UNIK 4690 Computer Vision Project
Pose & Seg
Source

Joseph Knutson & Jacob Alexander Hay

2018-05-27

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 a person in its entirety. 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

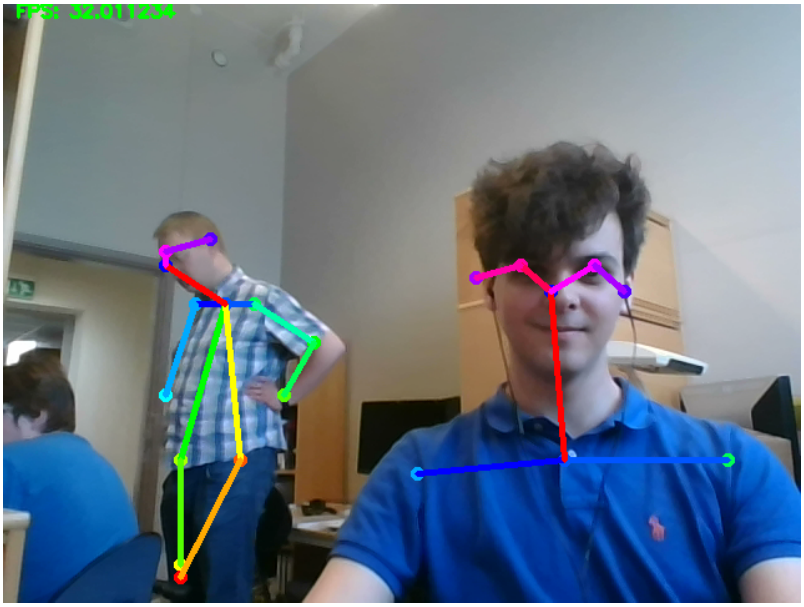

Figure 1: Pose detection algorithm which finds key body-points.

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.

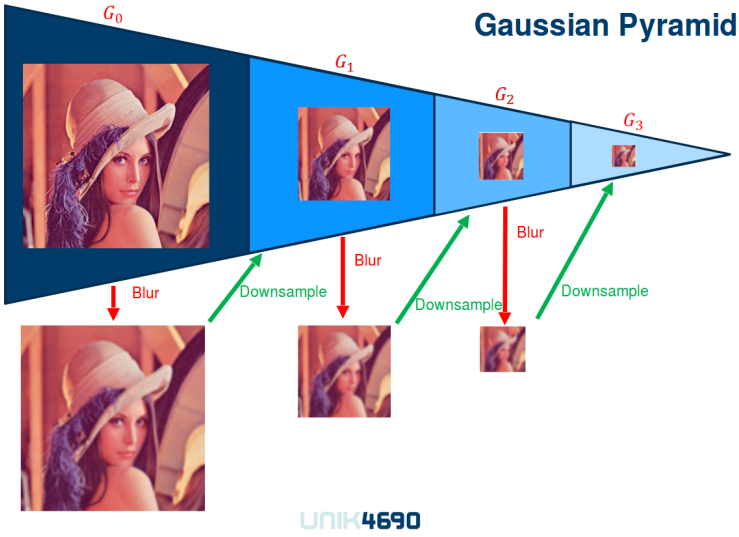

Figure 4: The Gaussian image pyramid.

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 function. Mathematically, we can express the edge intensity in a pixel as the difference between its neighbouring pixels' values:

$$I_e(x, y) = |I(x-1, y) - I(x+1, y)| + |I(x, y-1) - I(x+1, y+1)| \tag{1}$$

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 mask-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.

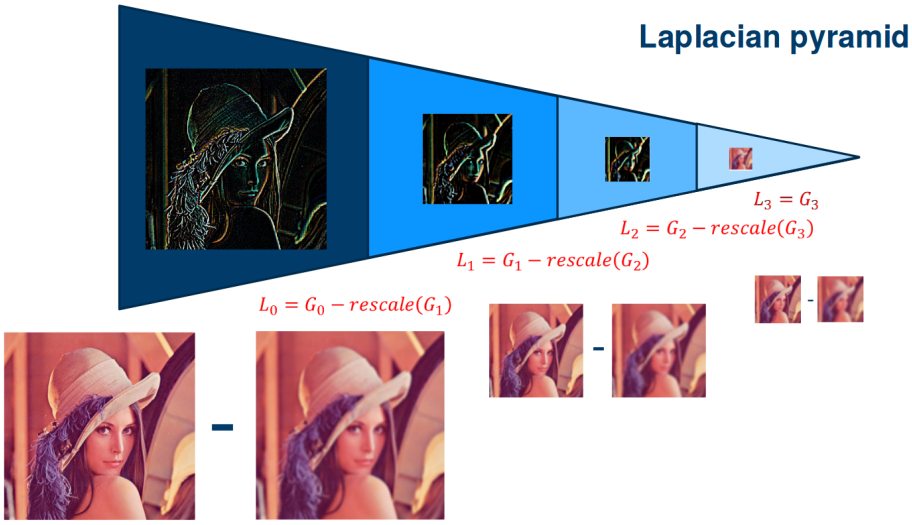

Figure 5: The Laplacian pyramid.

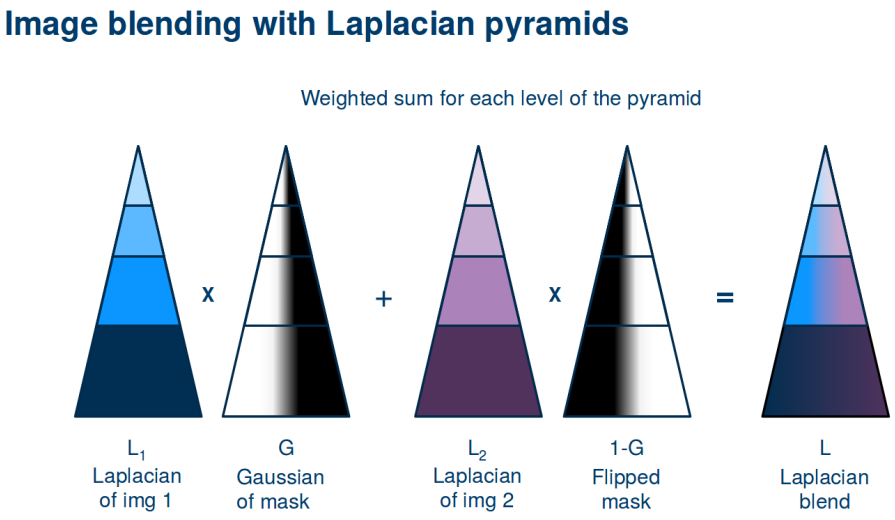

Figure 6: The conjoining of the images at multiple resolution levels with the help of a mask.

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.

Process

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

Contours

Our initial attempt at capturing the pixels of a human in the image consisted of using the Canny Edge detection algorithm. This proved too difficult as a near perfect, closed curve around the individual was impossible to get. Our plan was to fill the contour once it became a closed curve, like MS paint's "fill" tool. Image 7 shows the promising contours that we managed to produce.

Convex Contours

Since the contours would'nt close themselves, we got the idea that we should try closing them by treating them as polygon vertices. Using cv2.fillconvexPoly(), we made a last ditch effort to make contours work, resulting in image 8. Not too pretty. If our goal is to confine all the pixels of the person, this metod only covers half of them. Morphology didn't fix this.

Body Estimation and Draw Functions

We started making a model of the human body, a dummy, while working on the contouring (see image 9) and kept using it for Laplacian Blending too! The dummy was a rough estimation of where the person's body was positioned, this helped us get rid of outside noise. By simply using a Hamarand product (a sort of dot product between matrices, but with no reduction afterwards), all edges outside of our dummy was annuled.

We implement several different draw-functions for head, jawline, torso, arms and legs.

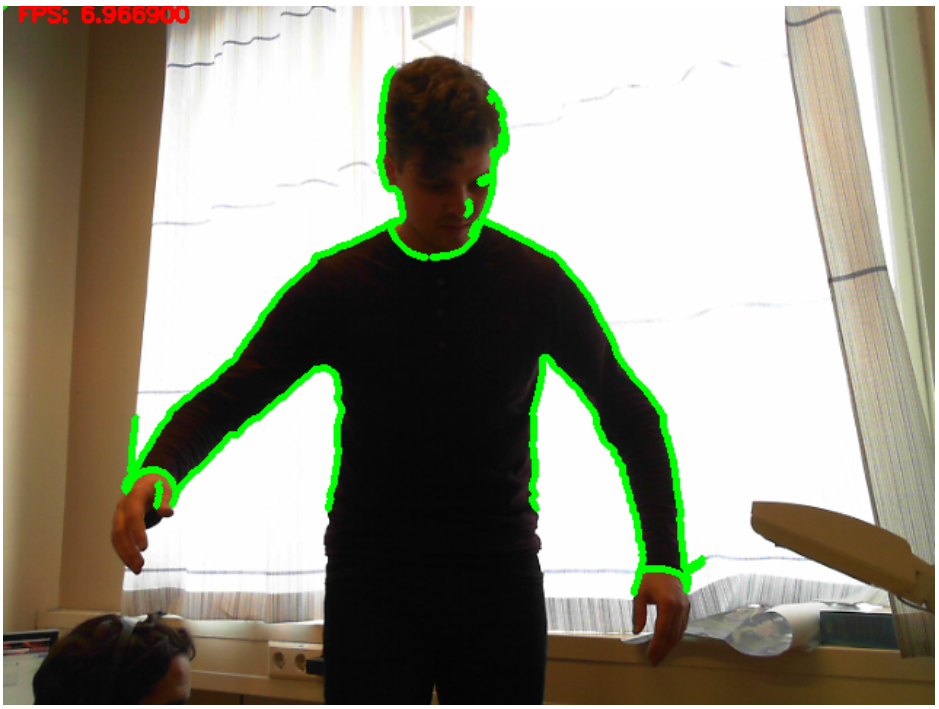

Figure 7: The use of edges to create a closed curve proved difficult.

Figure 8: Open CV's FillConvexPoly algorithm was not fitting, but produced an interesting result on our open contour lines.

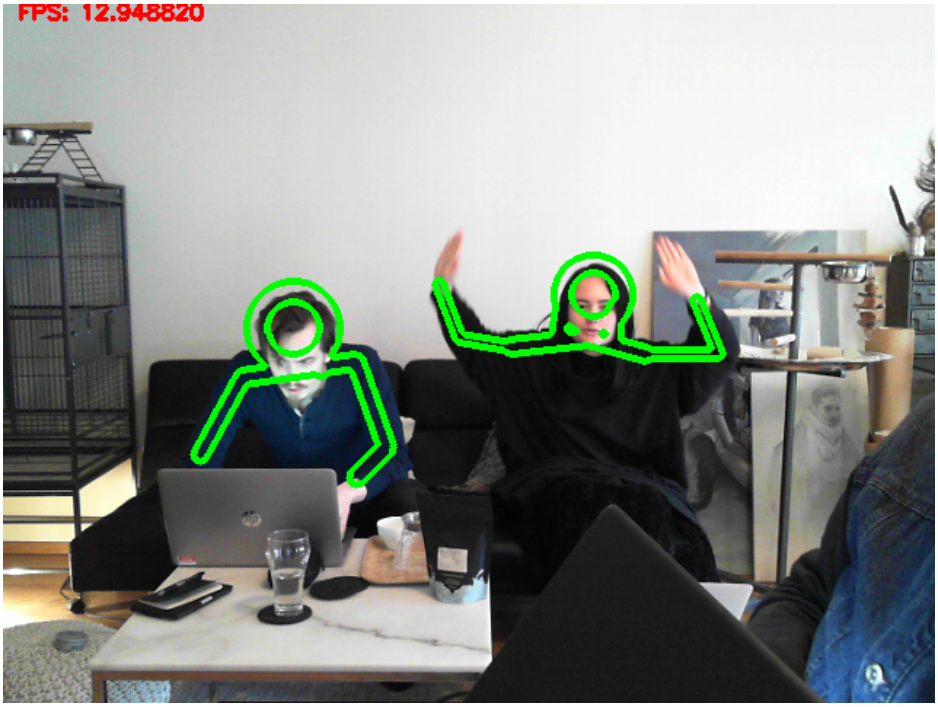

Figure 9: Geometric shapes based on the pose detection DNN. We started making a dummy to represent the human. This made it easy to find edges local to the person.

Head This function finds keypoints; eyes, nose, ears, and neck. Generating a semicircle or ellipse that covers the base structure of the back of the cranium.

Jaw This draws a set of vertices that form a square on bottom, and vertices at about 45 deg angle from the eyes to cover noses or chinbones.

Torso The torso is drawn in as two opposing semi-ellipses, which can be thought of as a slightly chubbied hour-glass.

Arms & legs These are in our model assumed to be approximately cylindrical. So we draw thick lines or squares that cover these, keeping with the theory that the 2D projection of a cylinder, or cone-section (non parallel cylinder sides), will form a square-like shape.

Hands Since our Pose-library doesn't feature hands, we decided to simply cover these by semicircles that somewhat cover the range of motion that these are likely to be found in.

We kept improving the dummy, because we knew we could use it along other image processing techniques. Creating more limbs from lines, ellipses and quadrilaterals, using simple pose transformations based on the key-points returned by the DNN. By making it dynamically grow and shrink based on pixel distances of the DNN's keypoints, it now mimics the human form quite well. As of now, the dummy looks like it does in image 10.

Laplace Blending

With a human dummy that roughly fits over the person in the image, Jacob proposed to simply laplace blend the person into another image, using the dummy as a mask. This gave some rather good results, but with a bit blurred out edges along the person (see figure 11 and 12).

Figure 10: More geometric shapes mimicing the person's body based on the pose detection DNN. Silly as it looks, a lot of works went into creating this guy (or girl).

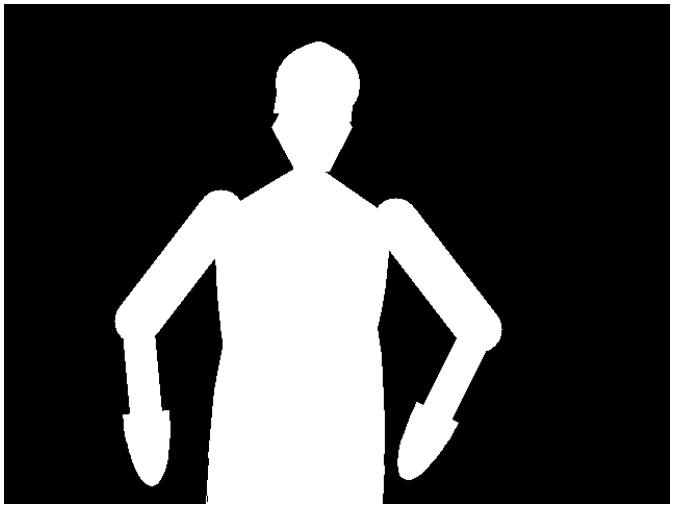

Figure 11: Using the dummy as mask for Laplace Blending

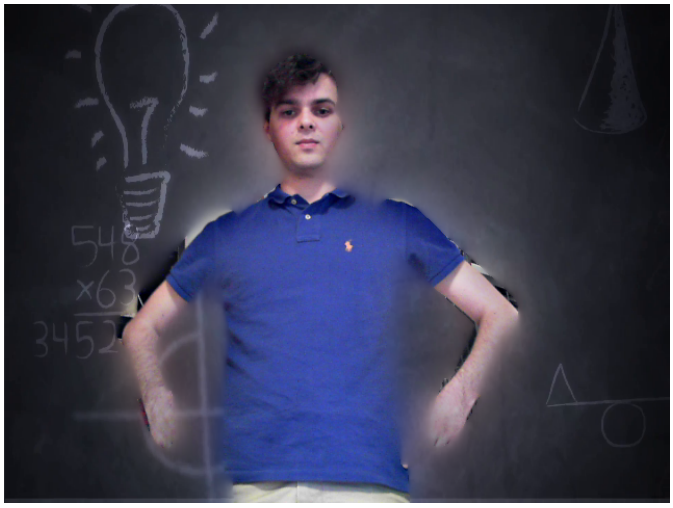

Figure 12: Laplace Blending of webcam stream and artificial background

Evaluation and Future Ambitions

Our Green Screen has the potential to become more precise by combining it with a successful method for finding a closed contour around the person. We were unable to implement a proper contour and ended instead up with a bulky Laplace Blending method for extracting the person's pixels instead.

We want to improve this code further, so that it can extract people from an image and used their pose and pixel values to recreate a model of them in 3D. As we were given feedback, we were guided towards doing more image segmentation, and less of our primary concept to "scan" a human being into 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.

Along the way we would then need to be able to extract the portions of the image that correlates to the people in the image. Which essentially turns the program 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 to create 3D models of people.

Bibliography

[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