# Using Apple Vision Pro to Train/Control Robots

Younghyo Park

## 1 Introduction

Apple Vision Pro (AVP), a virtual/augmented reality (VR/AR) device recently released by Apple, tracks various movements of the user wearing the device. Wrist and finger movements, for instance, are constantly tracked and used as its primary interface for user interaction; sensors included in the device are optimized to provide an accurate tracking of human movements. Such tracking capability makes the device particularly appealing for many robotic applications: AVP can be used to (1) record the navigation and manipulation behaviors of humans in real-world environment, and can also be used to (2) teleoperate a robot with intuitive human motions. The device's virtual and augmented reality capabilities also opens up new avenues for immersive experiences during robotic teleoperation.

This repository thus aims to provide diverse array of tools for using AVP in robotics applications, starting with an easy-to-use library with minimal dependencies that can stream the tracking data from the Vision Pro to any client device connected to the same network.

## 2 Tracking Streamer: VisionOS App

Tracking Streamer is an VisionOS app that you can install from the App Store. The code is also open-sourced in the repository. The app serves two primary purposes: (a) tracking the human movements, and (b) streaming the tracking data over network.

**Retrieving Tracking Data** The app uses Apple's ARKit as its core library to track every movements of the user wearing AVP. There are three main movements that are being tracked by the ARKit: (a) head, (b) wrist, and (c) fingers. Note that AVP also tracks user's eye

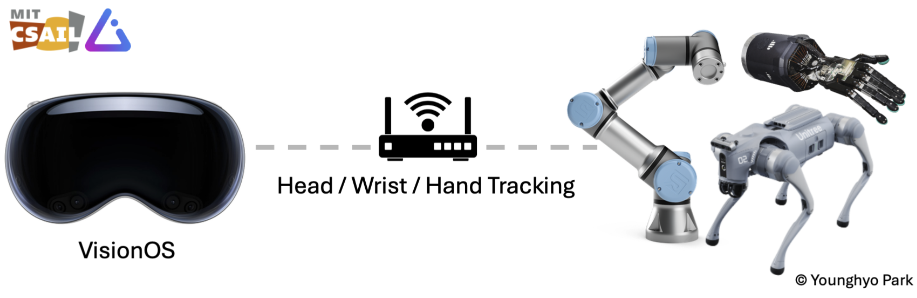


Figure 1: The app streams human movements to robots connected to the same network.

movements, but Apple restricts developer's direct access to the tracking data due to privacy concerns.

To get the device's location (head frame), the app constantly calls `queryDeviceAnchor` which returns the SE(3) location of the device with respect to a fixed global frame $\mathbf{T}_g$. Global frame is initialized when the app is first launched, and is attached to a ground where you're in. You can also reset the global frame to current position by long-pressing the digital crown. The head frame (device location) is then tracked using Apple's own localization algorithm, which can accurately locate the user even in large scale environments (Video). The app also uses `HandTrackingProvider` to track the user's wrist and fingers during the session. It tracks the position and orientation of user's two wrists (left and right) with respect to the ground frame, i.e., $\mathbf{T}_{\text{gw}_\text{L}}, \mathbf{T}_{\text{gw}_\text{R}}$, and tracks the pose of 25 finger joints in each hand with respect to its wrist frame, i.e., $\mathbf{T}^i_{\text{wf}}$.

**Communication with gRPC** The app then uses gRPC as its network communication protocol to stream the data to any clients existing in the same network. Use of gRPC instead of other robotics-oriented communication protocol (e.g. ROS2) allows the data to be subscribed from wider range of devices including Linux, Mac, Windows machines.

## 3 Python API

Subscribing to the data is easy: users can simply install the Python package:

```
pip install avp_stream
```

Below code snippet demonstrates how developers can access the data it's being streamed.

```python
from avp_stream import VisionProStreamer
avp_ip = "10.31.181.201"   # example IP
s = VisionProStreamer(ip = avp_ip, record = True)

while True:
    r = s.latest  # gets the latest tracking data
    print(r)
```

### 3.1 Available Data

The tracking data is represented as a dictionary containing the following key/values. Most of them are raw data streamed from the device, but some are post-processed by the Python library.

- `right/left_wrist`: SE(3) pose of right/left wrist, measured from ground frame. `np.array` with shape (1,4,4)
- `right/left_fingers`: SE(3) pose of 25 finger joints in right/left hand, measured from the wrist frame. `np.array` with shape (25,4,4).
- `head`: SE(3) pose of the head, measured from the ground frame. `np.array` with shape (1,4,4).
- `right/left_pinch_distance`: distance between thumb and index finger. `float`.

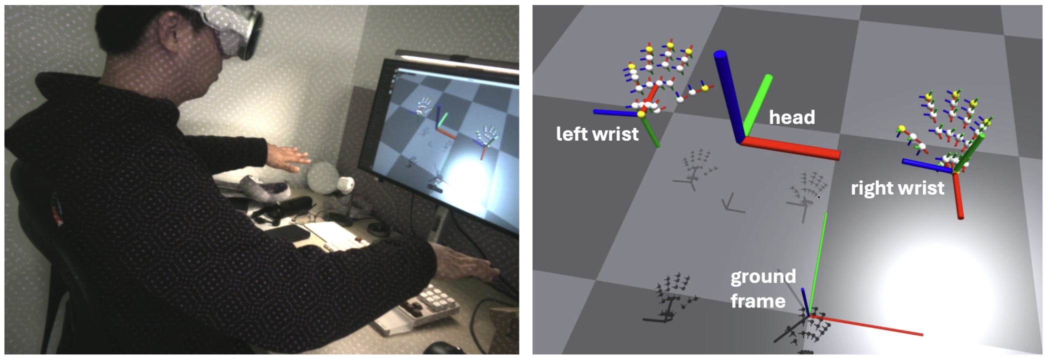


Figure 2: How the axes are defined for head, wrist, and fingers.

- `right/left_wrist_roll`: roll rotation of your wrist. Measures how you rotate your wrist around your arm axis. `float`.

### 3.2 Axes Conventions

Figure 2 shows how the axes are defined for each components. Note that Apple originally uses `Y_AXIS_UP` convention for their tracking. Considering that most robotics applications use `Z_AXIS_UP` instead, our python library automatically converts Apple's representation following `Z_AXIS_UP` convention.

### 3.3 Recording Data

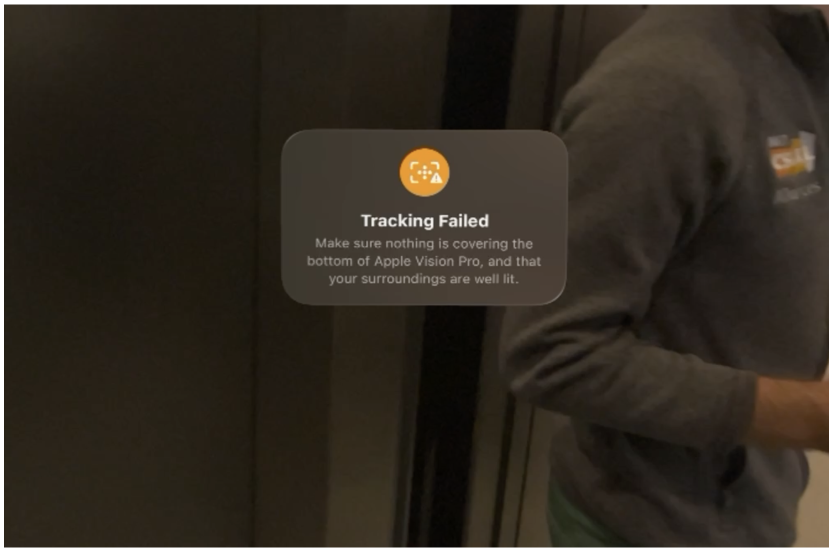
If you pass in `record = True` when initializing the `VisionProStreamer` class, it will simply store every streamed data into a list. To access and save the recorded data, simply call:

```
# save the recording when the app finishes
torch.save(s.recording, file_path)
```

## 4 Things to be careful about

During our experiments, we realized couple of scenarios where the app didn't behave as we expected. Most of these failure cases actually stemmed from our misunderstanding of how Apple's ARKit is designed to work.

- **Don't use it inside an elevator (or any moving vehicles!)** — Apple's ARKit will fail to localize the device while you're in a contained, but moving, environment. That includes airplanes, cars, and even elevators! That's why AVP has a separate "airplane" mode which you can turn it on from control center (perhaps a mode that exclusively uses visual information for localization). Whenever you're in those environments without the mode being turned on, you'll see this error message (Figure 3) and every tracking will stop.
- **If you're moving downhill, your z will also be decreasing.** — Once you opened the app, the ground will not move. It'll stay there unless you actively reintialize your ground frame by long-pressing the digital crown. If you're moving slightly downhill (or


Tracking Failed

Make sure nothing is covering the bottom of Apple Vision Pro, and that your surroundings are well lit.

Figure 3: AVP fails to track its device location whenever it's in a moving vehicle (e.g. elevators, airplanes, cars, trains).

uphill), for instance, your **z** will actually decrease (and increase) without you noticing. This might actually affect the robot's behavior, depending on how you designed to use the tracking.

- **If you put your hands completely down, AVP cannot see your hands.** — When you're walking normally with your hands around your waist/hip, AVP struggles to detect your hands. In those scenarios, the hand tracking stream might be a bit noisy. The library doesn't do any smoothing or filtering on the streamed data: so you might have to implement your own post-processing function to filter out those noisy estimations.

## 5 Conclusion

In the near future, people might be wearing devices like AVP all the time, like how we wear glasses. Imagine how much data we can collect from those! It's truly a promising source of data which robots can learn how humans interact in the real world. I hope this repository is helpful for anyone trying to use AVP for any robotics applications.

**Future Plans** This repository will be updated frequently, adding more features for robotics applications using Apple's ARKit and RealityKit. Investigating the possibility of establishing a bilateral connection between simulation/real-world and AVP is our immediate next step.

**Acknowledgement** Thanks to Gabe Margolis who gave me awesome feature requests that significantly enhanced the usability of the app/library.