Pose Auto-fix System for Full-Body Inverse Kinematic Tracking using HTC Vive Trackers

Seunghyun Lee
Information & Communication Engineering
Daegu Gyeongbuk Institute of Science & Technology
Daegu, South Korea
coder@dgist.ac.kr

Junsang Park
Information & Communication Engineering
Daegu Gyeongbuk Institute of Science & Technology
Daegu, South Korea
jshparksh@dgist.ac.kr

***Abstract*—We propose a system that automatically fixes tracking points' poses when it is not working well. Our system targets a full-body inverse kinematic tracking system using 6 tracking points, which contains a VIVE headset, two VIVE controllers, 3 VIVE trackers. Still, it is possible to apply this solution to any kind of system that has 6 tracking points that provide 3-dimensional position and rotation data. Our solution calculates the missing tracking device's position and rotation based on other tracking devices using the naive method or recurrent neural network. Code is available at https://github.com/r3coder/ViveTrackerAutofix**

***Index Terms*—Virtual Reality, Full Body Inverse Kinematic Tracking, full-body IK Tracking, HTC Vive**

I. Introduction

The virtual reality system is one of the demanding industry in recent years. After the commercial success of the Oculus Rift [1], people who demand VR for general purposes like games, applications have also access to the VR system. Numerous companies target several milestones to make the normal user is also accessible to VR. Price, Scan rate, or the resolution of the HMD, Tracking availability, comfort while wearing VR devices, and other technologies are the main goal for companies to develop VR Systems targeted on general users.

Above main goals, tracking availability is one of the most major interest among VR users. Especially, applying inverse kinematic to humanoid models is one most thing to evaluate tracking performance, since it can mimic human movement into a virtual world and provide an immersive experience to users.

A VR system needs several tracking points to execute an inverse kinematic tracking. Most of the tracking system uses 3 tracking points(Head, Left hand, Right hand) to do it, and it can track a real user's upper body to a virtual world. Because human's activity is quite focused on the hand movement, tracking system with 3 tracking points is quite enough for general VR users who target for games or applications.

Still, users who target more specific movement including the lower half of the body, there should be more tracking points. There are more tracking methods like 4-Points and 6-Points [2], and 8-Points [3], still 6-Points are most popular. A tracking system that has more than 6-Points are usually called as full-body tracking since tracking system using more than 6 points can track leg movement. Various tracking devices can execute these tracking systems. Kat loco [4] contains 3 sensors that can support a max of 6-Point tracking. It has an advantage at price, yet tracking may fail in several situations due to its tracking method. NOLO VR [5] can be a solution for full-body tracking since it supports 6-Point tracking using 3 additional tracking device, but it's tracking ability is quite disappointing. Microsoft Kinnect (or Azure Kinect DK [6]) is another method to do tracking since it captures the whole body and applies to the tracking system. Still, the tracking area is limited to the area that the camera can see, which may limit the user's movement.

Fig. 1. HTC Vive base station 2.0

Because of the stability of the tracking performance, HTC Vive Tracker (2018) [7] is the best solution for full-body tracking, especially if a user is using the HTC Vive headset and controllers. HTC Vive VR System uses Base stations (Fig. 1) mounted in the room and method that base stations working makes tracking system stable. There is a more specific explanation about base station and tracker at II-F. Also, if a user is using HTC Vive HMD or Value Index HMD, the user doesn't have to install additional devices to make trackers working.

Even if the tracking system using Vive trackers is quite stabilized, it can fail in several situations. Explained on II-F, a Vive Tracker uses infrared rays, so if a tracker is hidden under a certain object, the tracker will malfunction since infrared can't penetrate solid objects. Also, if there is an object reflects light(e.g. window, mirror), it will disturb the tracking

device to work properly. Those situations may cause tracker malfunction, and it leads to the displacement of the model using inverse kinematic.

There were also some efforts that tried to fix errors while tracking. The Method that Mendelsohn et al. [8] to fix the tracker's position using Line of sight method. But this works on Projection-Based VR Systems, so it may not work properly on the system we are targeted. Also, the method that Peer et al. [9], Niehorster et al. [10] corrects the tracking alignment of the minor difference in position on trackers. We focus on the situation that a certain tracker is completely malfunctioning, so this method can't be applied to our situation, too.

We focus on the situation when a certain tracker is malfunctioning, especially the tracker placed on the pelvis. That's because the tracker placed on pelvis has the highest possibility to malfunction. Moreover, if the pelvis tracker fails to get tracking data, the avatar's displacement is strongly visible since the pelvis tracker is connected to a major vertex group that decides the main position of the avatar.

Also, We focus on developing a system that can fix the tracker position within less than 12ms or 8ms. This is based on the scan rate of current VR systems. VR system's scan rate is different by machine varies from 75Hz to 120Hz. Most of the current VR systems using a 90Hz scan rate, some are using 120Hz. Since the tracker position is refreshed every frame, calculation done by the algorithm must meet the deadline of 12ms, which is for the VR system that has the 90Hz scan rate, and 8ms, which is for the VR system for the 120Hz scan rate.

II. Background

A. Abbreviations and Acronyms

HMD stands for head-mount display.
VR stands for virtual reality.
IK stands for inverse kinematic, which is explained on II-E

B. VR System and VRChat

A VR system is a system using VR headset and controllers, and other accessories. A VR System requires a VR headset. There are numerous VR headsets by various companies. Based on a Steam hardware & software survey at May 2020 on VR headset [11], 26.1% of users use HTC Vive, 22.0% of user uses Oculus Rift S, 16.6% of user uses Oculus Rift, 11.9% user uses Valve Index, 8.5% user uses Windows Mixed Reality, and 14.87% for other VR headsets like Pimax VR, Oculus Quest, Oculus Go, Vive Focus, etc.

The majority of the full-body IK tracking uses Vive trackers, and it supports the HTC Vive headset and Valve index headset. Those systems share a tracking environment based on base stations which are explained at II-F. Also, there are some Oculus Rift users who use Vive trackers, even if they have to buy separate base stations to make Vive trackers working. Still, not all users have a system for full-body tracking, because there are only a few games that support full-body IK tracking using Vive trackers.

VRChat, however, is the most famous application using VR. Since it's original purpose is to gather around and *chat* while wearing a VR headset, it is possible to create its content by users, and share it with other users. One of the major content created by the user is the *avatar*, which is a way to identify itself on VRChat. Most of the avatars on VRChat are human-like, also tracking device is mapped to avatar's bones, which is explained on II-D. This is to mimic human movement, and it enhances the play experience of the users to make users feel they are existing on the VR.

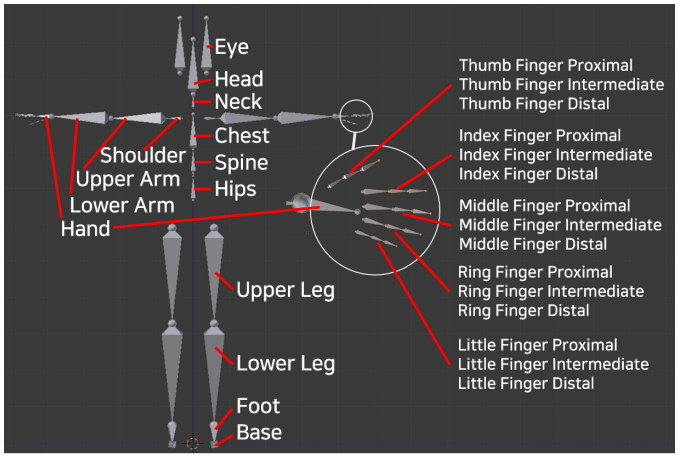

Fig. 2. The bone structure of the humanoid rigged avatar model used on VRChat to track full-body

C. Full Body Tracking

Originally, if a player uses an HMD and controller to play VRChat, a player can only move their arms and head. Other body parts like hips, legs are calculated automatically, it doesn't mimic player's movement. It is handy for most situations, but still, there's some disorientation on some actions. For more immersive play experience, players sometimes prepare systems that can mimic the rest part of the body. There are several solutions like KAT loco [4], NOLO [5], and Microsoft Kinect [6]. But the one most popular solution is the system using HTC Vive trackers. A more specific explanation about the Vive tracker is in Section II-F. Upon using an additional 3 Vive trackers and place them on the pelvis and both feet, it is possible to mimic the movement of the head, arm, leg, and waist to a humanoid avatar.

D. Humanoid Rigged Avatar Model

Full-body tracking is usually applied to the humanoid rigged avatar model. To mimic the movement of the human, the model has few essential bones attached, controlling those bones makes easier to mimic human movement.

An avatar model supports full-body tracking in VRChat contains bones like; Hips, Spine, Chest, Neck, (Left/Right) Upper leg, (Left/Right) Lower leg, (Left/Right) Foot, (Left/Right) Base, (Left/Right) Shoulder, (Left/Right) Upper Arm, (Left/Right) Lower Arm, (Left/Right) Hand. Also, there are 15 finger bones to mimic hand gestures and 2 eye bones to eye tracking. Each bone's position and shape are marked in

Fig. 3. VIVE Tracker (2018)

Fig 2, still, each bone's size and shape may vary depends on the avatar shape.

E. Inverse Kinematic

To mimic a player's movement into a humanoid rigged avatar model, inverse kinematic is applied. It is the mathematical process of calculating the variable joint parameters needed to place the end of a kinematic chain. This is widely used in robotics engineering to control robots, but this is also applied to animate a character's skeleton. Full-body inverse kinematic in VR using 6 tracking points are positioned on eye, pelvis, each hand, and each foot.

By using IK, rotation or position change of the tracking points changes the transform of the bone, and connected bones to make movement natural and reasonable. For example, if a player raises a leg, then the foot bone's position rises, and to match the position of the foot bone, lower leg bone's rotate, and vise versa for the upper leg. Also, hip bone needs to be adjusted a bit, because a real human's body's hip also moves when he tries to raise its own feet.

F. HTC Vive Tracker

HTC Vive Tracker [13] is a tracking device developed by HTC to bring real-world's objects to VR. It's shaped like Fig. 3, it can be attached to various devices like camera, controller, gimbals, etc. As mentioned before, if a player has 3 Vive trackers, it is possible to mimic most of the human movement to the character.

To make Vive trackers work properly, at least two *base station*s are needed. Base stations are also essential for Vive headset and Vive controllers. Base stations emit an infrared signal to the tracking devices. In Fig. 3, there are some small holes all over the device. From each hole, there are infrared sensors, and signals are gathered. Then, it is processed preliminary inside, and transmitted to the desktop that controls the VR system. Vive controllers and Vive headset also have sensors to gather infrared signals emitted from base stations.

Tracking data from a single Vive tracker provides a 4x3 float array. The additional calculation is needed if we need to use them as location and rotation that easy to handle. To retrieve position data, it is possible by accessing the array data directly. Pos_x, Pos_y, Pos_z indicates x,y,z position of the tracker based on two base stations.

$$\begin{bmatrix} i_{00} & i_{10} & i_{20} \\ i_{01} & i_{11} & i_{21} \\ i_{02} & i_{12} & i_{22} \\ Pos_x & Pos_y & Pos_z \end{bmatrix}$$

But for rotation data, more complex calculation is needed.

$$Rot_w = \frac{1}{2}\sqrt{max(0, 1 + i_{00} + i_{11} + i_{22})}$$

$$Rot_x = \begin{cases} \frac{1}{2}\sqrt{max(0, 1 + i_{00} - i_{11} - i_{22})}, & \text{if } i_{21} - i_{12} \geq 0 \\ -\frac{1}{2}\sqrt{max(0, 1 + i_{00} - i_{11} - i_{22})}, & \text{otherwise} \end{cases}$$

$$Rot_y = \begin{cases} \frac{1}{2}\sqrt{max(0, 1 - i_{00} + i_{11} - i_{22})}, & \text{if } i_{02} - i_{20} \geq 0 \\ -\frac{1}{2}\sqrt{max(0, 1 - i_{00} + i_{11} - i_{22})}, & \text{otherwise} \end{cases}$$

$$Rot_z = \begin{cases} \frac{1}{2}\sqrt{max(0, 1 - i_{00} - i_{11} + i_{22})}, & \text{if } i_{10} - i_{01} \geq 0 \\ -\frac{1}{2}\sqrt{max(0, 1 - i_{00} - i_{11} + i_{22})}, & \text{otherwise} \end{cases}$$

Rotation based on this calculation is not a Euler rotation, this is Quaternion rotation values. Quaternion rotation values have four dimension, Rot_x, Rot_y, Rot_z, Rot_w. In brief, Rot_x,Rot_y,Rot_z indicates vector of the rotation and Rot_w indicates size of the vector. We convert this values to Euler rotation if we needed.

III. Motivation

In this section, we explain why this system is on-demand for full-body IK users who using VR applications based on our observations we had. Our simulation environment is desktop with an AMD Ryzen 1600 with NVIDIA GeForce RTX 2080 Ti. Our VR environment is HTC Vive HMD with 2 HTC Vive controllers, 3 HTC Vive trackers.

Upon using a full-body IK tracking system, one of the most disturbing things is the displacement of the tracker. Fig. 4. shows what happens if pelvis tracker malfunctions. This happens because HTC Vive tracker uses infrared rays which can't penetrate objects. Since it is needed to place pelvis tracker around the pelvis to do full-body IK tracking, there's a high possibility to hide tracker by clothes, own body.

There is three major placement of the pelvis tracker, and it has its advantages and disadvantages. The first method is placing pelvis tracker just below the navel. This is also the majority of the full-body IK tracking users' method to place pelvis tracker. It is widely used because with this method, a gesture is quite unlimited, so it's easy to express movement without limit. But, tracking malfunctions when the user tries to fold their body forward because the tracker is hidden by the user's own body. The second method is placing pelvis tracker around the coccyx. With this method, folding body forward may not be a problem, but this time, the user can't lie down to the floor. The third method is placing pelvis tracker to side. With this method, there's no limitation at movement, but overall IK tracking accuracy drops since tracker is biased on one side.

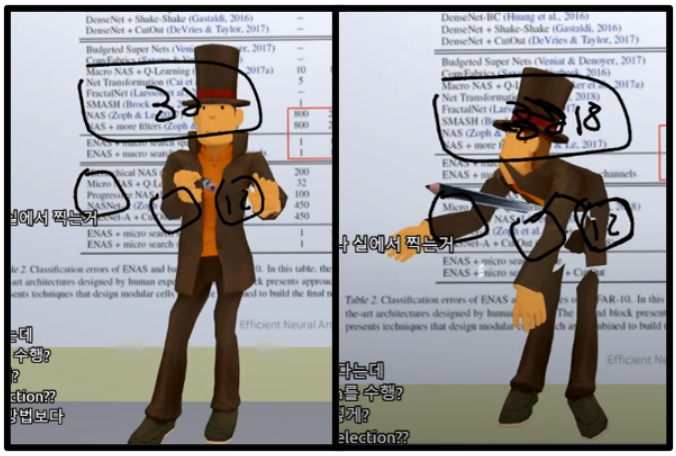

Fig. 4. Effect of malfunction tracking device, Avatar displacement. Left: every tracking device is fully working. Right: Pelvis tracker is malfunctioning, so waist is hanging up [11]

TABLE I
ERROR RATE WHILE USING FULL BODY INVERSE KINEMATIC TRACKING SYSTEM USING 6 TRACKING POINTS, PELVIS TRACKER PLACED ON NAVEL.

Action	Pelvis Tracker	Left Leg Tracker	Right Leg Tracker
Idle(Standing)	3.2%	0.3%	0.4%
Moving	3.8%	0.7%	0.7%
Dancing	4.5%	1.0%	0.9%
Idle(Sitting)	7.0%	0.4%	0.5%

Table I shows *error rate* of the trackers on various situations. We defined error rate as

$$[\text{Error rate}] = \frac{[\text{Number of frames malfunctioning}]}{[\text{Total number of frames}]} \quad (1)$$

Since HMD and each controller have a very low error rate (0.01%) in every situation, we didn't include them on the table.

Here's a brief explanation of each action. *Idle(Standing)* is just standing, with some simple hand gestures. This action supposes a situation when a user has a conversation with other people while using hand gestures. *Moving* is roaming freely inside the room. This action supposes the situation with some random movements while the user plays on VR space.*Dancing* is the action that the player moves quickly. Usually, quicker movement easily leads to more errors based on experience, so we danced on while using full-body IK tracking. Last, *Idle(Sitting)* contains various sitting gestures like sitting on a chair, floor.

Every action was measured about 10 minutes(about 50000 frames), placing pelvis tracker around the navel. Also, we tried to remove every other object that reflects light(e.g. window, mirror) to retrieve the best possible result.

As shown on the table, the error rate of pelvis tracker on *Idle(Standing), Moving, Dancing* actions are around 3-4%, which seems quite low because it seems for average 3-4 frames are dropped per second. But, tracking error usually continues more than 60 frames(about 0.7 seconds), and the displacement of humanoid avatar causes unpleasant experiences to users. This unpleasant experience is more severe in *Dancing* action, since a small displacement of the trackers while vivid movement like dancing is more disturbing.

Apart from other actions, *Idle(Sitting)* action's pelvis error rate is quite high, due to the pose of users have a high possibility to hide pelvis tracker by own body. Since the pelvis tracker's position is quite stabilized while sitting, error from this action is not a major issue while full-body IK tracking.

IV. METHOD

We created a tracker position adjustment system based on VRPlayspaceMover [14]. VRPlayspaceMover is a utility that can modify play space. It is widely used on users in VRChat who uses full-body tracking to manipulate their position. There are two major reasons that we selected VRPlayspaveMover to work with. First, since it is using an application program instance for VR, it is more convenient to create and check if we use this program. Second, adjusting tracker's position is also one way to moving play space, so if we combine features of VRPlayspaceMover with our solution, it's result will give a better experience to players.

Our major task is to fix the pelvis tracker's position. It's because unlike other tracking devices, the error rate of the pelvis tracker is significantly higher than other trackers. Also, since the pelvis tracker is connected to the hips bone which decides the avatar's main structure, the malfunction of the pelvis tracker leads to major displacement of the avatar, unlike other tracking devices. We tried three methods to fix the position of the pelvis tracker.

A. Naive Method

For the first task, we adjusted the pelvis tracker's position and rotation by calculating manually. While the player is in standing motion including moving a simple action, we observed that the human body doesn't allow the pelvis tracker to change its position dramatically.

At here, we define the *center position* as the mean value of position values from 6 tracking points. Upon comparing the position value of the center position and the position value of the pelvis tracker, it turns out that it may possible to approximate the position of the pelvis tracker using center position and adding some offsets. Based on data gathered on using tracking, we picked the ideal ratio of the position of the pelvis tracker based on the other tracker's position.

For rotation, we calculated the normal vector of the plane with 3 points. Three points are: HMD, left foot, right foot. Since pelvis' direction is quite determined, calculating normal vector like this method may derive similar rotation by original form. We converted Euler rotation of the normal vector to Quaternion rotation to apply to the pelvis tracker. Result is on section V.

B. DNN

The Naive method is easily implementable and also easily deploy-able still, there are some limitations to this method.

First, this method can't automatically configure if the player's tracker setup changes. As mentioned on III, the tracker could be placed on various positions on the body. Not only the pelvis tracker, minor position, and set up for the headset, controller, and foot trackers are also varied by each users' setup. It is not only inefficient but also impossible to write a program code for every user.

Due to the limitation of the first method, we've designed a method using deep neural networks. We used a multi-layer perceptron to get the position of the pelvis tracker. Since data only contains a few float values, a simple network structure can get the ideal position of the trackers.

The network contains an input layer with 35 nodes, and 2 hidden layers with 50 nodes, and an output layer with 7 nodes, which is the desired pelvis tracker's position and quaternion rotation. Data is pre-processed before since we only need a relative position of each tracking device. The reason that we don't use absolute value is that players can free roam inside the area. The player's absolute position will change if the player moves, but the player's pose will not change, so it is more efficient if we use relative values.

Each input position values are processed with a simple calculation.

$$input_{x1} = mean_x - orig_{x1}$$
$$input_{y1} = mean_y - orig_{y1}$$
$$input_{z1} = mean_z - orig_{z1}$$

$mean_x$ is average value of the every other x values, and $mean_y$, $mean_z$ for vise versa. After converting data to relative position, data is converted within in range of -1 to 1. Then, we put them into a network to train a small network. We used Mean Squared Error to calculate loss, and optimized with Stochastic Gradient Descent. Every layer didn't used activation function.

This system needs to deployed while training. At first, the network is not optimized, so it is needed to train using players' data. Data will train if every tracking device is working. While training, the network tests its data and compares it with the original value and gets the error. If error became considerable, the network stops training and starts to deploy. If the network is not able to deploy, it will use the Naive method mentioned earlier. If the pelvis tracker is malfunctioning, the system inputs other devices' data into the network and gets output from the network. Then, it replaces data of the malfunctioning pelvis tracker's data and applies.

Also, there is a possibility to user change their pose, which is data not trained. So, the network is keep evaluated, and go back to training mode if the average error is below the threshold. When it happens, it will train again using new data given by the user. Switching between training mode and deploy mode is required because biased data may disturb the training of the network. Algorithm 1 simply abstracts the implementation of this method.

V. Result

We evaluated the result of our method by creating a fake tracker in the system and replacing data with calculated data.

Algorithm 1: Smart network training method

```
while Every frame do
    if Every tracker is working then
        if isNetworkReady is false then
            Train DNN Network once every 5 frames.
        end
        Evaluate network and save to table. Every 100
         frames, Check network's average error. If is
         below threshold, set isNetworkReady to
         true. Else, set isNetworkReady to False.
    else
        If isNetworkReady is true, assign pelvis
         tracker's data from DNN Network. Else, use
         Naive method.
    end
end
```

TABLE II
ACCURACY COMPARISON OF NAIVE METHOD. DIST MEANS 3-D DISTANCE. DIST, POSX, POSY, POSZ'S DOMAIN IS CM, ROTATION'S DOMAIN IS RADIAN.

Dist	PosX	PosY	PosZ	RotX	RotY	RotZ
13.67	6.55	7.57	7.42	0.76	1.43	1.85

A. Accuracy

Accuracy is compared when every tracker is working. We compared data between real tracker's value to calculated value. Result of the Naive method is on II, and DNN method is on III.

For naive method, distance between real value and calculated value is not so far. Considering the human body's length, displacement about 10cm is may not cause big problem. Still, those calculated result is observed on Idle(standing) movement, so if a player take other actions, then the disorient of the pelvis tracker may gone far. Also, rotation values are quite inaccurate, maybe because tracker's real rotation degree and calibrated rotation is different, that may cause large displacement on the rotation.

A method using DNN is quite optimistic. As shown as III, overall result is significantly better than naive method. Especially, rotation values are much more accurate. However, we observed that loss suddenly increases when player takes other poses, but this is not a major problem. Since network is small and easily trainable, network can train new values faster and available to deploy in a short time.

TABLE III
ACCURACY COMPARISON OF DNN METHOD. DIST MEANS 3-D DISTANCE. DIST, POSX, POSY, POSZ'S DOMAIN IS CM, ROTATION IS QUATERNION ROTATION.

Dist	PosX	PosY	PosZ	RotX	RotY	RotZ	RotW
6.57	4.64	0.510	4.62	0.0111	0.00579	0.00816	0.0148

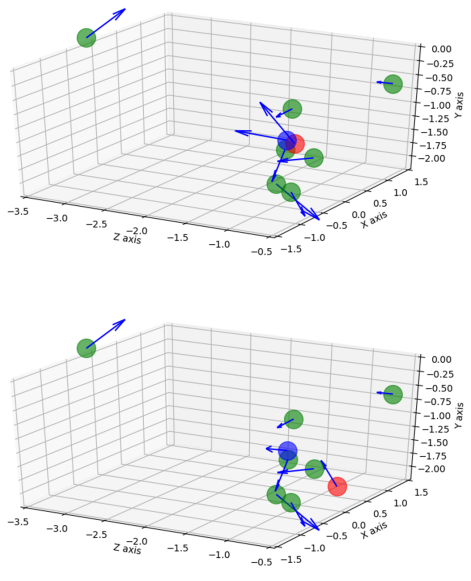

Fig. 5. Position of tracking devices expressed on 3-D area, Naive Method. Green circle indicates other tracking devices, Red circle indicates measurement data from pelvis tracker, Blue circle indicates calculated data from pelvis tracker. Up: Pelvis tracker is functioning, Down: Pelvis tracker is malfunctioning

B. Execution Time

To apply the solution, the algorithm is needed to be executed as fast as possible to get fair response time. Differed by VR systems, usually, the VR headset has 90 to 120 Hz frequency. Based on this frequency, our targeted execution time is less than 8 milliseconds that can make the algorithm to work with the VR system.

Execution time of navie method is around 0.000061 milliseconds on the test environment, which is far below from baseline. That's because this method simply calculates based on existing values.

For the DNN method, it's also far below targeted excution time, since network is very simple. If we train network every 5 correct data, average execution time is around 0.055ms. It is far below our target goal, also there are enough leftover time to compute additional technique to improve the network.

VI. Future Goal

Upon researching, we figured out that LSTM(Long Short-Term Memory) models can works well with our solution, since it has memory cells to remember data. Applying LSTM to this solution may increase the performance.

VII. Conclusion

We've developed a pose auto-fix system for full-body inverse kinematic tracking users who uses more than 3 Vive trackers. Upon executing, this algorithm will adjust the position and rotation of the pelvis tracker if it malfunctions. The algorithm gathers data from the user's action if it is valid, and trains data into the network. Code is available at https://github.com/r3coder/ViveTrackerAutofix

References

[1] Desai PR, Desai PN, Ajmera KD, Mehta K (2014) A review paper on oculus rift—a virtual reality headset. Int J Eng Trends Technol (IJETT) 13(4):175–179
[2] VRChat Team, Full Body Tracking, Online Documentation, https://docs.vrchat.com/docs/full-body-tracking
[3] Jerrith, Vive Tracker Demo (5 trackers, using VRIK from Final IK), Youtube Video, https://www.youtube.com/watch?v=l_yD6U247qs
[4] KAT loco, Website, https://katvr.com/products/kat-loco
[5] NOLO VR, Website, https://www.nolovr.com/index
[6] Azure Kinect DK, Website, https://azure.microsoft.com/ja-jp/services/kinect-dk/
[7] HTC Vive Tracker, Website, https://www.vive.com/kr/vive-tracker/
[8] Czernuszenko, M., Sandin, D., & DeFanti, T. (1998, May). Line of sight method for tracker calibration in projection-based VR systems. In Proceedings of 2nd international immersive projection technology workshop (pp. 11-12).
[9] Peer, A., Ullich, P., & Ponto, K. (2018, March). Vive tracking alignment and correction made easy. In 2018 IEEE conference on virtual reality and 3D user interfaces (VR) (pp. 653-654). IEEE.
[10] Niehorster, D. C., Li, L., & Lappe, M. (2017). The accuracy and precision of position and orientation tracking in the HTC vive virtual reality system for scientific research. i-Perception, 8(3), 2041669517708205.
[11] S. Lee, IDSLab Paper Review Seminar 2020/03/18, (2020), Youtube, https://www.youtube.com/watch?v=Diw_Erus1Z4
[12] Steam Hardware & Software Survey: Apirl 2020, (2020), https://store.steampowered.com/hwsurvey/Steam-Hardware-Software-Survey-Welcome-to-Steam
[13] HTC, HTC VIVE Tracker (2018) Developer Guidelines Ver 1.4, (2019)
[14] N. Dalton, VRPlayspaceMover, (2018), GitHub repository, https://github.com/naelstrof/VRPlayspaceMover