**Auto-calibration of Multi-camera System Based on Human Joints**

A hand holding a black box

Description automatically generatedA pair of black cameras on a table

Description automatically generated2024/07/01～2024/07/07: High-Speed Camera Settings and Setups  
we couldn’t finish this within the deadline since we needed the help of seniors who had used the camera previously, the camera used was Emergent Vision HR-3000-S-C depth camera for object capturing.

The power cable and the port for connecting it to the computer use a network cable (supposedly LAN, it also needed 網卡). So as an alternative, we decided to use 2 IPEVO Ziggi-HD Plus USB Camera.

A close up of a device

Description automatically generatedThe setup process is ongoing, and we are still figuring out ways to simultaneously use these cameras for both human keypoint detection and feature matching (HigherHRNet and SuperGlue). Initial guesses would be to use 3rd party applications such as OBS Studio, etc.

Human Joint Point Detection Algorithm Testing: 2024/07/08～2024/07/28

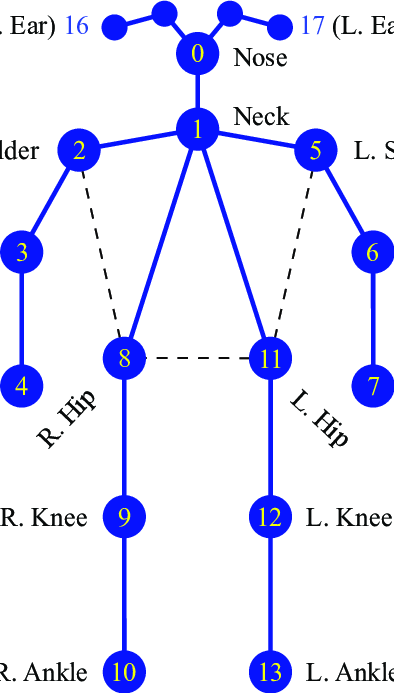
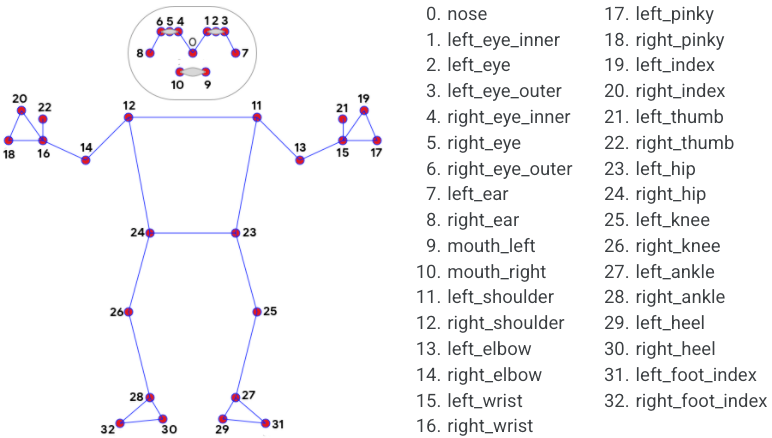
We researched 3 human joint detection algorithms, MediaPipe, OpenPose, and HigherHRNet. Below are key notes that were observed when experimenting with these algorithms as well as a bit of info of their performance. The evaluation process is carried out by using `pycocotools` from the coco Python API on the COCO 2017 dataset, specifically the `eval 2017` folder.

A computer screen shot of a computer code

Description automatically generated

Imports that were used for the mediapipe evaluation. Imports for evaluation of the other 2 models remain similar in terms of numpy, pycocotools, cv2, as well as os. Differing in only the import of the models.

**Mediapipe**: Really easy to setup as it is one of the more traditional algorithms for human joint detection, only requires us to download mediapipe from pip, which is really simple compared to when building from source. Detects 33 keypoints on the human body as shown. But keypoints are conflicting with that of COCO’s keypoints which only has 17 keypoints. The left one shows the COCO 17 keypoints format while the right one shows the mediapipe 33 keypoints format.



To counteract this, we made a rough estimate on the keypoints and made some changes in the keypoints detection of mediapipe, we removed the extra keypoints as well. The left image is the part of the code that converts the keypoints. The result can be seen on the right image, 1826 images were processed which took approximately 6 to 7 minutes to finish. The AP score is relatively low, our initial guesses were that it had something to do with the amount of keypoints, and the conversion wasn’t as smooth as we thought it would be.

Takeaways: Easy to use, easy to setup, relatively smooth for a traditional algorithm, 33 keypoints.

A screenshot of a computer

Description automatically generatedA screenshot of a computer code

Description automatically generated

**HigherHRNet**: Needed to be cloned from a repository on github, <https://github.com/xingyizhou/Higher-HRNet-Human-Pose-Estimation>, the process was relatively simple, but a few dependencies needed to be taken care of, as well as the versions of each dependencies, but nonetheless quite a simple process. Has 5 pretrained weights that can be accessed from google drive, with each weight having tradeoffs (performance vs. speed).

A screenshot of a computer program

Description automatically generatedA screen shot of a computer

Description automatically generatedA computer screen shot of a computer code

Description automatically generatedA screenshot of a computer

Description automatically generatedwe are currently using w32 512 (default) weight for all our experiments. Can be initialized, in which we could control the device that it’s using which is really convenient and it also boosts performance.  
HigherHRNet in total images processed 2317 images, took approximately 8 – 9 minutes to finish. The AP score are relatively high since there is no difference in keypoints from the COCO dataset, which is 17. In the `misc.visualization.py` script, there are a lot of methods for visualizing the keypoints, such as coco, mpii, etc.

takeaways: we used simple-hrnet repository (https://github.com/stefanopini/simple-HigherHRNet) since it simplifies the model into a class that could be easily imported and used in ipynb scripts. Fast, High AP score.

**OpenPose**: the Python API for openpose was believed to be really hard to build when using CMake and Visual Studio. The dependencies such as caffe, boost, which are deeplearning frameworks are also unable to be found and downloaded due to the website being down, I had to look up github discussions and download it from other users’ google drive. At the end, with high AP score, I was unable to build the openpose python API which makes openpose hard to use.

**Final Decision for pose estimation algorithm: HigherHRNet**

Data Processing & Final Touches: 2024/07/29～2024/08/31:

A screenshot of a computer

Description automatically generatedA screenshot of a computer program

Description automatically generatedWe found that when we use cv2 to grab the image, and then process the image using the model, before the visualization process, it returns a numpy.ndarray with dimensions (1, 17, 3).

The first 2 are the y and x coordinates, and the last one is the confidence score. (y, x, c). Conversion is also needed for both coordinates and confidence. These datas can be said as the anchor point of the 2 different camera views. As of now all experiments are only conducted using our personal laptop camera (1 view).

-need to consider time when calibrating (timestamp)

A computer with a screen showing a couple of images

Description automatically generated**Update:**

We have tried to simultaneously open two cameras, and it was successful, no lag was encountered.

A screen shot of a computer program

Description automatically generatedAt first, we were thinking of detecting the joints simultaneously. There were also some errors during the script writing process. It has something to do with cv2 closing and opening processes. At the end we changed the code as follows:

But then the display was very laggy as the computer has to process two camers’ input and do some pose estimation with a GPU.

A computer screen shot of a program

Description automatically generated

We tried changing the resolution, to make it lower, but it turns out that it has not much of a difference. So at the end we decided to take pictures initially and then process it afterwards, in addition to creating a new directory. We then wrote some scripts to take n number of images, based on the amount we have set prior to the execution of the script.

(just for note, Press 'q' to quit, 'c' to capture 1 image, 't' to capture n amount of images)

In our experiment, we tried taking 20 pictures for each camera, namely cam1 and cam2. We then store these images in the directories cam1 and cam2. We then access a total of 40 images from both directories and do pose estimation on them.

A computer screen shot of a program code

Description automatically generatedsince there are no lag anymore, we can set the resolution to a higher one, we also changed the weight to w48 with a resolution of 640 which will have more accurate predictions but at the cost of slower processing time, but we think this could be ignored since we are not doing real-time detection. In the process of this, the channel, c, is also changed to 48. In addition to that we also stored the joint coordinates in a txt file to make the overall process simpler to export this data into a different script.

A screenshot of a computer program

Description automatically generated This is the overall hierarchy. Cam -> images

Cam\_pose -> images with joints

.txt files -> address of each images, coordinates

Feature Point Matching (SuperGlue): 2024/09/01～2024/09/30 (we finished this part at 25/08/2024 which way before the deadline)

We then have to make the superpoint algorithm to find these joints. Superpoint generates a lot of feature points, based on how “special” that specific point is, whether it’s color difference, something that sticks out, a corner. So we thought of two methods:

1. Retrain the superpoint model so that it only detects the joints from higherhrnet
2. Or, we just constrain the model so that it detects feature points only on the coordinates that are part of the joints.

We tried experimenting the second method first.

A screen shot of a computer program

Description automatically generatedThis was how we “constrained” the points. We set a tolerance range, in which the points (feature points from superpoint) that are near the joints’ positions will be selected with a tolerance of n distance. So the algorithm is not point perfect, but it’s so small that it can be ignored, so we experimented with various tolerance ranges.

**Why do we need to process these joints in superpoint? Because superpoint will output three things [coordinates, confidence, and descriptors]. Descriptor is the most important as it tell us what are the “nearby” surroundings around the point (joint). This is important for the matching process.**

Jacknote:

HigherHrNet superglue As we all known, higherhrnet can get the human joint and return its (y,x,confidence),However ,according to the paper, we need to treat the joints as keypoints and match it using superglue, but the problem is that we don’t have the descriptors that get from superpoints ( superpoints use heatmap and only het background keypoints )which means we are not able to match the joints in both pictures using superglue. We got two solutions for it, the first one is to rebuild the model and then train the module to get the human joint instead of the background. Even though this is feasible but the process is complicated and not practical, so that we came up with another solution which is we don’t rebuild it, instead, we steal it😂

basically the idea is that since the superpoints use heatmap to get the keypoints, which means if We lower the threshold, then the human joints supposed to be included inside the keypoints we got from superpoints, In order to implement it, We first try to use the camera to get two pictures so we can compare it, then we return the joints using higherHRnet, but there’s something we need to notice is that the joints get from hrt is in the order of (y, x, confidence) so we have to reverse the x , y coordinate first then return the list of keypoints(joints) of both pictures. Then we use them compare to the keypoints we got from superpoints, but we then found another problem. Which is that since we set the new\_keypoints to the keypoints that match the joint, so the descriptors and scores (superpoints will return keypoints, descriptors, scores - so I thought each keypoint should have its corresponding score and descriptor) won’t match the new\_keypoints In order to solve this problem, we get the index of the new\_keypoints so we know the position of it in its original list? Then we can get its corresponding score and descriptors by the index! ( notice that descriptors are an 256-dimensions vector) new\_score = score[index] New\_descriptors = descriptors[:, index]

jack note: match keypoints 和 get\_matching\_indices 因為是用joints跟superpoints 的keypoints 來做比較，也就是說由Simple HigherHRNet 取出的joints, superpoints裡的keypoints有可能會沒有，於是我們加了tolerance讓他們可以有誤差，簡單來說我們使用的都是superpoint裡抓到的東西，所以不只match keypoints 需要tolerance, index 也會需要，因為index也是利用相同道理得到的A screen shot of a computer code

Description automatically generated

(no more jacknote)

This is the example of the matching algorithm results:

A screenshot of a computer screen

Description automatically generated

So now the overall process is:

multicamera.py (take pic from 2 cameras, save in cam1 and cam2, process joints, save in cam1\_pose and cam2\_pose) 🡪 superglue\_multi.py (read txt files, match the names, find the keypoints of joints with its descriptors, feature match the images)

we will now find a way to find the intrinsics, we have found a way to utilize face based calibration, using depth perception. But when forking the repository there were some version issues that needs to be resolved. In detail, it needs pptk in which it’s only available in the older versions of python which is python 3.6, but some of the scripts utilize features that are only present in the newer versions of python (the one that we are using now). We are now considering some alternatives to pptk

a thing to consider is that we need to put a clock when we are taking pictures, to make sure that we take the pictures in the same time.

Our overall workflow rn:

Multicamera.py(capture images, capture pose estimation, capture checkerboard, find K1,K2) 🡪