# Laboratório VISGRAF

## Instituto de Matemática Pura e Aplicada

**Tensorpose: Real-Time Pose Estimation using Tensorflow for Interactive Applications**

*Luiz Schirmer, Djalma Lucio, Luiz Velho, Helio Cortes*

Technical Report TR-19-03 Relatório Técnico

March - 2019 - Março

The contents of this report are the sole responsibility of the authors.
O conteúdo do presente relatório é de única responsabilidade dos autores.

# TENSORPOSE: REAL-TIME POSE ESTIMATION USING TENSORFLOW FOR INTERACTIVE APPLICATIONS

A PREPRINT

**Luiz José Schirmer Silva**
GALGOS
PUC-Rio

**Djalma Lucio**
Visgraf
IMPA

**Luiz Velho**
Visgraf
IMPA

**Hélio Côrtes**
DasLab
PUC-Rio

March 14, 2019

**ABSTRACT**

In this technical report, we describe the TensorPose architecture for motion capture applications. We also present some changes in the reference work to speed up the inference network and create temporal coherence.

***Keywords*** Pose Estimation · Convolutional Neural Networks · Motion Capture

## 1 Introduction

This work aims to use techniques that involve deep neural networks to motion capture and apply them to the animation of humanoid characters and other related applications. The project is based on the work of Cao et.al.[1] which generated the OpenPose project developed by Carnegie Mellon University. We use OpenPose's deep neural network model for identify human skeletons in images that can be provided by a video or in a real-time capture. To increase performance, we made some modifications to the original model, as well as to post-processing activities.

## 2 Pose Estimation

### 2.1 Openpose model

In this section, we will analyze the general network aspects proposed by Openpose. The network architecture consists of a feedforward neural network that predicts an set $S$ of 2D "confidence maps" from which the body parts are located in an image and also a set $L$ of 2D vector fields that help identify the connection between two body parts to generate a skeleton. The set $S = \{S_1, S_2, ..., S_n\}$ has n "confidence maps" one for each part of the body (hand, elbow, head) and the set $L = \{L_1, L_2, ..., L_C\}$ has C 2D vector fields, each used to construct the skeleton, identifying some member of it, i.e., an arm or leg for example. $N$ candidates for each body part are generated, creating for each pair a bipartite graph. Finally, the results are analyzed following a greedy strategy, using the Hungarian algorithm with some modifications to create a connection between two parts. This process is performed each frame.The figue 1 show the network architecture.

### 2.2 Network architecture

The neural network is divided into two branches: one responsible for generating the "confidence maps" and other the affinity fields. Also, these two branches are divided into t stages. The data are prepossessed by the first ten layers of a VGG-19 convolution network, generating a set of $F$ features for the first stage of the network in each branch. As a result of the first stage, we have a set $S$ of confidence maps and a set $L$ of Affinity Fields (vector fields), where $\rho^1$ e $\Phi^1$ represent the convolution networks. At the end of this step the results are concatenated with the initial set F. At each subsequent stage, the predictions of both branches, together with the features of the original image, F, are concatenated and used to produce refined results as we can see in the equations 1 and 2.

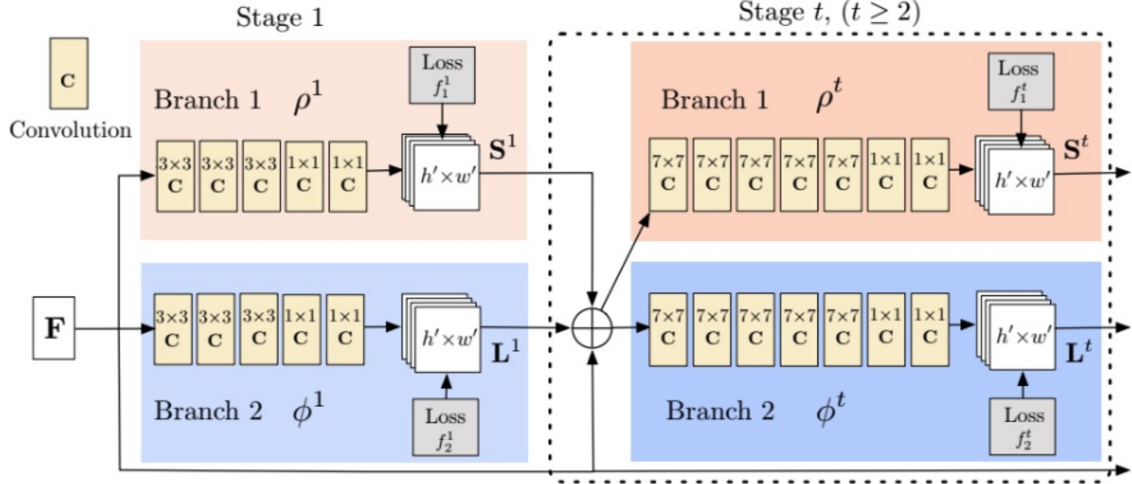


Figure 1: OpenPose Model

$$S^t = \rho^t \left(F, S^{t-1}, L^{t-1}\right) \forall t \geq 2, \tag{1}$$

$$L^t = \Phi^t \left(F, S^{t-1}, L^{t-1}\right) \forall t \geq 2, \tag{2}$$

To guide the network in its predictions, two objective functions are given at the end of each stage, one for each branch. The L2 norm is used between the network output data and the ground truth generated for the confidence maps and vector fields. The objective functions for both branches are defined by equations 3 and 4:

$$f_S^t = \sum_{j=1}^{J} \sum_{p} W\left(p\right) ||S_j^t\left(p\right) - S_j^*\left(p\right)||_2^2 \tag{3}$$

$$f_L^t = \sum_{c=1}^{C} \sum_{p} W\left(p\right) ||L_c^t\left(p\right) - L_c^*\left(p\right)||_2^2 \tag{4}$$

where $S_j^*$ e $L_c^*$ represent the ground truth of the annotated data, $J$ and $C$ are respectively the confidence maps for each part of the body and the vector fields, and also, $W\left(p\right)$ a binary mask with $W\left(p\right) = 0$ where there is no data annotated at the point p of image. The mask is used because there are images in the training dataset where they are not entirely annotated, and this mask is used to prevent true positive data from being penalized. The overall loss function is defined by equation 5:

$$f^T = \sum_{t=1}^{T} (f^t_S + f^t_L), \tag{5}$$

where $T$ represents the number of stages. The training images, as well as their annotations, were obtained through the COCO (Common objects in context) dataset [2] , using more than 150000 images for the task.

Following the architecture model, we first analyze the set $S$ to identify the possible candidates for a part of the body. In total, 18 reliable maps are generated by branch 1 defined by the convolution network $\rho$.

Each confidence map of the set $S$ can be viewed as a Heatmap. For this, we use a technique called Non-maximum suppression to identify the position of potential candidates. Such a method is described below in the algorithm 1:

With all possible candidates obtained (points), we need to find the actual connections between them. Therefore, for each connection, we have a complete bipartite graph where each node is a part of the body and each edge is a connection representing an arm or leg,for example. To find the right relationship, we face a well-known problem of graph theory to find the best match between the vertices of a bipartite graph: an assignment problem. But to solve it is necessary to find the respective weights of the edges.

**Algorithm 1:** Non maximum supression

**Data:** 2D Heatmaps
**Result:** Image peaks
1 Start on the first Heatmap Pixel
2 Surrounding the first pixel , create a window of side 5 and find the maximum intensity value inside that area and substitute the value of the center pixel for that maximum.
3 Slide the window one pixel and repeat these steps until cover the entire heatmap.
4 Compare the result with the original heatmap. The remaining pixels with same value are the peaks we are looking for. Suppress the other pixels setting them with a value of 0.

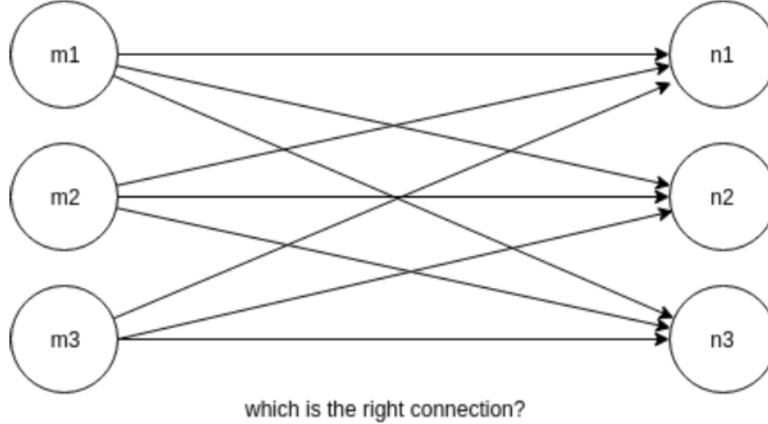


Figure 2: Bipartite graph without weights

For this task, we use the vector fields that were generated by the output of the network. We compute an integral line along the segment described by each pair of candidates and since the integral line shows the influence of a given field along a line, the equation 6 gives us the scores of each segment represented by each pair of body parts.Given two points $d_1$ and $d_2$ as candidates for two parts of a skeleton forming a connection, the score of an edge connecting these two points are given by the following equation:

$$E = \int_{u=0}^{u=1} L_c\left(p\left(u\right)\right).\frac{d_{j2} - d_{j1}}{||d_{j2} - d_{j1}||_2} du \tag{6}$$

where $L_c$ represents the vector field and $p(u)$ interpolates the position between two body parts $d_{j1}$ e $d_{j2}$,

$$p\left(u\right) = \left(1 - u\right) d_{j1} + u d_{j2} \tag{7}$$

Afterwards, to obtain the optimal solution, the problem solution can be achieved through a classical method called Hungarian Algorithm. Each score is then associated with an edge, and the assignment problem is solved as follows:

**Algorithm 2:** Assignment Problem

**Data:** PAF Scores
5 Sort each possible connection by its score.
6 The connection with the highest score is indeed a final connection.
7 Move to next possible connection. If no parts of this connection have been assigned to a final connection before, this is a final connection.
8 Repeat the step 3 until we are done.

Regarding the detection of multiple people, determining the pose of each becomes an n-dimensional matching problem. Since this is a problem considered NP-Hard, some restrictions can be made. First, a minimum number of vertices for each body segment is chosen to generate a spanning tree instead of having the complete graph. After, the problem is

Figure 3: 2D Pose Estimation

decomposed into n bipartite graphs to determine the matching between adjacent nodes independently. With all the connections defined, we make the union of all the connections that share common vertices, together with each other, to create the skeleton as a whole. The figure 3 show the final result of the pose estimation.

## 3 Tensorpose Network and Skeleton Tracking

To increase the performance of the network, some modifications have been made. Initially, we modified the generation of the first features using Depthwise Separable Convolution of the MobileNet [3] network that is typically used for embedded devices. We replace the original VGG network because the performance in this step represents the first bottleneck. Here, we decrease the number of operations required in the convolution network process,making an approximation through successive convolutions. Depthwise Separable Convolution deals not only with the spatial characteristics of the image but also with depth, i.e.,the RGB channels.In this process, a used kernel is separated into 2 to perform two convolutions: a depthwise convolution and a pointwise convolution. Let's assume that a 12X12X3 image is given as input to the network. A depthwise convolution iterates the kernel separately on each of the channels of the image. So, suppose we have a 5x5 kernel applied to each channel, in the end, we will have an 8x8x3 image. A pointwise convolution has a kernel that iterates over each pixel of the image separately and has a depth relative to the number of channels in the image. In this way, as in the given example, we iterate a 1x1x3 kernel on the output 8x8x3 to generate an 8x8x1. Depending on the operation we can create multiple kernels to get the desired output.For example, 256 kernels generate an 8x8x256 output. As stated, this step was performed to decrease the number of operations required and to approximate the expected result. In a traditional convolution we would have, for example, 256x3x5x5x8x8 = 1,228,800 operations. With the new model we would have, 3x5x5x8x8 = 4,800 and 256x1x1x3x8x8 = 49,152 operations. Adding the two steps, we would have 53952 operations, which is much less than 1228800 originals. Based on the architecture of the mobilenet, 12 layers of convolution were used instead of 10 of the original paper.

Another problem identified was the lack of temporal coherence in the original paper; that is, there is no relation between the objects defined in one frame to another. In this way, the reference is lost for each identified person. For the context of our applications, it is necessary to create a module for this task. To solve this problem, in the first experiment, we create a module for temporal coherence using optical flow. Optical flow is a pattern of apparent motion of objects in images between two consecutive frames caused by the movement of an object or the camera. At first, we used the Kanade-Lucas-Tomasi algorithm [4] and preliminary tests not only presented an increase in the performance of the technique, considering the frame rate but also a smooth in the motion capture.

Initially, we pass, for the KLT algorithm, the points of skeletons detected by the network. At each interval of t frames, these points are iteratively tracked through the optical flow, where the previous frame, the position of the previous points and the next frame are passed to the detection function. With each new frame, the location of each point of the tracked skeletons is updated. If any position or point are lost in the process, the tracking is stopped. Then, the network is executed again and the skeletons recalculated. The same is done at the end of the interval of t frames, even if no information is not lost to guarantee the consistency of the tracking. The optical flow assumes that the intensity of the pixels of an object does not change with time and neighboring pixels have the same pattern of movement. Classical approaches as optical flow fail when it comes to environments with illumination changes and long-range motions. To

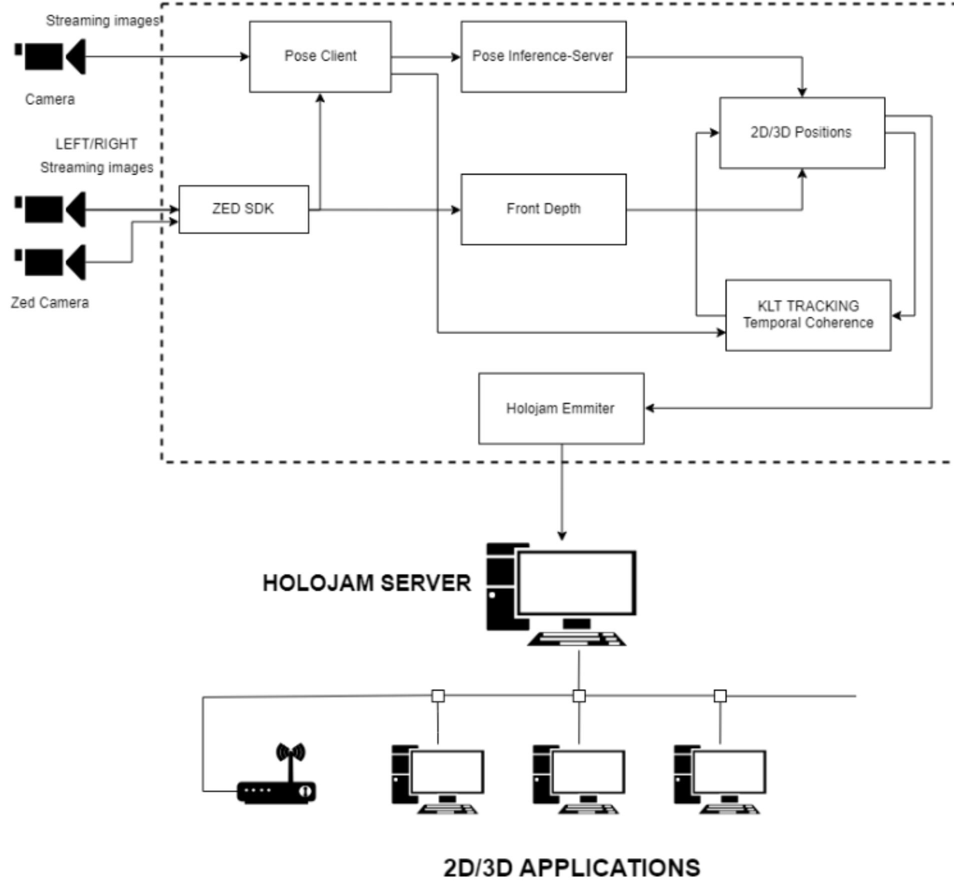


Figure 4: Tensorpose Architecture

solve these issues, we use the Robust Local Optical Flow [5] following the framework proposed by Senst et. al. taking into account an illumination model to deal with varying illumination.

Also as support to the tracking process, we use the Kalman and Multiple Instance Learning (MIL). The Kalman filter [6] is used to predict the position of the skeleton in the frame after the actual and to ensure that the references for each skeleton detected are maintained. Given the distance between the points calculated by the network and those predicted by the Kalman filter, a threshold is given to ensure that the reference points are maintained. We use the MIL [7] just to ensure the reference of detected person is maintained. We use as an additional feature to identify the bounding box of detected head keypoints and to maintain the reference of a person, even when the network is called. Although, their use is optional.

## 4 TensorPose Architecture

Tensorpose was developed with the aim of being a platform for the development of 2D and 3D applications that involve motion capture. In this section, we will describe the general aspects of its architecture, as well as the framework for network communication of different applications. We have developed an architecture model for 2D and 3D applications for motion capture, where each module is independent in terms of video processing, the inference of captured skeletons, temporal coherence and information transmission. The figure 4 represent the architecture model.

### 4.1 Motion Capture Architecture

**Pose Client** This module is responsible for processing video data from different cameras. Here each frame is prepared using the OpenCV library, where we also do an equalization of the histogram to improve the contrast in the images and send the data to the inference module or the temporal coherence module.Also, here we reduce the image resolution to 386x656 to fit in GPU memory.

**Pose Inference** The neural network here receives data from the pose client and performs the inference of the skeletons in an image. Also, post-processing is performed here to solve the assignment problem. For each set of key points detected for each person, a dictionary, as follow, is created containing the 2D position of each body part and the connections between them. For each recognized person is given an Id and their corresponding dictionary is added to a list of tracked persons who are sent the module responsible by instantiating objects that will represent the capture of those people.

```
skeletons = {
    people_id:{
        Nose : [],
        Neck" : [],
        RShoulder : [],
        RElbow : [],
        RWrist : [],
        LShoulder: [],
        LElbow : [],
        LWrist :[],
        RHip = : [],
        RKnee : [],
        RAnkle : [],
        LHip :[],
        LKnee :[],
        LAnkle :[],
        REye :[],
        LEye :[],
        REar :[],
        LEar :[],
    }

}
```

Listing 1: Skeleton dictionary

**Zed SDK** Here we use the ZED stereo camera SDK for 3D data capture. This module was developed using the Python API where data from a 2D image is sent to the Pose Client to be processed and the 3D data to the depth module.

**Depth Front** This module is responsible for processing the depth data of one or more ZED cameras and receiving intrinsic camera data.The ZED has two cameras separated by 12 cm which capture a high-resolution 3D video of the scene and estimate depth and motion by comparing the displacement of pixels between the left and right images. This module stores a distance value Z for each pixel in the image.

**2D/3D Positions** It receives data from the positions inferred by the network and creates objects identifying each detected person. In case of 2D, keeps the positions received in coordinates of the image. In the case of 3D, it receives data from Front-Depth and performs transformations in 2D coordinates for positions in the 3D world. It is also responsible for passing information to the temporal coherence module, also processing the Kalman filter, identifying the points to be tracked and ensuring the reference of the detected skeletons.

**Temporal Coherence** Track the points using the KLT algorithm. It receives the initial data detected by the network, and every t frames perform the tracking of the points.

**Holojam Emmiter** It uses the framework of the Holojam platform to send the captured data via the network. It is a Python client that communicates with the Holojam Server.The Holojam platform is described in the following section.

## 5 Holojam

Holojam is a virtual space sharing platform developed by the Future Reality Lab of New York University coordinated by Prof. Ken Perlin. The platform consists of the Holojam Node library and the Holojam SDK project.This platform enables content creators to build complex location-based multiplayer VR experiences in a simple, unified Unity project. The development framework provides an extensible and clean interface, allowing for rapid prototyping and extension. Additionally, it abstracts away specific VR hardware, promoting a flexible and customizable creation of virtual reality experiences.

**Holojam Node** It is a client-server library developed in Node.js and targeted to virtual reality applications running on a local network. One of its main characteristics is to perform low-latency communication between the various clients and the server.It consists of the following components: relay, emitters, sinks and clients.

The relay component is responsible for routing the messages (updates and events) in a Holojam network. It acts both as a central server which collects data received through a preconfigured (upstream) address, as well as performs a broadcast of this data through a multicast (downstream) address.

In addition to the relay in a Holojam network, there are several nodes, also called endpoints. Endpoints are either emitters, which only send upstream data; or sinks, which only receive data through the downstream; or clients that receive downstream data and send upstream data. Holojam Node also provides a WebSocket interface where you can receive and transmit data over the web.

### 5.1 Holojam Protocol

Packets routed through a network are either hosted or updated. An update is essentially an array of flakes (generic Holojam objects). An event has an array containing only one flake. There is also a notification that is an abstraction of an event that includes just the label.

#### 5.1.1 Holojam Objects

Holojam provides two types of objects: Nuggets and Flakes. These objects are defined through Google's FlatBuffer Interface Description Language (IDL).

Nugget Composition:

- It is event type or update. Default is update
- It is mandatory to have an array of flakes

```
enum NuggetType : byte { UPDATE, EVENT }

// Message (update or event)
table Nugget {
   scope : string; // Namespace
   origin : string; // Source

   type : NuggetType = UPDATE;
   flakes : [Flake] (required); // Data array
}
```

Flake Composition:

- A label is mandatory

```
table Flake { // Data container
   label : string (required);

   // Optional data

   vector3s : [Vector3];
   vector4s : [Vector4];

   floats : [float];
   ints : [int];
   bytes : [ubyte];

   text : string;
}
```

There are also the Vector3 and Vector4 that can be used in Flakes:

```
struct Vector3 { // XYZ
   x : float; y : float;
   z : float;
```

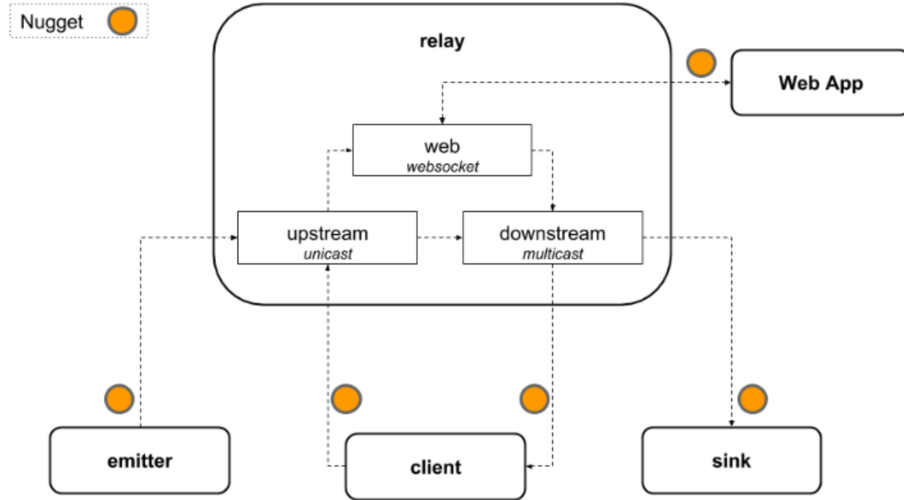


Figure 5: Holojam Architecture

```
4 }
5 struct Vector4 { // XYZW
6     x : float; y : float;
7     z : float; w : float;
8 }
```

### 5.2 Holojam Architecture

In the figure 5 , it is possible to verify how communication takes place in a Holojam network. Here an emitter/client/WebSocket sends a Nugget, and it is received by the relay through the upstream port and to then be sent to the sink/client via the downstream.

### 5.3 Holojam SDK

The Holojam SDK project is a Unity3D project containing all the elements needed to create a multiplayer virtual reality application. This project provides a C API that allows the application created to use or extend this API allowing rapid prototyping. Also, it abstracts the configuration of the VR hardware used in the project. One of its main components is the implementation of the Holojam client in C. Thus all Holojam objects can be integrated easily into the Unity project.

### 5.4 Holojam and Tensorpose applications

All components of the Holojam platform were used in the TensorPose project applications, both in 2D applications as well as 3D and VR applications. Since TensorPose is implemented in Python, then the components needed to use Holojam in TensorPose were also deployed in Python, including a serializer for the keypoints and the emitter. The serialization of the obtained keypoints was implemented from the IDL FlatBuffers of Holojam. With this, it is possible to send the keypoints to the relay, and any sink/client/websocket that is connected to this relay can access the data.

The figure 6 shows the components for various applications created in the TensorPose project and which component each is related to.

## 6 Experiments

The work began by experimenting Openpose, and by identifying possible improvements in the model since it is an open source library. The models generated for Openpose were developed using Caffe [8], which is a framework for deep learning model development. Caffe was discontinued, and for this reason and performance issues, the model was converted to Tensorflow. To improve the initial performance of the network, following the idea of convolution pose machines [9], the VGG19 used was replaced by the Depthwise Separable Convolution of the MobileNet network.

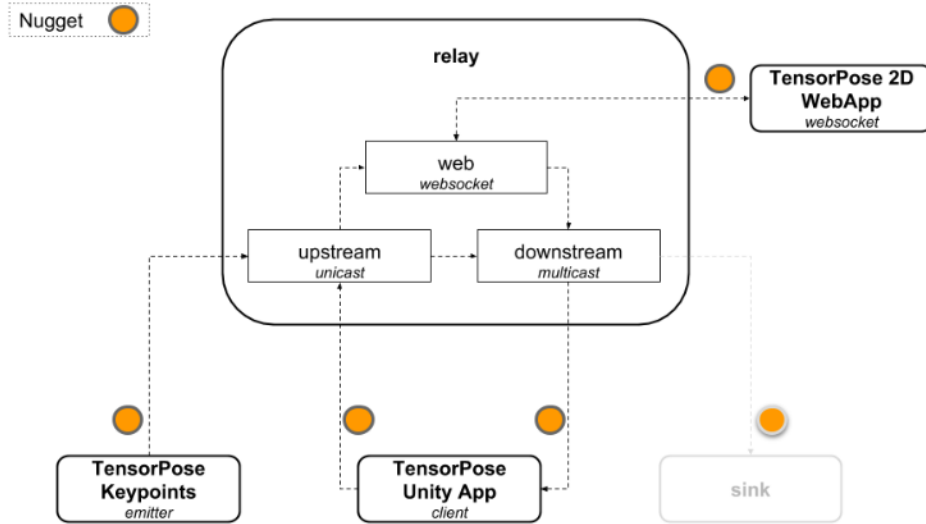


Figure 6: Tensorpose Apps

| GPU | CPU | OpenPose | TensorPose |
|---|---|---|---|
| Nvidia Tesla P40 | Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz | ~10 FPS | ~34.43 FPS |
| Nvidia Titan RTX | Intel(R) Core I9(R) CPU @ 3.3GHz | ~11. FPS | ~36.8 FPS |
| Nvidia TITAN XP | Intel(R) Core I7(R) CPU @ 3.3GHz | ~7.28 FPS | ~27 FPS |
| Nvidia GTX 960 | Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz | ~3.73 FPS | ~12 FPS |

Concerning Openpose, in this stage, performance improvements from 10 fps to 15 fps have already been presented. The application code was re-implemented in Python, including the post-processing steps, such as the line integral calculation and the assignment problem, using Cython. The network was implemented with 6 stages for each branch, without taking into account the extraction of the features by the layers of the mobilenet. In training, 117576 images were used and in the validation 2400 with a maximum of 200 iterations per epoch. To avoid calculating the line integral and the execution of the Hungarian algorithm at each step, we use sparse optical flow. We get not only performance improvements, with a frame rate above 30 fps, but also a smoothing in motion tracking, also due to the Kalman filter. In the first tests, we defined a fixed interval of 10 frames, where the optical flow is processed and after its parameters updated by a new processing step in the neural network. Such range was defined empirically, where we saw that there was not a significant difference in the accuracy of the tracking. In the original paper, the runtime consists of two major parts: (1)CNN processing time whose runtime complexity is $O(1)$, constant with varying number of people; (2) Multi-person parsing time whose runtime complexity is $O(n^2)$, where n represents the number of people. For 9 people, the parsing takes 0.58ms, and the CNN takes 99.6ms. Our strategy reduces the overall time for tracking and increases the frame rate, as we can see in table 6.

Subsequently, with the trained network, as we said before, an architecture was defined for the creation of applications for motion capture, connecting different cameras and applications, such as scenes in the Unity3D engine and web applications.

The initial module was developed for 2D applications where capture can be done through ordinary cameras such as webcams. Each frame is captured and sent to the detection module, which is in charge of performing the inference and detection of the people in each frame. Subsequently, the data regarding the captured persons, are sent via the net using the framework Holojam [10].

In addition to the 2D capture, tests involving stereo cameras were also conducted. We used the ZED camera to map three-dimensional coordinates of the world from two-dimensional coordinates of images. The skeletal position is initially processed in the same way, but in a later step, it is transformed into 3D space using intrinsic camera data and depth information. As in the previous process, the capture and processing modules are independent, and Holojam is also used to send the information.

Figure 7: Unity3D app

### 6.1 Applications

Some applications were developed as a proof of concept using the Unity 3D engine and also WebGL. Regarding applications in Unity3D, the data is sent via the network by the Holojam Server, using multicast. This application was developed for the use of multiple devices, but it is also possible to use unicast connections. Regarding the 3D scene in the Unity engine, "Holojam Unity" objects implement support for multiplayer, which includes communication with the Holojam Server, components with position recognition and Avatar presence. These objects have a script component called "Holojam Network" which contains some fields indicating how to communicate with the server.

The "Multicast Address" field indicates the IP address used to receive data from the server. This field is not relevant if there is already a unicast connection between this client and the server since in this case, the client will receive the data directly from the server. The Server Address field can be used to indicate the IP of the server directly. This address is used to send data to the server, but if the indicated address is not local (not starting with 192), Holojam will ping this address, which creates a connection between the server and this client. All data that is sent and received uses an update data structure. When Holojam sends an object, it uses the label "SendData" for such an update. For the virtual characters, they are created from the tracked data of the real person using inverse kinematics. A manager implemented in the application reconstructs the entire body of the Actor from pairable elements corresponding to the connections between hands, elbows, shoulders, neck, head and so on. The reconstructed data are used as targets for a 3D model in the scene.The figure 7 show the Unity3D application.

Similarly, a web application with a unicast connection was developed as we can se in figure 8. The entire application was designed in javascript using WebGL. A script called "Manager" was implemented, responsible for receiving the data sent by the server, considering each object which identifies each person tracked and the position of their skeleton. The Manager, instance objects called characters, for the identification of actors and still be in charge of updating their positions to each frame. The Manager also manages the removal of actors from the scene, if they are no longer in it or the track has been lost. Subsequently, the characters are rendered as a ragdoll using the position of each part of their skeleton directly, not using here inverse kinematics.

## 7 Conclusion

We described the architecture of TensorPose project for the development of real-time motion capture applications. This kind of application can be used to create multi-user and interactive applications for VR and computer animation. As a prof of concept we present two main applications using Unity3D and WegGL integrated with Holojam plataform. As future work, we aim to explore possible ways to speed up the network inference. To do that, we can decompose the convolution layers of the network using Tensor decomposition, such as High order SVD [11].

## 8 Acknowledgement

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research.

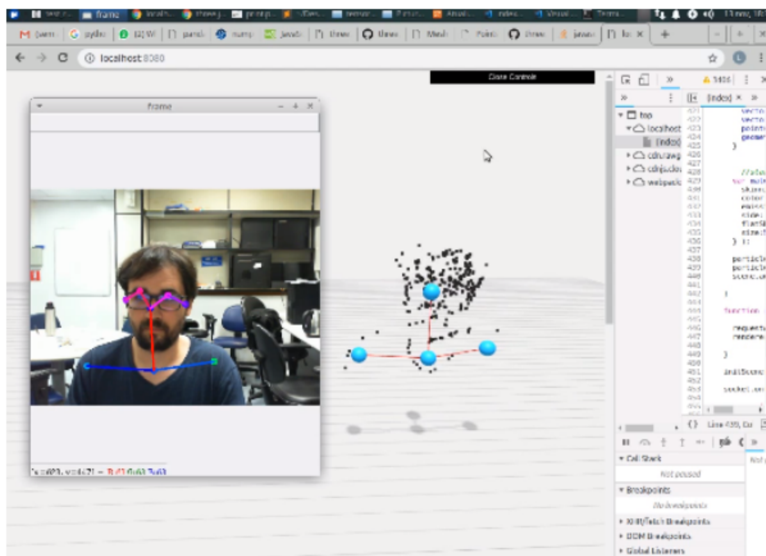

Figure 8: Tensorpose Web Test

## References


[1] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In *CVPR*, 2017.

[2] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[3] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[4] Jun-Sik Kim, Myung Hwangbo, and Takeo Kanade. Realtime affine-photometric klt feature tracker on gpu in cuda framework. In *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, pages 886–893. IEEE, 2009.

[5] Tobias Senst, Jonas Geistert, and Thomas Sikora. Robust local optical flow: Long-range motions and varying illuminations. In *IEEE International Conference on Image Processing*, pages 4478–4482, Phoenix, AZ, USA, September 2016. IEEE. IEEE Catalog Number: CFP16CIP-USB ISBN: 978-1-4673-9960-9 DOI:10.1109/ICIP.2016.7533207.

[6] Charles K Chui, Guanrong Chen, et al. *Kalman filtering with Real-Time Applications*. Springer, 2017.

[7] B. Babenko, Ming-Hsuan Yang, and S. Belongie. Visual Tracking with Online Multiple Instance Learning. In *CVPR*, 2009.

[8] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[9] Shih-En Wei, Varun Ramakrishna, Takeo Kanade, and Yaser Sheikh. Convolutional pose machines. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4724–4732, 2016.

[10] Terrence Masson, Ken Perlin, et al. Holo-doodle: an adaptation and expansion of collaborative holojam virtual reality. In *ACM SIGGRAPH 2017 VR Village*, page 9. ACM, 2017.

[11] Shuo Zhou, Nguyen Xuan Vinh, James Bailey, Yunzhe Jia, and Ian Davidson. Accelerating online cp decompositions for higher order tensors. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1375–1384. ACM, 2016.