Real-time hand tracking

Project „Hot Topics in Computer Vision“

Yannick Schickel, Amin Jaber
Prof. Olaf Hellwich
8. August 2021

1. Introduction

This project deals with hand tracking also known in the field as hand pose estimation. As hand articulation is a crucial part in Human's non-verbal communication, estimating hand poses improves user-friendly human-machine operations.

Tracking algorithms can be roughly categorized in two method groups: the Appearance-based and model-based approach. Appearance-based methods run algorithms that are basically classifiers. In terms of pose estimation they're trained on generalizing to unseen poses. These algorithms are data-driven and need big and complete datasets in order to run robustly.

Model-based tracking methods on the other hand carry encoded knowledge, a so called model of the tracking object, and are capable of fitting the model to the unseen poses. These methods are not as data-linked, but need exact modeling.

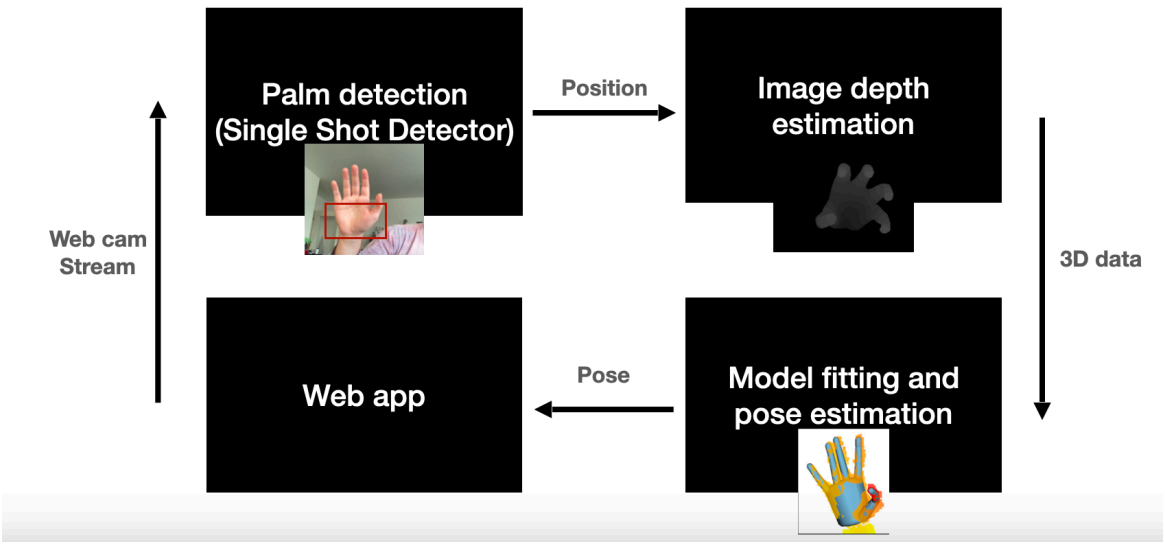

Image 1:

2. Project goals

Our team decided on the goal of implementing a real-time hand tracking via web application. Image 1 shows the prototype plan for our project implementation and the major milestones. Our hand tracking method both, appearance and model-based. We confident that this approaches a seamless hand registration and a robust pose estimation.

In order to offer the technology to many people as possible, we think implementing the hand tracking in a web application provides easy access. We are aware of the fact that this brings high visibility of our code. Since we're not running any confidential code, this is acceptable.

The user reaches the web application and will be asked to enable device's webcam stream. In the next step hand palms in the stream should be detected in real-time. This can be achieved by catching the stream and performing state-of-the-art object detection via Single Shot Multibox Detector. Once the positioning of palms in the frames are obtained, it will be necessary to estimate the depth on the hand region. The depth data employs the model fitting routines and ends up with the final pose estimation.

3 Palm detection

The state-of-the-art in Object detection algorithms today was introduced 2015 and is called Single Shot Multibox Detector (SSD). The SSD is a convolutional neural network and its architecture consists out of a base network for high quality image classification extended by multi-scale feature maps for detection. Tensorflow's Object Detection Model Zoo provides a collection of detection models pre-trained on the COCO 2017 dataset. Since our application needs real-time detection, we're going for one of the fastest models possible.

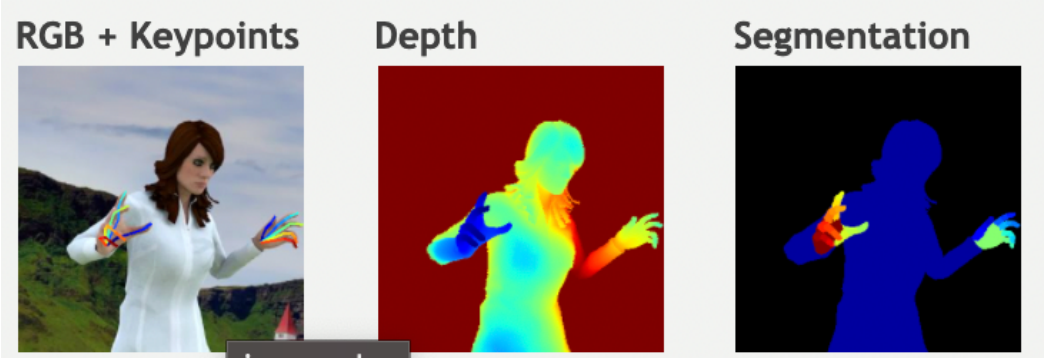

Image 2: Sample of the Rendered Handpose Dataset

Using a pre-trained model and training it on new data, is called transfer learning. The main benefit of Transfer Learning is faster generalization and training.

3.1 Dataset

For training our pre-trained SSD model to detect hand palms, we chose the Rendered Handpose Dataset from Uni Freiburg. It contains 41258 training samples, each consisting of RGB, Depth and Segmentation Masks (Image 2). Our detection model works with detection boxes and not with segmentation masks. Therefore we need to perform a pre-processing on the data to convert the segmentation into Bounding-Boxes.

3.2 Pre-Processing (Segmentation-to-BoundingBox)

Our goal is framing the segmented pixels of the hand palm with a bounding box. The algorithmic approach to achieve the framing, will be described in the following.

We're independently looking for the top left and the bottom right corner of the box. This can be achieved by scanning through the pixels from left to right and from top to bottom. Both scans each stop once we found the first palm pixel. The intersection of both scans gives us the top left coordinate of the bounding box, we were looking for. The bottom left corner can be computed by scanning from bottom to top and from right to left in the same way.

In the next step the data needs to be served as annotations in Pascal Visual Object Challenge (VOC) format. For every sample of our dataset we generate an XML-file label, that includes the class and location of the detections.

3.3 Training our SSD

The Training of our detection model was done in **PalmDetectionColab.ipynb**. Setting up the environment is a crucial part of the process and needs us to install libraries and Tensorflow Object Detection API. It is also the need for an additional format conversion of our dataset into TFRecords file format, Tensorflow own binary storage format.

Once this is set up, the training is straight forward and takes 7-8 hours for training 38k samples during 8000 epochs.
The resulting model performance could be validated by testing the real-time detection in the web application.

4 Web application

The web application (https://lake-ribbon-creator.glitch.me) includes three main files (index.html, script.js and style.css). In order to use the JavaScript- Version of Tensorflow and OpenCV, we included the the libraries using the recommended script tags in the html-file. Accessing the web-cam stream was challenging, but could be solved using the Chrome Browser. In order to perform object detection in our web app using a Tensorflow model, we need to transform the model into a TensorflowJS model. After conversion, we store the model on GitHub for easy import when needed.

4.1 Post-processing

While detecting hand palms in the stream, there are still multiple detections per region. In order to estimate the highest scored detection, we implemented a non-maximum suppression in JavaScript. The winner region of this suppression gets cropped and displayed in order to being feed to the depth estimation model.

Depth Image Estimation model

In our application we used a good and easy to use pre-built model. To run this model we needed to install the following dependencies: Python 3.7, PyTorch 1.9.0, OpenCV 4.5.1, and timm 0.4.5.

After successfully installing the model and testing it by executing throughout the cmd command panel. The next step was to implement the model in our web application by executing it from a JavaScript code, where the code spawns a child process (sub process) to run the model and produces the correct output as anticipated.

running the model:

simply navigate to “Handtracking\Handtracking\bin\Debug\netcoreapp3.1” and add images that are needed to process in the “input” file, then double clicking the green icon “App-win.exe” file. The results will show in the “output_monodepth” folder.

Hand fitting model

For the web application we used a pre-built model accompanied with a large size of data set that were needed to train and evaluate the model before implementing the model. This model was successfully running after installing the required dependencies: Python3.7, PyTorch 1.9.0, CUDA 10.2, cuDNN 5.1, tqdm, progressbar, scipy and for preprocessing phase of the model, it needed

the following to be installed: Matlab with add ons (Statistics and Machine Learning Toolbox, Computer Vision Toolbox and Image Processing Toolbox). For the data set luckily the developer team of this model left a separate download link to download their own data sets as an option. In addition before starting to run the preprocess phase, a small change must be done. for example In the **preprocess.m** file the **dataset_dir** directory must be changed depending on the dataset directory. and to run the model on the cloud other small changes had to be made on the code depending on the PyTorch version. The first phases of preprocessing of the data set were successfully concluded, by using Matlab installed on the pc.

Building the 3D hand model

To build a light and well performance 3d model and are compatible with web applications, we used the three js Library provided by Nodejs to create the 3d model. Many changes happened to the library, where they completely removed their support of building a 3d model with daze studio. Alternatively we used the available geometry and built them together visually to create a hand model. It has the ability to move depending on the coordination and Angel that are given later on from the hand fitting model.

From studying the paper provided in the Hand fitting model the coordinate of the finger may change the distance between the joints of the finger (example finger bones length) and this may result in an error, since the structure of building the hand was like a family, each part depend on the previous part, for example the first index joint is connected directly to the palm as a child of it and the first bone is also connected to the first Index Joint as a child and so on. And if the parents move, it also makes the child move along with it, at the same time the child holds their own local coordinate. A simple example is when moving the palm without moving any fingers all the fingers stay attached to the Palm while the palm moves.

To prevent an error to accrue caused by slight different distance coordinate, we created a function (name Angel) that are able calculate the Angle using the current coordinate point (first triangle point) with the new entered coordinate pointe (second triangle point) and for the third triangle point is the earlier connected point (direct parent of that child) with the joint in respect of each axes, then the result of the function is an Angel used to rotate the targeted joint. A small demo is already given in the code after manually entering a coordinate. The limitations are [3]:

- the fingers cannot reach a negative angel on the X axes
- finger joint angel limitations (only towards positive X axes)
- limitations of the first joints angel on the Z-axes
- no rotations around the Y-axes for the fingers

running the model:

to run the 3D hand model using cmd, root the path file called "3D_hand" then type in the command "npm start", it should open the browser and show the 3D hand model. all the code for the hand implementation is the src\ help.ts file.

Achievements

- Detecting the hand palm was fully functional trained to an acceptable degree of 50% accuracy and was implemented into the web application.
- Depth Estimation pre-build model is fully functional but were unable to implementing it in a web application due security restrictions.
- Hand fitting model: unfortunately an unfinished model due to lack of memory capacity and failing to find a way around it by editing the codes in the model
- 3d Hand model: a 3d model was created and ready to be used, but needed the real input in order to adjust any mistakes may Accor while in live time action.

- The creation of the exe file was for testing purposes, to make sure the depth estimation model works fine after each changes without the need to make a run execution by using the cmd command (it is a translation file from js type to exe all provided in the deposit).