International Journal of Advance Innovations, Thoughts & Ideas

Video Open Access

Proxy: AI Vision controlled long distance Video Conferencing robot

Saharsh Sinha*

Department of Electronics and Communication Engineering, Harvard University, United States

Abstract

Proxy: Enhancing Inter-Continental Video Conferencing Proxy is a revolutionary robot designed to make inter-continental video conferencing more dynamic and interactive. What makes Proxy unique is its AI Vision technology, which allows users to control the robot hands-free through pose estimation. This means you can move the robot just by using gestures, making the experience more natural and immersive. Key Features of Proxy Hands-Free Control: Proxy uses AI Vision to understand your gestures and control the robot. This eliminates the need for traditional controllers, making it easier to navigate and interact. Real-Time Connectivity: Proxy can be controlled in real-time from anywhere in the world. For instance, someone in California can easily operate Proxy in India, ensuring seamless communication. Extra Dimension in Video Calls: Unlike traditional video calls, Proxy allows you to move around the remote location. This adds a new dimension to video conferencing, making interactions more engaging and lifelike. Whether youâ€™re attending a virtual family gathering or an international business meeting, Proxy makes you feel as if youâ€™re physically present. Building the Prototype Creating Proxy started with a prototype to tackle several challenges: Stability: Ensuring the robot remains balanced and steady. Agility and Control: Making sure the robot responds quickly and accurately to commands. Network Connectivity: Dealing with connectivity issues to ensure smooth operation. Battery Life: Optimizing power usage for longer use. The prototype focused on proving these key aspects before moving on to improving the robot's design. Resourceful Construction Building Proxy involved using common household items, showcasing that advanced robotics can be achieved with available resources: Chassis: Made from a wood composite panel and an end table. Internet Connection: Utilized an old laptop. Video Display: Used a standard monitor. Webcam: Meta's Portal TV served as the webcam, but any compatible webcam would work. Power Source: Powered by an Uninterruptible Power Supply (UPS). Overcoming Challenges A significant challenge was creating a control system that could handle network interruptions. The system needed to: Convert control signals in real-time. Detect signal breaks and pause the robot appropriately. Differentiate between brief and persistent network issues to ensure a smooth user experience. Improving Personal Connections Proxyâ€™s ability to move around during video calls helps maintain closer connections with loved ones across the world. This added mobility makes interactions more personal and engaging. In summary, Proxy is a cutting-edge solution that transforms how we experience inter-continental video conferencing. By adding mobility and hands-free control, Proxy brings a new level of connection and interaction, making remote communication feel more real and personal.

Proxy is a robot designed to make video conferencing more dynamic and interactive. What makes Proxy unique is its AI Vision technology, which allows users to control the robot hands-free through pose estimation. This means you can move the robot just by using gestures, making the experience more natural and immersive.

Detailed video: https://youtu.be/Fq1NKqIYgFo

Keywords: AI Vision Robot

Key features of proxy

Hands-free control: Proxy uses AI Vision to understand your gestures and control the robot. This eliminates the need for traditional controllers, making it easier to navigate and interact.

Real-time connectivity: Proxy can be controlled in real-time from anywhere in the world. For instance, someone in California can easily operate Proxy in India, ensuring seamless communication.

Extra dimension in video calls: Unlike traditional video calls, Proxy allows you to move around the remote location. This adds a new dimension to video conferencing, making interactions more engaging and lifelike such as when attending a virtual family gathering, Proxy makes you feel as if you're more present at the remote location.

Building the Prototype

Creating Proxy started with a prototype to tackle several challenges:

Stability: Ensuring the robot remains balanced and steady.

Agility and control: Making sure the robot responds quickly and accurately to commands.

Network connectivity: Dealing with internet connection intermittency to ensure smooth operation.

The prototype focused on proving these key aspects before moving on to improving the robot's design.

Resourceful construction

Building Proxy involved using common household items, showcasing that advanced robotics can be achieved with available resources

***Corresponding author:** Saharsh Samata, Department of Electronics and Communication Engineering, Harvard University, United States, E-mail: saharsh.sinha.career@gmail.com

Received: 01-June-2024, Manuscript No: ijaiti-24-139532; **Editor assigned:** 04-June-2024, PreQC No: ijaiti-24-139532 (PQ); **Reviewed:** 18- June-2024, QC No. ijaiti-24-139532; **Revised:** 24-June-2024, Manuscript No: ijaiti-24-139532 (R); **Published:** 29-June-2024, DOI: 10.4172/2277-1891.1000272

Citation: Saharsh S (2024) Proxy: AI Vision controlled long distance Video Conferencing robot. Int J Adv Innovat Thoughts Ideas, 12: 272.

Chassis: Made from a wood composite panel and an end table.

Internet connection: Utilized an old laptop.

Video display: Used a standard monitor.

Webcam: Meta's Portal TV served as the webcam, but any compatible webcam would work.

Power source: Powered by an Uninterruptible Power Supply (UPS).

Overcoming challenges

A significant challenge was creating a control system that could handle network interruptions. The system needed to:

Convert control signals in real-time.

Detect signal breaks and pause the robot appropriately.

Differentiate between brief and persistent network issues to ensure a smooth user experience.

Improving personal connections

Proxy's ability to move around during video calls helps maintain closer connections with loved ones across the world. This added mobility makes interactions more personal and engaging.

In summary, Proxy is a solution that transforms how we experience long distance video conferencing. By adding mobility and hands-free control, Proxy brings a new level of connection and interaction, making remote communication feel more real and personal.

Detailed video: https://youtu.be/Fq1NKqIYgFo

Part I: AI vision control mechanism

One the most unique aspects about proxy is it's hands free control mechanism that uses AI Vision (Tensorflow + MoveNet). In this article, we look at how it works.

Move Net is an ultra-fast and accurate TensorFlow model that detects 17 keypoints of a body. The model is offered on TF Hub with two variants, known as Lightning and Thunder. Lightning is intended for latency-critical applications, while Thunder is intended for applications that require high accuracy. Both models run faster than real time (30+ FPS) on most modern desktops, laptops, and phones, which proves crucial for live fitness, health, and wellness applications.

For this application, we have used Thunder**.**

Here we are leveraging this capability to detect various poses and then use that to determine and translate into control signals. Take this image for example [Figure-1].

The MoveNet model can detect in real time17 points. Let's take the case of detecting a simple gesture. The robot's activation "switch" is triggered when the left arm is bent at a certain angle at the elbow. This can be done using the coordinates of my left shoulder, left elbow and left wrist, identified as points 5, 7 and 9 respectively.

Then passing these coordinates into the following method returns the angle at which the left arm is bent [Figure-2].

Similarly, the rotation of Head can be determined using the ears and nose. Take this example [Figure-3].

Points 0, 3 and 4 are coordinates for the nose, left ear and right ear respectively. When the head is tilted left, as in Figure below, the nose

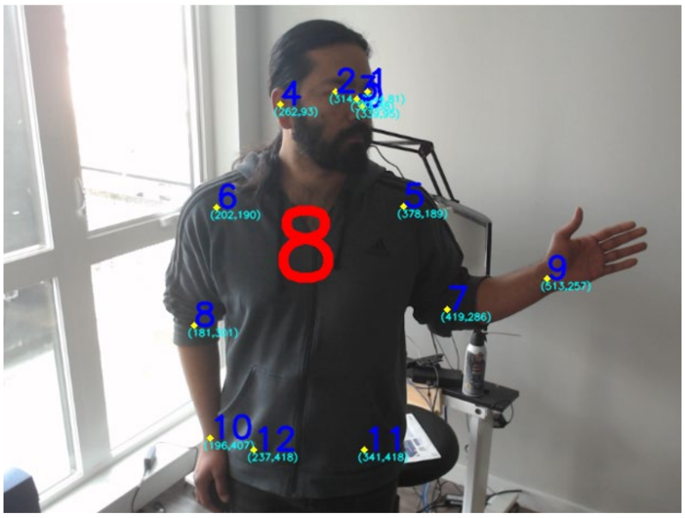
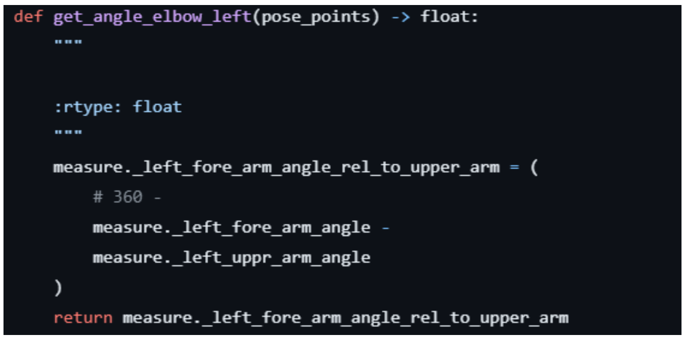


Figure 1: Hands-Free Control Mechanism Using AI Vision (TensorFlow + MoveNet).



```
def get_angle_elbow_left(pose_points) -> float:
    """

    :rtype: float
    """
    measure._left_fore_arm_angle_rel_to_upper_arm = (
        # 360 -
        measure._left_fore_arm_angle -
        measure._left_uppr_arm_angle
    )
    return measure._left_fore_arm_angle_rel_to_upper_arm
```

Figure 2: Then passing these coordinates into the following method return the angle at which the left arm is bent.

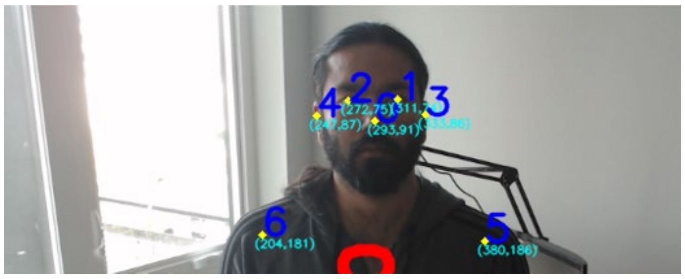


Figure 3: the rotation of Head can be determined using the ears and nose.

(point 0) is closer to the left ear (point 3) and further from the right ear (point 4) [Figure-4].

With the help of these three points, we can detect the rotation of the head using the method below.

https://github.com/SaharshSinha/proxy.mity/blob/main/Movenet_Webcam_App/motion_provider.py

An important aspect to this approach is that whatever parameters we decide for various directional controls, there should be a range that is neither too narrow nor too wide. If it was too narrow it would be difficult to maintain that pose. If it is too wide it might erroneously detect a pose as something else. With some experimentation I was able to identify the proper range for different poses.

The control signals are transmitted using the conventions on the numeric keypad. For example [Figure-5].

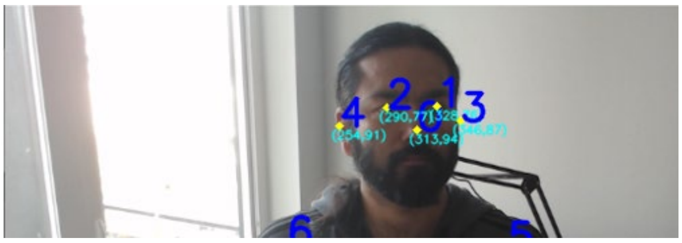


Figure 4: Nose and Ear Coordinates.

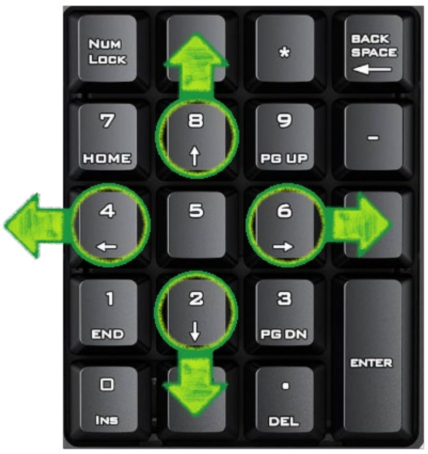


Figure 5: The control signals are transmitted using the conventions on the numeric keypad.

8: Move Forward

2: Move Back

4: Turn Left

6: Turn Right

7: Move Forward diagonally to the Left

9: Move Forward diagonally to the Right

1: Move Back diagonally to the Left

3: Move Back diagonally to the Right

These single digit numeric signals need low bandwidth to be transmitted from California to Bengaluru, and are easier to process at the receiving end, so while they are limited to 8 directions, they offer fast, real time control of the robot and prove sufficient to move the robot in various directions.

Part II: Transmission mechanism

To illustrate these communication mechanisms, we'll make use of the following graphic with some (grainy) gif animation [Figure-6]

TCP (Push)

TCP ensures reliable data transfer with robust error detection, recovery, and flow control, maintaining data integrity and order. However, its connection setup and management introduce overhead and latency, potentially reducing performance and requiring significant system resources. TCP's reliability features may be excessive for real-time applications where speed is prioritized over perfect data integrity.

https://github.com/SaharshSinha/proxy.mity/tree/main/Communicator/TCP.Sender

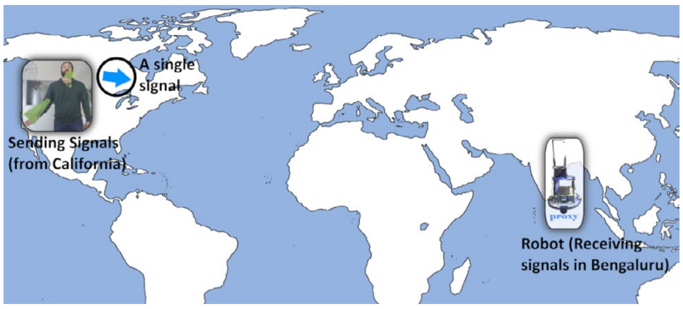

A single signal
Sending Signals (from California)
Robot (Receiving signals in Bengaluru)
proxy

Figure 6: To illustrate these communication mechanisms, we'll make use of the following graphic with some (grainy) gif animation.

https://github.com/SaharshSinha/proxy.mity/tree/main/Communicator/TCP.Receiver

HTTP Polling (Pull)

HTTP polling is simple to implement, allowing clients to control the frequency of update checks. However, it can be inefficient, generating many unnecessary requests and increasing network and server load. This method also introduces higher latency and can be resource-intensive, making it less suitable for real-time applications.

https://github.com/SaharshSinha/proxy.mity/tree/main/Communicator/Conveyer.Core31.Sender

https://github.com/SaharshSinha/proxy.mity/tree/main/Communicator/Conveyer.Core31.Receiver

Pub/Sub (Google Cloud Platform)

GCP Pub/Sub is a fully managed, real-time messaging service that allows applications to send and receive messages between independent systems reliably. It supports event-driven architectures, facilitating scalable and decoupled communication between services. Pub/Sub automatically handles message delivery, retries, and failures, ensuring reliable message transmission with minimal operational overhead.

https://github.com/SaharshSinha/proxy.mity/tree/main/Communicator/GCP.Sender

https://github.com/SaharshSinha/proxy.mity/tree/main/Communicator/GCP.Receiver

SignalR

SignalR enables real-time, bi-directional communication across various platforms with automatic reconnection, making it ideal for dynamic applications like chat apps and live notifications. However, scaling SignalR can be complex and resource-intensive, requiring additional infrastructure and configuration. Its reliance on WebSockets for optimal performance can lead to degraded efficiency in environments where WebSockets are not supported.

https://github.com/SaharshSinha/proxy.mity/tree/main/Communicator/SignalR.Hub

https://github.com/SaharshSinha/proxy.mity/tree/main/Communicator/SignalR.Sender

https://github.com/SaharshSinha/proxy.mity/tree/main/Communicator/SignalR.Client

SignalR can use either push or pull communication model depending on environment.

These 4 mechanisms have different levels of latency, however all of them work well with the "Approach 2" method of moving the robot as it is more resilient to network latency up-to 1s.

The code for all of these mechanisms are details here:

https://github.com/SaharshSinha/proxy.mity/tree/main/Communicator

Part III: Robot control mechanism

Interpreting the control signal and moving the robot

There were two approaches I tried for moving the robot based on a stream of control signals. The main challenge was to mitigate network speeds with respect to individual packets, each if which would contain one signal.

One approach better than the other, so let's look at the two approaches. A brief look at the structure of the animations below would help in understanding the nuances [Figure-7].

Approach 1 (Easy to implement, but causes janky motion)

The first approach was to move the robot for a short distance every time a control signal was received in Bengaluru. So if the Robot successively received "8", which is "Move Forward" command 20 times, it would move forward bit-by-bit, 20 times. This led to a very janky motion because the signal over the network would not arrive evenly spaced. I think this can be mitigated with some advance Arduino code, but needs to be explored further.

Approach 2 (Needed a bit more handling, but smoother)

To overcome the unpredictability of network speeds and bandwidth, the other approach was to keep the robot moving until a different signal was received or no signal was received.

The second approach worked better because in this case we would start moving when a signal was received, and keep moving in that direction until another signal was received. If no signal was received for 1000 milliseconds, the movement would "expire" and an abort signal would be issued which would would stop the robot. This approach was far smoother and led to a more fluid user experience.

This was accomplished at the C# layer using the following code [Figure-8].

https://github.com/SaharshSinha/proxy.mity/blob/main/Communicator/Conveyer.Core31.Receiver/COMCommunicator.cs

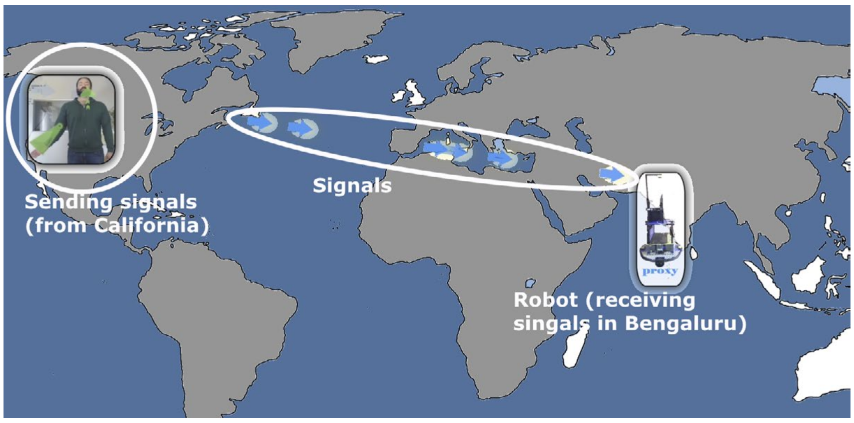
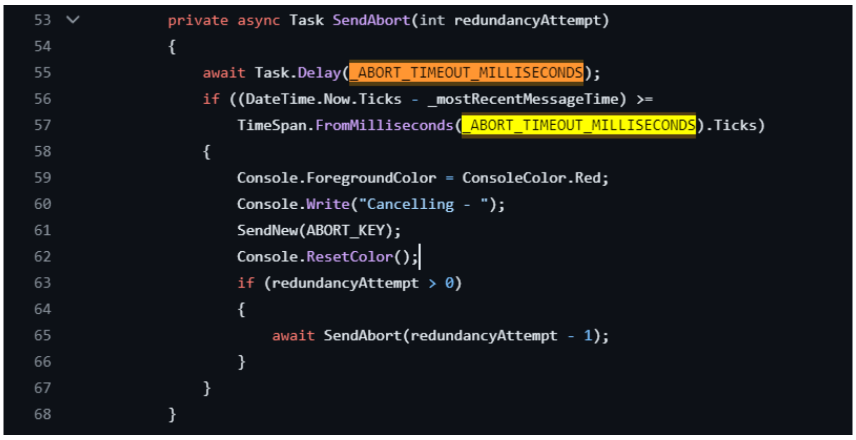


Figure 7: Interpreting the Control Signal and moving the Robot.



```
        private async Task SendAbort(int redundancyAttempt)
        {
            await Task.Delay(_ABORT_TIMEOUT_MILLISECONDS);
            if ((DateTime.Now.Ticks - _mostRecentMessageTime) >=
                TimeSpan.FromMilliseconds(_ABORT_TIMEOUT_MILLISECONDS).Ticks)
            {
                Console.ForegroundColor = ConsoleColor.Red;
                Console.Write("Cancelling - ");
                SendNew(ABORT_KEY);
                Console.ResetColor();
                if (redundancyAttempt > 0)
                {
                    await SendAbort(redundancyAttempt - 1);
                }
            }
        }
```

Figure 8: This was accomplished at the C# layer using the following code.