-smart-tv-voice-and-gesture-control-systems

[2] Shashi Bhushan, Mohammed Alshehri, Ismail Keshta, Ashish Kumar Chakraverti, Jitendra Rajpurohit, and Ahed Abugabah. 2022. An Experimental Analysis of Various Machine Learning Algorithms for Hand Gesture Recognition. *Electronics* 11, 6 (2022). https://doi.org/10.3390/electronics11060968

[3] William Cooper. 2008. The Interactive Television User Experience so Far. In *Proceedings of the 1st International Conference on Designing Interactive User Experiences for TV and Video* (Silicon Valley, California, USA) *(UXTV '08)*. Association for Computing Machinery, New York, NY, USA, 133–142. https://doi.org/10.1145/1453805.1453832

[4] Alexander Kapitanov, Andrew Makhlyarchuk, and Karina Kvanchiani. 2022. HaGRID - HAnd Gesture Recognition Image Dataset. arXiv:2206.08219 [cs.CV]

[5] David Katzmaier. 2012. Samsung brings voice, gesture control to tvs. https://www.cnet.com/tech/tech-industry/samsung-brings-voice-gesture-control-to-tvs/

[6] Camillo Lugaresi, Jiuqiang Tang, Hadon Nash, Chris McClanahan, Esha Uboweja, Michael Hays, Fan Zhang, Chuo-Ling Chang, Ming Guang Yong, Juhyun Lee, Wan-Teh Chang, Wei Hua, Manfred Georg, and Matthias Grundmann. 2019. MediaPipe: A Framework for Building Perception Pipelines. arXiv:1906.08172 [cs.DC]

[7] Christian Mills. 2022. Hagrid classification 512P no_gesture. https://www.kaggle.com/datasets/innominate817/hagrid-classification-512p-no-gesture

[8] Office of Communications. 2006. Summary of research on the ease of use of domestic digital television equipment.

[9] Roku. 2023. External Control Protocol. https://developer.roku.com/docs/developer-program/dev-tools/external-control-api.md

[10] Agam Shah. 2012. Use voice, gestures to control TV. https://www.pcworld.com/article/469667/use_voice_gestures_to_control_tv.html