REALTIME COMPUTER INTERFACE CONTROL WITH HUMAN POSE ESTIMATION

Ang Boon Yew, Tea Lee Seng, Yang Xiaoyan

Institute of Systems Science, National University of Singapore, Singapore 119615

ABSTRACT

Pose estimation is the task of estimating recovering joint positions from a set of given input parameters, such as an RGB image of a human body pose. Traditional feature extraction techniques and deep learning methods have been developed to estimate joint positions based on these inputs. Popular models based on deep learning such as OpenPose by CMU labs have been developed and show to have good performance for this task. The estimated pose and joint locations can be further used in tasks such as human activity recognition and movement detection. In this project, we leverage on an open source pose estimation model, tf-pose-estimation[1], a TensorFlow implementation of the OpenPose model to develop an intelligent computer interface control system. Using the extracted joint positions from the estimated poses, we trained a separate classifier model to detect specific human body movements, which are then used to trigger mouse movements and events on a web browser on the computer interface.

1 Introduction

1.1 Introduction to Pose Estimation

In relation to human activity recognition, pose estimation is the task of extracting and recovering a set of joints on the entire human body or a specific a body part, given a set of input images [2]. These inputs include images such as 2D RGB images, 3D point clouds, RGB-D Images (Depth images) or kinematic model based inputs such as skeleton models. Joint positions are estimated based on features extracted from these inputs using different methods. The number of joint positions vary between different models and specific body parts in order to cater for the specificity of joints required. Using the extracted poses as estimated joint location heat maps or joint coordinates, further classification for tasks such as activity recognition and analysis can be performed.

1.2 Applications of Pose Estimation

Pose estimation models have been used in a variety of tasks, such as human activity recognition, movement gait analysis and event detection. These tasks have much potential in areas such as surveillance, where continuous monitoring by manual human interaction is not feasible in order to detect security threats and events. In these scenarios, pose estimation can be used to detect suspicious behaviour and accidents such as falls, allowing timely intervention to these events.

Pose estimation can also be used to classify actions with the human body, such as hand waving, sitting, or other canonical poses. Using the recovered poses, events or actions can be triggered in another connected system. For example, the act of waving a hand can be detected and classified by a pose estimation model and trigger a door control system to open a door [3]. These methods are particularly useful in scenarios where physical human interaction with control interfaces are limited or unfeasible, such as medical surgeries or handling of hazardous materials.

1.3 Challenges of Pose Estimation

However, pose estimation also faces challenges in accurate joint position recovery due to self-occlusion of body parts, ambiguity in joint locations and multiple bodies in one image. These challenges are currently being addressed in research work, using more complex methods such as depth maps and multi-view methods [4][5]. Although these methods may result in more accurate pose recovery results, there are scenarios where only a 2D RGB image may be feasible due to cost and location constraints. For example, it may not be cost effective for a regular user to install a infrared depth camera on a computer notebook, who then has to rely on webcam images instead. Hence, methods to utilize limited inputs can also be considered for integration with such tasks.

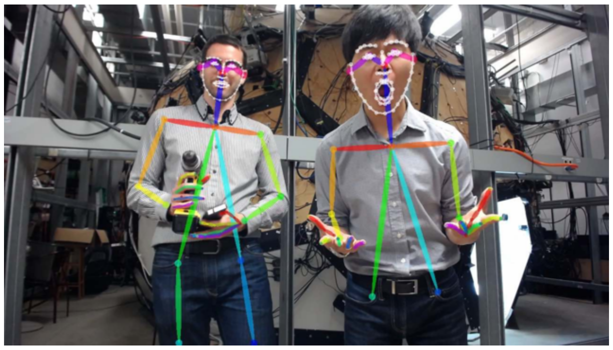


Fig. 1. Example of joint estimation using OpenPose[2]

1.4 Proposed Approach

In this project, we focus on the use of single person pose estimation and leverage off an open source pose estimation model, **tf-pose-estimation**[1], a TensorFlow implementation of the CMU OpenPose model, to perform pose estimation [2][6]. The extracted pose joint positions are used to develop an intelligent computer interface control system that allows a user to control mouse movements using specified human body actions. The pose estimation model is used to extract joint positions from 2D RGB webcam images for a sequence of video frames. These joint positions are then trained with a separate classifier model to recognize and classify certain body actions, which then trigger mouse movements and actions on the computer Graphical User Interface (GUI) screen.

2 Related Literature

Extensive research work has been done into pose estimation tasks in recent years, most notably with the rise of popular deep learning methods to achieve better model performance. Pose estimation methods largely fall into two categories: single person pose estimation and multi person pose estimation [2]. We focus on the review of single person pose estimation methods as in our study, the focus is on pose estimation on a single user with a fixed camera viewpoint.

2.1 Pose Estimation Techniques

Traditional methods for pose estimations rely on the use of local features around joint positions, and inferring other joint positions based on spatial correlations between these joints. Tree-structured models exploit the spatial relationships between adjacent or connecting joints based on a known prior kinematic model, such as between a mid-arm joint and the shoulder joint in order estimate joint positions [7][8][9]. Deep learnings employing the use of Convoluted Neural Networks (CNNs), which have achieved success in image recognition tasks, have also been explored with favourable results and performance [10][11][12][13]. Toshev & Szegedy use a stacked series of pose regressors using Deep Neural Networks (DNNs) in order to increasingly improve the precision of the joint estimates and is known as the DeepPose model [14]. Tomson et al. proposed a combination of CNN part-detectors and a spatial model to resolve the constraints between the joint locations [13]. The OpenPose model, developed by Cao et al. is a recent, popular model of pose estimation and provides high performance models for estimating joint positions for different body parts. It uses predicted part affinity fields, a method to describe the association between joints and confidence maps to locate joint positions, as well as to differentiate between multiple human bodies, making it effective for both single and multi person pose estimation tasks [2]

2.2 Human Activity Recognition

For human activity recognition and movement classification tasks, various methods using varying inputs have also been explored. Reiss et al. combine the use of inertial sensor data as well as a bio-mechanical model in order estimate the movements of the joint positions and the associated activity [15]. Thurau & Hlav utilize Histogram-of-Gradients (HoGs) descriptors extracted for single image frames and compare them a known distribution of histograms for each action class [16]. Biswas & Basu use depth images captured from Microsoft Kinect devices to extract region based histograms and use differences between frames to build a motion profile and classify human actions [17]. More notably on the use of image sequences, Chevalier uses an LSTM network architecture to recognize human activity based on sequences of recovered joint positions [18].

3 Methodology

In our approach, we first collected and annotated the training video sequences manually and use them with the **tf-pose**[1] estimation model in order to extract the joint positions. The joint positions are then used with a classifier model, such as a CNN or an LSTM model in order to classify them into possible actions. At the same time, changes in the joint positions are monitored and used by the system moving the mouse cursor across the screen. An architecture of the system is shown in Figure 2.

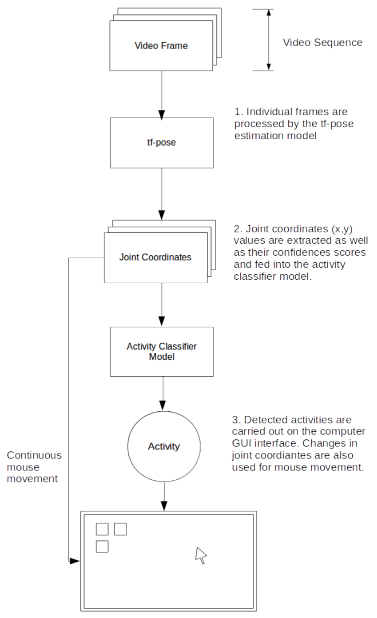

Video Frame
Video Sequence
1. Individual frames are processed by the tf-pose estimation model
tf-pose
2. Joint coordinates (x,y) values are extracted as well as their confidences scores and fed into the activity classifier model.
Joint Coordinates
Activity Classifier Model
3. Detected activities are carried out on the computer GUI interface. Changes in joint coordiantes are also used for mouse movement.
Activity
Continuous mouse movement

Fig. 2. Architecture of the proposed interface control system

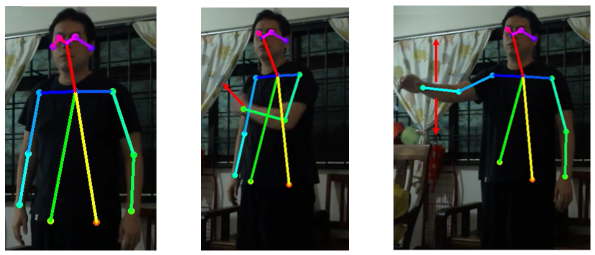
3.1 Training Dataset

In our study, we define 3 main types of human movement activity to control the mouse movement and the actions on the computer GUI interface:

1. Looking: Looking at the screen. No action. This is surprisingly important state to reduce significant noisy operation. we choose to stand with hands down. The most comfortable standing pose.

2. Righthandnext: Switch to next tab. Equivalents to Control Tab shortcut key in chrome.

3. Mousemoving: Simulates mouse movement. To scroll pages up and down.



(Looking) (Righthandnext) (Mousemoving)

Fig. 3. Classes of actions

We collected a total set of 9 training sequences, with 3 sequences for each class using a portable webcam and processed them using the **cheese** utility application on Ubuntu. After further training and reviews, we noted that **cheese** introduces motion blur to the recording. Hence, we recommend using opencv webcam API recording in future, which is also the video streaming tool for PoseNet.

3.2 Pose Estimation

We leveraged on the **tf-pose-estimation**[1] Python package to perform pose estimation on our collected images in order to retrieve the 19 predicted joint coordinates along the *(x,y)* axes, each referring to the horizontal and vertical coordinates of the joint in the image. The **tf-pose-estimation**[1] package is a custom implementation of the original OpenPose model using Tensorflow and includes trained graph files such as MobileNet for deployment. The package also includes accessible APIs for running inference on a webcam as well as offline processing. A sample of the inference result in show in Fig. 5. The 19 joint coordinates and their associated parts used in this model are:

1. Nose = 0
2. Neck = 1
3. RShoulder = 2
4. RElbow = 3
5. RWrist = 4
6. LShoulder = 5
7. LElbow = 6
8. LWrist = 7
9. RHip = 8
10. RKnee = 9
11. RAnkle = 10
12. LHip = 11
13. LKnee = 12
14. LAnkle = 13
15. REye = 14
16. LEye = 15
17. REar = 16
18. LEar = 17
19. Background = 18

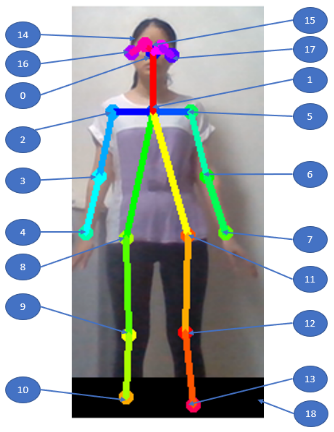


Fig. 4. The pose estimation joint coordinates

In addition, the confidence scores of the joint coordinates were also recorded as raw data for training. The total feature vector resulting from each video frame image contained 57 real figures. For joint positions that were not captured in the video, a value of 0 was used in place of the coordinate values.

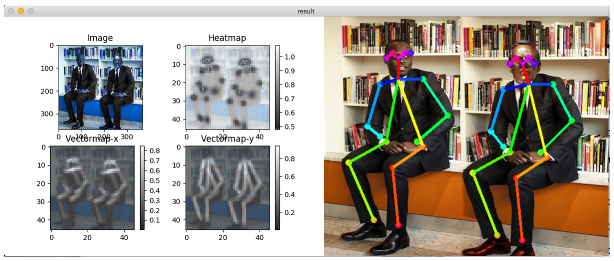


Fig. 5. Inference results using tf-pose-estimation[1]

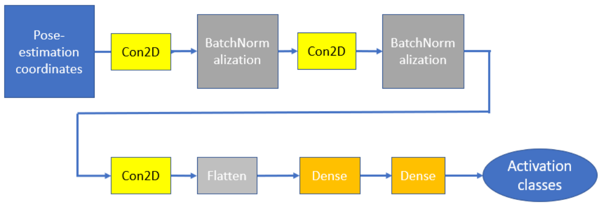
3.3 Action Recognition

3.3.1 Initial Setup

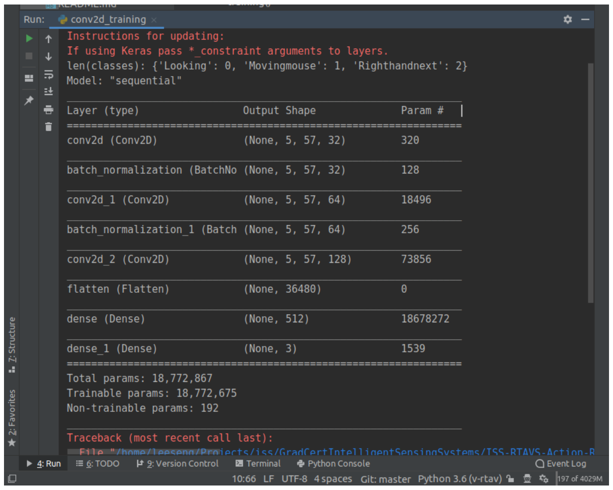
The estimated joint coordinates are then used as inputs to a classifier model in order to classify the video sequence into one of the 3 specified actions. Initially, we collected a few other action clips e.g. raising hands and palms down. We also try to explore capability of **pyautogui** [19]. A demo of mouse movement control was created as **sample_left_hand_moving_mouse.py** To generate pose data for next stage Action classification model training, we developed a script to run poseNet[1] and dumped the data to training-csv folder. The script can be run as **sample_build_training_data.py**

3.3.2 Initial Model with Full Features

For the action recognition tasks, we implemented a simple 3 layer 2D CNN to recognize the action pose of a sample of 5 frames with 57 joint features. In this initial approach, 5 frames from the video sequence are used in the action inference step using the pose estimation model in order to determine the human action in a moving window fashion. In successive steps, the oldest frame in this sequence of 5 frames is discarded and the current frame is used instead. This method is used to stabilise the predictions of the recognized activity over a fixed history period.



As the sequences have to be generated for each training step, we did not use the Keras ImageDataGenerator API and used the model.train() API for the training, which provides us control over the each loop. For each training iteration, we generated the training data with by sampling 5 image frames. e.g. random start frame plus 4 following frames randomly selected from next 10 frames. As we only have about 3 clips for each action, the total number of frames was less than 1000.



```
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
len(classes): {'Looking': 0, 'Movingmouse': 1, 'Righthandnext': 2}
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 5, 57, 32)         320       
_________________________________________________________________
batch_normalization (BatchNo (None, 5, 57, 32)         128       
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 5, 57, 64)         18496     
_________________________________________________________________
batch_normalization_1 (Batch (None, 5, 57, 64)         256       
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 5, 57, 128)        73856     
_________________________________________________________________
flatten (Flatten)            (None, 36480)             0         
_________________________________________________________________
dense (Dense)                (None, 512)               18678272  
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 1539      
=================================================================
Total params: 18,772,867
Trainable params: 18,772,675
Non-trainable params: 192
_________________________________________________________________
Traceback (most recent call last):
  File "/home/leeseng/Projects/iss/GradCertIntelligentSensingSystems/ISS-RTAVS-Action-R
```

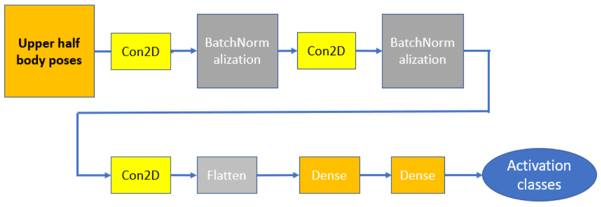
Fig. 6. conv2d neural network for 57 features

As a result of this training design, online validation on a separate validation set during the training process of the model was not performed. Our experiment shows that a small number of training epochs did not have result in good accuracy for the trained model, while longer epochs shows high accuracy scores and low loss values. Without the validation accuracy score, we saved the weights based on minimum loss value from the training process.

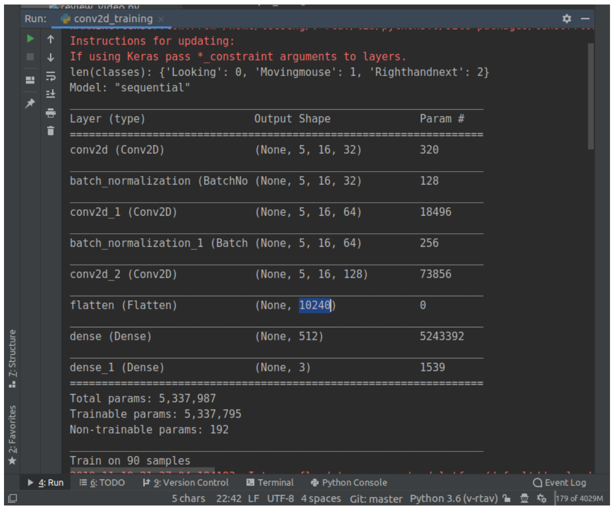
Although the training metrics were favourable, the action recognition ability was not as accurate in practice when integrated with the application to control the user interface. We also noticed some training clips won't get recognized due to motion blur from fast motion. The training process can be initiated using **conv2d_training.py**.

3.3.3 Updated Model with Feature Selection

Based on previous iteration's results, we train the model without confidence scores and only included upper half body poses. The video clips were also re-captured in order to obtain clearer video sequences with slower motion.

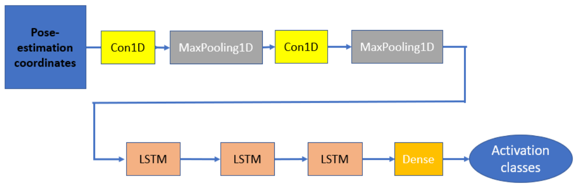


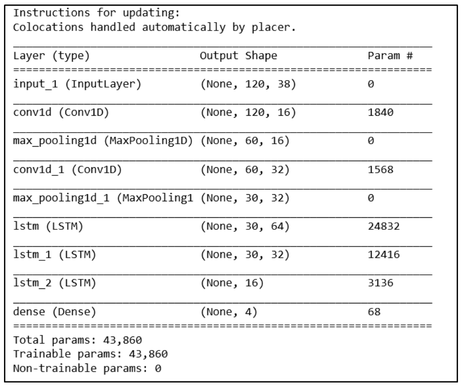
With these changes in the extracted features, the model was able to recognize the Looking and Mousemoving actions more accurately. However, the performance was not suitable for current tasks such as simple web surfing. In addition, we also noted that when we stand lower to bottom right area in the camera, there is a slight chance of triggering other ac-



```
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
len(classes): {'Looking': 0, 'Movingmouse': 1, 'Righthandnext': 2}
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 5, 16, 32)         320
_________________________________________________________________
batch_normalization (BatchNo (None, 5, 16, 32)         128
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 5, 16, 64)         18496
_________________________________________________________________
batch_normalization_1 (Batch (None, 5, 16, 64)         256
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 5, 16, 128)        73856
_________________________________________________________________
flatten (Flatten)            (None, 10240)             0
_________________________________________________________________
dense (Dense)                (None, 512)               5243392
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 1539
=================================================================
Total params: 5,337,987
Trainable params: 5,337,795
Non-trainable params: 192
_________________________________________________________________
Train on 90 samples
```

Fig. 7. Conv2d neural network for 16 upper body features





```
Instructions for updating:
Colocations handled automatically by placer.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 120, 38)           0
_________________________________________________________________
conv1d (Conv1D)              (None, 120, 16)           1840
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 60, 16)            0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 60, 32)            1568
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 30, 32)            0
_________________________________________________________________
lstm (LSTM)                  (None, 30, 64)            24832
_________________________________________________________________
lstm_1 (LSTM)                (None, 30, 32)            12416
_________________________________________________________________
lstm_2 (LSTM)                (None, 16)                3136
_________________________________________________________________
dense (Dense)                (None, 4)                 68
=================================================================
Total params: 43,860
Trainable params: 43,860
Non-trainable params: 0
```

Fig. 8. LSTM network architecture

tions. A quick test on any newly trained model can be using **quick_test_demo.py**.

3.3.4 Final Model with Centered Neck Coordinates

With previous findings, we noted that the training data may have had different pose centres, which was causing an mean shift along the features and thus affecting the accuracy of the model. With this consideration, we fixed the neck position at 0.5 of screen width and 0.3 of screen height, calculated the difference from neck and apply the difference to the coordinates of the joints in other body parts.

In this iteration, the trained model is able to tracking 3 different poses accurately enough for next stage of Google Chrome browsing interface integration within 1 week effort.

3.4 Experiments with LSTM Architecture

With the new processed coordinates, we also experimented with using an LSTM network architecture in order to perform the action recognition task. An initial approach using the whole video sequences with the joint coordinates at each frame as feature vectors was used and were padding with 0 values to standardize the training sequence length. However, we noted that this approach may not be practical for integration with the realtime interface control function as the actions have to be recognized continuously. Using the same training frame sampling approach, the LSTM model also had similar performance to the CNN model.

3.5 Computer GUI Input Control

In order to control mouse movements and other interactions with the computer interface, we use the **pyautogui** package to develop an application for controlling using the predicted actions[19]. The **pyautogui** package provides Python APIs to control and simulate computer inputs, such as the mouse movement or keyboard strokes, allowing for the development of custom tools to control interfaces, which can be triggered by other means. The recognition action states from the trained classifier model are processed continuously and used to trigger the events using these APIs. We develop a custom application using this package and integrate this with the trained action recognition model. The details of the actions are listed in Table 1.

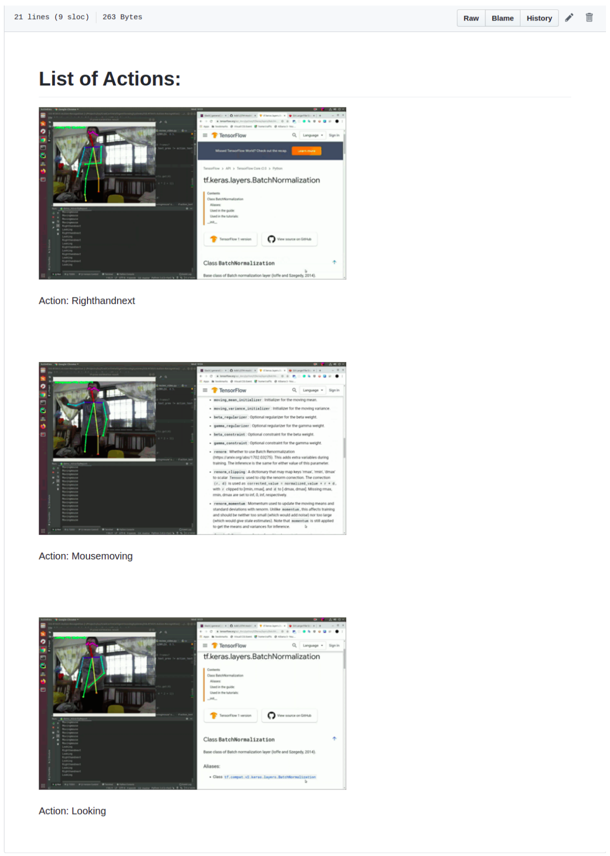
Actions	Purpose	Commands
Looking	No actions. A state for human to think what's next.	NOOP
Mousemoving	Scrolling page up and down. Scrolling up if palm is above neck level. We have another demo about moving mouse cursor at **sample_left_hand_moving_mouse.py**	pyautogui.vscroll(-1) or pyautogui.vscroll(1)
Righthandnext	right hand swipe to left.	pyautogui.hotkey('ctrl', 'tab')

Table 1. Details of Actions

4 Results and Discussion

Using this application, we tested the capability of the system by using a webcam to stream a video input of human actions to control a web browser on the computer. Through the trained actions such as Movingmouse and Righthandnext, we demonstrated control over a Google Chrome web browser by scrolling through the page and switching between tabs. For this implementation, the model was able to track the actions in most iterations and carried out the task correctly. An example of the Righthandnext interaction is shown in Fig. 9

However, the use of video streaming also present its own challenges to this UI interaction task. The action commands generated using human movement are translated into actual movements on the computer interface via state change mechanism, which required detections on state changes. For the Mousemoving action, the state change mechanism was not applied in order to cater for continuous page scrolling. A demonstration of the effects of the actions can be viewed at List of Actions page, Fig.9



21 lines (9 sloc) | 263 Bytes

List of Actions:

Action: Righthandnext

Action: Mousemoving

Action: Looking

Fig. 9. List of Actions. Click here to find out more.

The chrome browsing application can be tested using the **demo_minorityReport.py** script. The Chrome web browser should be initiated before running the script and put in focus of the window once the script is running.

4.1 System Limitations

Our system currently captures relatively large movements, such as an arm movement in order to trigger an action. However, certain actions may be better represented using a more detailed body part, such as hands gestures. For capturing finer gesture movements for controlling the interface, a hand pose estimation model would be more suitable. Although Google's Hand Tracking project was a possible option, there is currently a lack of Python APIs available for us and as the working examples are only available in Android and C++, we did not pursue this option [20]. We also attempted to use the Google-tensorflow-lite-pose-estimation tool, which is a mobile version of the pose estimation tool developed by Google [21]. However, the output provided by the model was not as intuitive and may not be easily integrated with a control application. As the **tf-pose-estimation**[1] package was built on top of the original OpenPose model, it was also a more viable option for use in this project.

Due to the quick prototyping nature for this project, we only prepared a small set of training videos for training and development of the end to end workflow. The small set of training videos was likely insufficient to train the classification model, for it to be more generalized to different video inputs and conditions. With more training samples, the performance can be improved in order to provide better and more accurate control over the computer interface. However, the system was still able to function despite the low number of training samples available, which may be attributed to the random sampling done during the training phase.

At the current stage, the human activity classifier for the different interface control actions is limited to the few specified actions that the model has been trained for. In order to cater for more fine actions, hand tracking is likely a must.

4.2 Future Work

The current system is that it utilises only a single RGB image in order to estimate the human pose and the joint positions, which might not work as well with positions that tend to have self-occlusion and high ambiguity. In order to further cater to a wider variety of possible poses and actions, the system can be augmented with inputs from another similar 2D RGB camera, which can then be used to take a different view of the pose and improve the accuracy of the estimated pose and joint positions in a fusion approach.

After further training and review on the input images, we noted that **cheese** introduces motion blur to the recorded clips. However the pose estimation model was unable to perform recognition on an area with motion blur. Hence the OpenCV webcam API recording might be a better option for future implementation.

For finer control, a hand pose estimation must be required for further integration. We look forward to prototyping an system that's similar to over-the-air user interface in movie, Minority Report, 2002.

5 Conclusion

In this project we used the **tf-pose** open source pose estimation model in order to determine joint locations of a human pose in an image and developed a human activity classification model with computer user interface API, **pyautogui** to control inputs to the browser.

We mainly demonstrated the scenario of web browsing through human actions and without any physical contact with the computer itself, which could be a useful approach in similar scenarios. e.g. controlling system with huge projection on building wall.

Through this project, we learnt to leverage on the capabilities of well-trained model such as OpenPose and integrate them with other models and tools in order to create a usable solution.

We hereby provide our GitHub repo,
https://github.com/XiaoyanYang2008/ISS-RTAVS-Action-Recognition[22], for anyone interested to explore further.

6 References

[1] ildoonet, "Deep pose estimation implemented using tensorflow with custom architectures for fast inference," `https://github.com/ildoonet/tf-pose-estimation`, 2019.

[2] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh, "OpenPose: realtime multi-person 2D pose estimation using Part Affinity Fields," in *arXiv preprint arXiv:1812.08008*, 2018.

[3] Michael B Holte, Cuong Tran, Mohan M Trivedi, and Thomas B Moeslund, "Human pose estimation and activity recognition from multi-view videos: Comparative explorations of recent developments," *IEEE Journal of selected topics in signal processing*, vol. 6, no. 5, pp. 538–552, 2012.

[4] Liuhao Ge, Hui Liang, Junsong Yuan, and Daniel Thalmann, "Robust 3d hand pose estimation in single depth images: from single-view cnn to multi-view cnns," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 3593–3601.

[5] Helge Rhodin, Jörg Spörri, Isinsu Katircioglu, Victor Constantin, Frédéric Meyer, Erich Müller, Mathieu Salzmann, and Pascal Fua, "Learning monocular 3d human pose estimation from multi-view images," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8437–8446.

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

[7] Pedro F Felzenszwalb and Daniel P Huttenlocher, "Pictorial structures for object recognition," *International journal of computer vision*, vol. 61, no. 1, pp. 55–79, 2005.

[8] Deva Ramanan, David A Forsyth, and Andrew Zisserman, "Strike a pose: Tracking people by finding stylized poses," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. IEEE, 2005, vol. 1, pp. 271–278.

[9] Yi Yang and Deva Ramanan, "Articulated human detection with flexible mixtures of parts," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 12, pp. 2878–2890, 2012.

[10] 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*, 2016, pp. 4724–4732.

[11] Umer Rafi, Bastian Leibe, Juergen Gall, and Ilya Kostrikov, "An efficient convolutional network for human pose estimation.," in *BMVC*, 2016, vol. 1, p. 2.

[12] Alejandro Newell, Kaiyu Yang, and Jia Deng, "Stacked hourglass networks for human pose estimation," in *European conference on computer vision*. Springer, 2016, pp. 483–499.

[13] Jonathan J Tompson, Arjun Jain, Yann LeCun, and Christoph Bregler, "Joint training of a convolutional network and a graphical model for human pose estimation," in *Advances in neural information processing systems*, 2014, pp. 1799–1807.

[14] Alexander Toshev and Christian Szegedy, "Deeppose: Human pose estimation via deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 1653–1660.

[15] Attila Reiss, Gustaf Hendeby, Gabriele Bleser, and Didier Stricker, "Activity recognition using biomechanical model based pose estimation," in *European Conference on Smart Sensing and Context*. Springer, 2010, pp. 42–55.

[16] Christian Thurau and Václav Hlavác, "Pose primitive based human action recognition in videos or still images," in *2008 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2008, pp. 1–8.

[17] Kanad K Biswas and Saurav Kumar Basu, "Gesture recognition using microsoft kinect®," in *The 5th international conference on automation, robotics and applications*. IEEE, 2011, pp. 100–103.

[18] Guillaume Chevalier, "Lstm for human activity recognition," https://github.com/guillaume-chevalier/LSTM-Human-Activity-Recognition, 2016.

[19] asweigart, "Pyautogui," https://github.com/asweigart/pyautogui, 2016.

[20] Google, "Hand tracking (gpu)," https://github.com/google/mediapipe/blob/master/mediapipe/docs/hand_tracking_mobile_gpu.md, 2019.

[21] Google, "Tensorflow lite pose estimation," https://www.tensorflow.org/lite/models/pose_estimation/overview, 2019.

[22] "Iss-rtavs-action-recognition," https://github.com/XiaoyanYang2008/ISS-RTAVS-Action-Recognition, 2019.