UNIK 4690 Computer Vision Project
Pose & Seg

Joseph Knutson & Jacob Alexander Hay

2018-05-27

Abstract

Introduction

Object Detection is a field within Computer Vision with the purpose of detecting objects of different classes. These classes of objects range from footballs to pedestrians. There are many methods for detecting the latter, but sadly, most of these methods care not to specify the position of of a person with high precision. Instead, some methods return rectangular patches where the person resides, while others return only a handful of key-points.

This project combines modern pose detection methods, like the one introduced by *Toshev et al* (2014) [1], with image processing and feature detection methods like Laplacian Blending and Edge Detection in order to not only detect people in images, but to better describe which pixels they inhabit. Once the human's pixels are extracted, we transfer them to other images via image blending. Like a green screen without a green screen.

The following chapters will discuss what tools we've used and roughly touch on the theory behind them. We'll clarify what we've borrowed from others. Which things we've implemented during our process, both that which works and that which doesn't. And finally, we point out what we wish to improve and further implement.

Theory

This chapter presents the methods we've used and a little bit about the theory behind them.

Pose Detection

Pose Detection is a computer vision technology which aims to detect key-points points, often on a person's body, in order to define that object's pose. Pose is in this case the translational information of the person in the image, its position in other words. In image 1 you can observe the algorithm[1] at work. The pose detection algorithm is the basis for our project, and as you can see, it finds key-points on both bodies and faces.

There are many ways to go about pose-detecting. The method we have borrowed from OpenPose is a modern method based on deep-learning, first presented by [1] in 2014. The algorithm is built upon a deep convolutional neural-network developed by Carnegie Mellon University with help of COCO and MPII datasets. This deep convolutional neural network has a generic DNN architecture, consisting of several layers – each being a linear transformation followed by a non-linear one. The first layer takes as input an image of predefined size. The further convolution of this image can be observed in image 2. The network's output is, as we saw, the key-points of the person's joints and face.

[1] This algorithm was written by OpenPose
Link to source: https://github.com/ildoonet/tf-pose-estimation

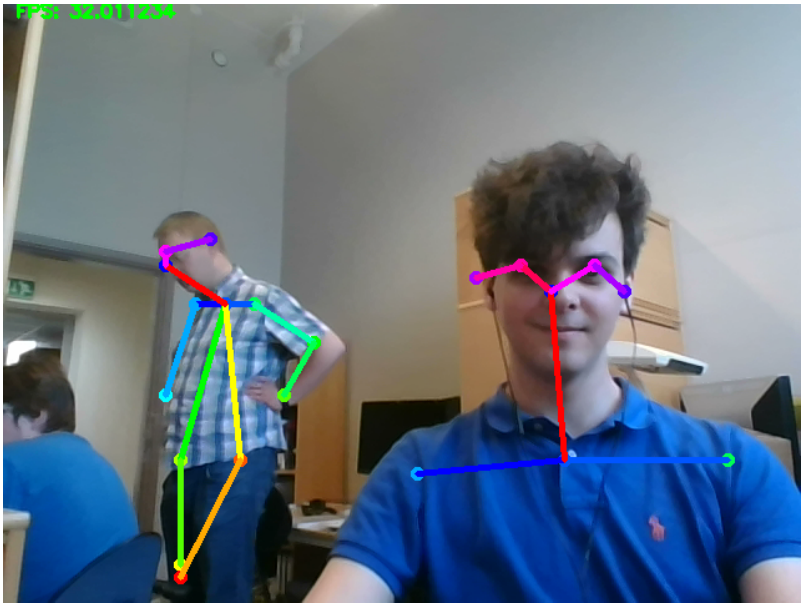

Figure 1: Pose detection algorithm which finds key body-points.

Figure 2: Diagram of the pose detection DNN, showing the convolutional behavior of this kind of network.

Figure 3: The Canny function detects edges in an image. It also includes threshold parameters which can reduce noisy details.

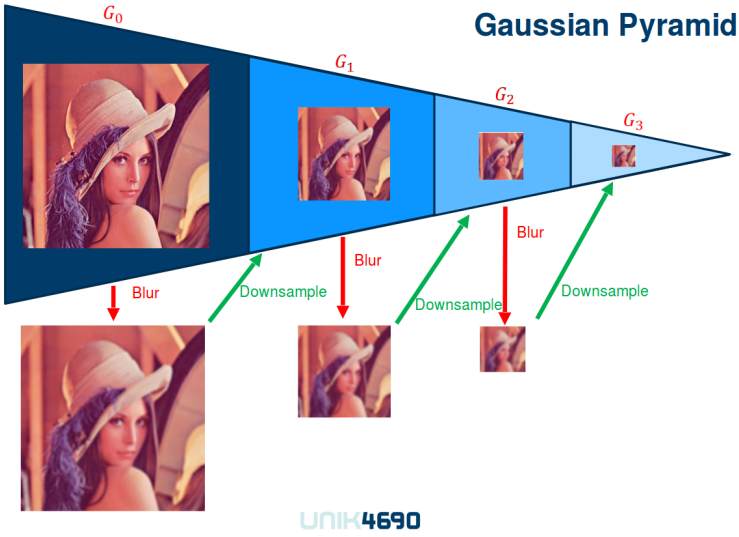

Figure 4: The Gaussian image pyramid.

Edge Detection

Edges can be both basic yet informative features in images. A basic way of finding edges is to look for large changes in pixel values along the x and y axes. This is equivalent to finding the gradient, or derivative of the image. Mathematically, we can express the edge intensity in a pixel as the difference between its neighbouring pixels' values:

$$I_e(x,y) = |I(x-1,y) - I(x+1,y)| + |I(x,y-1) - I(x+1,y+1)| \quad (1)$$

where I_e is the pixel value in the new edge image, I is the pixel value in the original image, and x and y are image coordinates. There are many more methods to finding edges that add more complexities, like using the Laplacian or adding threshold techniques. A famous edge detector function found in Open CV, called Canny works its magic in image 3

Image Pyramids and Laplace Blending

Laplace blending is an important part of our algorithm. The idea of Laplace blending is to first create a Gaussian and Laplacian pyramid for two different images, use a ramp-filter to define the overlapping behaviour they should have, and then collapse the overlapping image pyramids. Image 4, 5, and 6 are intuitive lecture slides from our course that explain the smooth image merging that Laplace blending offers. Not only do we use these pyramids for Laplace blending, we also use them for contouring.

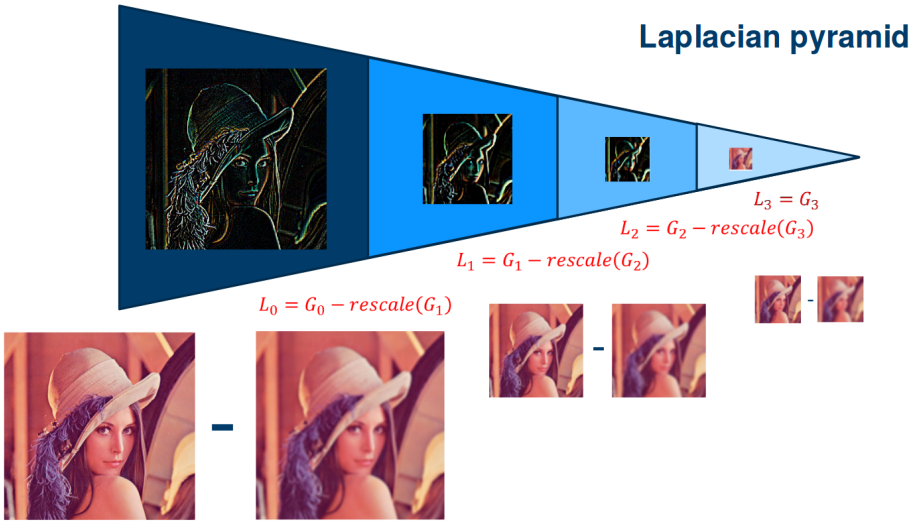

Figure 5: The Laplacian pyramid.

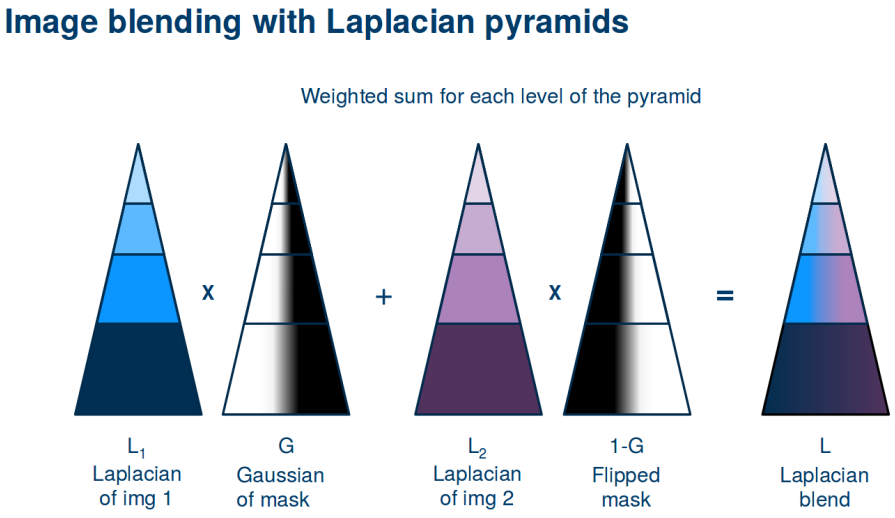

Figure 6: The conjoining of the images at multiple resolution levels with the help of a ramp/mask.

Tools and Programming Language

Apparatus

Our code can run live on webcam, so as long as you have a webcam, no matter how low-res it is, you can run our code. If you want a decent fps, e.g. over 20 fps, you'll need a dedicated graphics card.

Source that is not Our

As previously stated, the DNN that finds key-points along the body is not written by us. It is however an important input for the code we've written ourselves.

Python

Our project is written in Python 2.7. The libraries we're using are Numpy, Open CV2, Numba and Sci-Kit Learn. Numpy for matrix creation and matrix operations. Open CV is mainly used for edge detection, laplace blending, human modeling and morphological operations. Numba is for compiling Python code into C++ before it's run. This is mainly for image normalization. Sci-Kit had some useful image resize functions.

The Process.:

Along the long road towards a usable 2D segmentation we tried a few different approaches.

First out was to use a Canny Edge detection algorithm and try to filter out which edges were closest to any pose-points. After exhausting our paths that way, we determined that it would be better to operate on just a portion of the image. So we rewrote the algorithm to extract regions around points of interest, using element-wise multiplication with Filters.

These Filters were geometric shapes, and we first tried to simply extract the portions of the Canny image that were along the border of our picture. With the goal of using this then with the convexHull-function in openCV.

Our next attempt brought in a bit more interesting twists, as we started playing around with Laplacian Pyramids, and the concept of using the lower tiers of the Laplace Pyramid as our edge-detector. (The concept is not unlike extracting Detail-Spaces using Wavelets)

This turns out to maybe have some uses, but we are still then on a more or less plain segmentation-implementation.

Idea Process, 2D/3D, and Library Independence:

In the beginning we had an idea to try to go into segmentation and explore methods to possibly segment people in 3D. The concept there was to try to find the enveloping hull that closest fit the human being in 3D, using only a single camera and known geometric shapes that approximate the human form.

In order to do that we looked into using Pose-estimators such as OpenPose, and generating Spheres, Cubes, etc. to envelop the person.

On further feedback, we were guided towards doing more image segmentation, and less of the first concept to "scan" a human being into a 3D model. It was even suggested that we would/could try to map the images as we captured them onto a 3D model.

However, in order to do so, we would need to generate a 3D imaging tool which would allow us to both segment, capture, create a mosaic, and use something like ORBSLAM to keep track of the features and keypoints on this expansive surface that covers a human being.

Working on how to implement this, we decided therefore to first get the 2D elements correct. That is; To do segmentation using geometric shapes (Triangles, Circles, Polygons, etc.) and then, if time allowed us, to use these shapes as projected images of the 3D shapes of pyramids, Spheres, etc.

Along the way we would then also need to be able to extract the portions of the image that correlates to the overlap of these shapes and the camera feed. Which essentially turns it into a green-screen (without the green).

Ideally we would have a code that can take any coupled set of Canonical shapes, and a Pose-detection software, map the Shapes to the Pose, and from there be able to extract the information necessary.

The concept should hold for any kind of object, be it a simple or complicated one ideally.

References

[1] Alexander Toshev (Google), Christian Szegedy (Google).
DeepPose: Human Pose Estimation via Deep Neural Networks. 2014.
Conference on Computer Vision and Pattern Recognition, 2014.
`http://openaccess.thecvf.com/content_cvpr_2014/papers/Toshev_DeepPose_Human_Pose_2014_CVPR_paper.pdf`
Google, 1600 Amphitheatre Pkwy, Mountain View, CA 94043