Human Pose Annotation Final Report

Team member: Runsheng Xu

Peiyuan Zhang

Faculty advisor: Ying Wu

Northwestern University

Northwestern University

Contents

1.Introduction

Human motion capturing has recently become a very hot and challenge topic. A key issue to estimate the articulated human motion is how to localize the human body part so that the computer or robot will recognize different segments. An extensive research on this topic reveals that there are many recent methodologies addressing this problem[1,2,3,4].

Most of these methods applied machine learning like in [5], which used Random Forest to train depth map and then combined with spatio-temporal graph cut to segment human body parts. However, the main issue in this kind of method is that it usually acquires a great deal of time as well as high-performance CPU, like the one mentioned before in [5], they spent 78 hours using 5 computers together to get the results.

So in our project, we proposed two methods to segment human body parts automatically given a sequence of RGB images(or video), corresponding depth maps and skeleton data, which just requires less than 1 hour for more than 500 frames.

The first method used grabcut [6] based on RGB images and skeleton data to gain a rough cut result and then applied superpixel segmentation algorithm with its corresponding depth map to improve the raw result. The use of grabcut is mainly to identify foreground and background to reduce computation energy and superpixel is

for coloring the body parts which have not been covered yet.

The second method applied superpixel segmentation given skeleton data directly on depth images, and then map to their corresponding RGB images to get the final result. There is a misalign problem in Kinect between RGB and depth images and the solution to it will also be included in this report.

The rest of this report is organized as follows: Section 2 presents the novel segmentation framework including superpixel, Boykov's graph cut(the base of grab cut) and grab cut algorithm. Section 3 the process or experiments as well as the results. Section 4 gives my own contribution to this project and section 5 concludes this paper.

2. Method

2.1 Superpixel Segmentation

Superixel was first proposed by Kevin Smith. It provides a convenient primitive from which to compute local image features. They capture redundancy in the image and greatly reduce the complexity of subsequent image processing tasks. They have proved increasingly useful for applications such as depth estimation, image segmentation, skeletonization, body model estimation, and object localization. For superpixels to be useful they must be fast, easy to use, and produce high quality segmentations. We use Simple linear iterative clustering (SLIC) to produce superpixels for us. SLIC performs a local clustering of pixels in the 5-D space defined by the L, a, b values of the CIELAB

color space and the x, y pixel coordinates. This algorithm could not only produces high quality, compact, nearly uniform superpixels more efficiently, but also produces segmentations of similar or better quality as measured by standard boundary recall and under-segmentation error measures.

The SLIC approach generates superpixels by clustering pixels based on their color similarity and proximity in the image plane. This is done in the five-dimensional [labxy] space, where [lab] is the pixel color vector in CIELAB color space, which is widely considered as perceptually uniform for small color distances, and x-y is the pixel position. While the maximum possible distance between two colors in the CIELAB space (assuming RGB input images) is limited, the spatial distance in the x-y plane depends on the image size. In order to cluster pixels in this 5D space, we introduce a new distance measure that considers superpixel size. Using it, we enforce color similarity as well as pixel proximity in this 5D space such that the expected cluster sizes and their spatial extent are approximately equal.

This algorithm takes as input a desired number of approximately equally-sized superpixels K. For an image with N pixels, the approximate size of each superpixel is therefore N / K pixels. For roughly equally sized superpixels there would be a superpixel center at every grid interval $S = \sqrt{N / K}$.

At the onset of our algorithm, we choose K superpixel cluster centers $C_k =$

$[l_k, a_k, b_k, x_k, y_k]^{\mathrm{T}}$ with $\mathrm{k} = [1, \mathrm{k}]$ at regular grid intervals S. Since the spatial extent of any superpixel is approximately S^2 (the approximate area of a superpixel), we can safely assume that pixels that are associated with this cluster center lie within a $2S \times 2S$ area around the superpixel center on the xy plane. This becomes the search area for the pixels nearest to each cluster center. we use a distance measure Ds defined as follows:

$$d_{lab} = \sqrt{(l_k - l_i)^2 + (a_k - a_i)^2 + (b_k - b_i)^2}$$

$$d_{xy} = \sqrt{(x_k - x_i)^2 + (y_k - y_i)^2}$$

$$D_s = d_{lab} + \frac{m}{S} d_{xy}$$

where D_s is the sum of the lab distance and the xy plane distance normalized by the grid interval S. A variable m is introduced in D_s allowing us to control the compactness of a superpixel. The greater the value of m, the more spatial proximity is emphasized and the more compact the cluster. This value can be in the range [1, 20]. This roughly matches the empirical maximum perceptually meaningful CIELAB distance and offers a good balance between color similarity and spatial proximity.

Then We begin by sampling K regularly spaced cluster centers and moving them to seed locations corresponding to the lowest gradient position in a 3×3 neighborhood. This is done to avoid placing them at an edge and to reduce the chances of choosing a noisy pixel. Image gradients are computed as:

$$\mathrm{G}(x, y) = ||I(x+1, y) - I(x-1, y)||^2 + ||I(x, y+1) - I(x, y-1)||^2$$

Where $I(x, y)$ is the lab vector corresponding to the pixel at position (x, y), and $||.||$ is the L_2 norm. This takes into account both color and intensity information. Each

pixel in the image is associated with the nearest cluster center whose search area overlaps this pixel. After all the pixels are associated with the nearest cluster center, a new center is computed as the average labxy vector of all the pixels belonging to the cluster. We then iteratively repeat the process of associating pixels with the nearest cluster center and recomputing the cluster center until convergence. At the end of this process, a few stray labels may remain, that is, a few pixels in the vicinity of a larger segment having the same label but not connected to it. While it is rare, this may arise despite the spatial proximity measure since our clustering does not explicitly enforce connectivity. Nevertheless, we enforce connectivity in the last step of our algorithm by relabeling disjoint segments with the labels of the largest neighboring cluster. This step is $O(N)$ complex and takes less than 10% of the total time required for segmenting an image.

2.2 Boykov's Graph Cut

The grabcut algorithm was first proposed by Carsten in his article *GrabCut – Interactive Foreground Extraction using Iterated Graph Cuts* [6] . His theory is more like an improved or revised version of Boykov's graph cut in the article *Interactive Graph Cuts for Optimal Boundary & Region Segmentation of Objects in N-D Images*. So before we introduce grabcut, we have to describe Boykov's graph cut first.

First, we describe the basic terminology that pertains to in the context of the

segmentation method. An undirected graph G=<V, E> is defined as a set of nodes(vertices V) and a set of undirected edges(E) that connects these nodes[7]. An example of a graph is shown in Figure 1.

For image graph cuts, the nodes in figure 1 represents pixels and their corresponding seeds O or B determines they are foreground and background. T, S are called terminals which can be divided into background and foreground terminal. Each terminal has a link with every pixel(nodes) and has different energy for every link. Besides that, each nodes also has a link themselves whose energy is also varied. The aim of graph cuts is to find a cut that can minimize the cost of edges to segment the images.

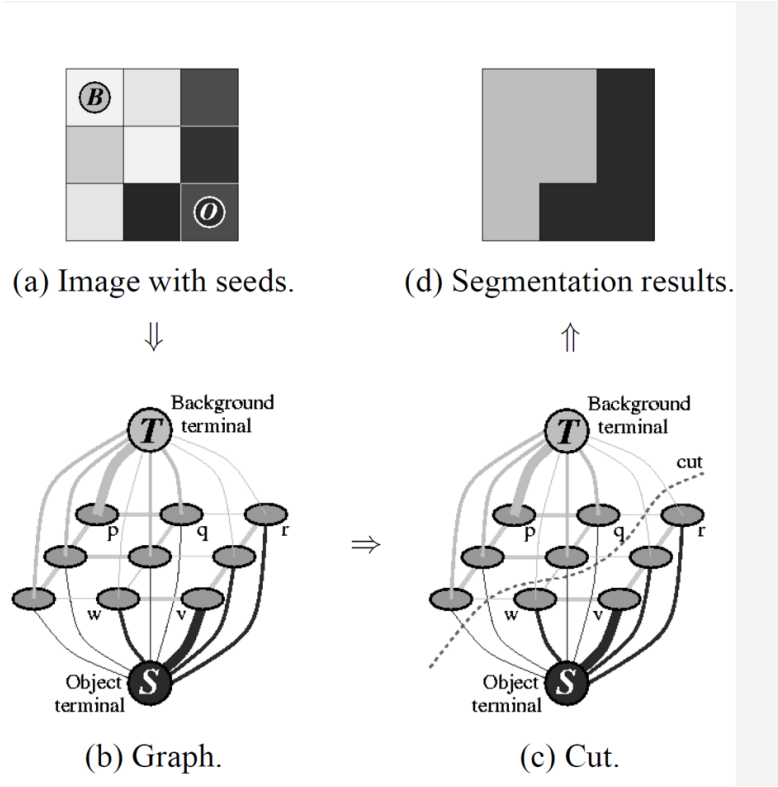

Figure 1 A simple 2D segmentation example for a 3*3 image

In [7], Boykov first proposed a general energy function to model the graph cut issue, which is shown below.

$$E(A) = \lambda \cdot R(A) + B(A) \quad (1)$$

where

$$R(A) = \sum_{p \in \mathcal{P}} R_p(A_p) \quad (2)$$

$$B(A) = \sum_{\{p,q\} \in \mathcal{N}} B_{\{p,q\}} \cdot \delta(A_p, A_q) \quad (3)$$

and

$$\delta(A_p, A_q) = \begin{cases} 1 & \text{if } A_p \neq A_q \\ 0 & \text{otherwise.} \end{cases}$$

R and B represents regional energy which exists between nodes and terminals and boundary energy which exists in the edges of neighbor nodes. From the formula (3) we can see, when two pixels are more similar, the more energy stores in their edges. And then, Boykov proposed a hard constrain which is set by users to initialize the label configuration, it could be expressed as follows:

$$\forall p \in \mathcal{O}, \quad A_p = \text{“obj”} \quad (4)$$

$$\forall p \in \mathcal{B}, \quad A_p = \text{“bkg”}. \quad (5)$$

where p represents pixels in image and A is a label vector for every pixel which can be divided in to background and foreground label. As a result, the weigh cost(energy) of edges is concluded as below:

edge	weight (cost)	for
$\{p, q\}$	$B_{\{p,q\}}$	$\{p, q\} \in \mathcal{N}$
$\{p, S\}$	$\lambda \cdot R_p(\text{“bkg”})$	$p \in \mathcal{P},\ p \notin \mathcal{O} \cup \mathcal{B}$
	K	$p \in \mathcal{O}$
	0	$p \in \mathcal{B}$
$\{p, T\}$	$\lambda \cdot R_p(\text{“obj”})$	$p \in \mathcal{P},\ p \notin \mathcal{O} \cup \mathcal{B}$
	0	$p \in \mathcal{O}$
	K	$p \in \mathcal{B}$

where

$$K = 1 + \max_{p \in \mathcal{P}} \sum_{q:\ \{p,q\} \in \mathcal{N}} B_{\{p,q\}}.$$

After everything is modeled, Boykov applied maximum flow(minimum cut) algorithm [7] to cut the graph based on the weight cost list and get the segmentation result.

2.3 GrabCut Algorithm

As mentioned before, grabcut algorithm is invented based on the Boykov's graph cut algorithm, but they are very different. Graph cut applies a one-time shot on grey scale image while grabcut uses iterative graph cut on RGB images which has three channels.

As it is impractical to construct adequate colour space histograms, Casten followed a

practice that is already used for soft segmentation [8] and use GMMs(Gaussian Mixture Model). Each GMM is taken to be a full- covariance Gaussian mixture with K components (K=5 in usual) [6]. Compared to graph cut, an additional vector k={k1,k2….kn} is added as Gaussian component to every pixel. As a result, the new energy function Gibbs energy becomes:

$$\mathbf{E}(\underline{\alpha}, \mathbf{k}, \underline{\theta}, \mathbf{z}) = U(\underline{\alpha}, \mathbf{k}, \underline{\theta}, \mathbf{z}) + V(\underline{\alpha}, \mathbf{z}),$$

depending also on the GMM component variables k, U is the unary potential and

$$U(\underline{\alpha}, \mathbf{k}, \underline{\theta}, \mathbf{z}) = \sum_n D(\alpha_n, k_n, \underline{\theta}, z_n),$$

Where D represents the Gaussian probability distribution:

$$D(\alpha_n, k_n, \underline{\theta}, z_n) = -\log \pi(\alpha_n, k_n) + \frac{1}{2} \log \det \Sigma(\alpha_n, k_n) + \frac{1}{2}[z_n - \mu(\alpha_n, k_n)]^\top \Sigma(\alpha_n, k_n)^{-1} [z_n - \mu(\alpha_n, k_n)].$$

Therefore, the parameters of the model now are:

$$\underline{\theta} = \{\pi(\alpha, k), \mu(\alpha, k), \Sigma(\alpha, k),\ \alpha = 0, 1,\ k = 1 \ldots K\},$$

And the smoothness terms V is basically unchanged compared to graph cuts except that the contrast term is computed using Euclidean distance in color space:

$$V(\underline{\alpha}, \mathbf{z}) = \gamma \sum_{(m,n) \in \mathbf{C}} [\alpha_n \neq \alpha_m] \exp -\beta \|z_m - z_n\|^2.$$

After the model is well established, an iterative energy minimization is then applied on the segmentation.

Different from graph cut, the hard constrain set by user now are not brush but a rectangle area in which pixels represent foreground and pixels out of the areas are

labeled as background for initialization. The following chart indicates the whole process of the iterative minimization.

Initialisation

- User initialises trimap T by supplying only T_B. The foreground is set to $T_F = \emptyset$; $T_U = \overline{T}_B$, complement of the background.
- Initialise $\alpha_n = 0$ for $n \in T_B$ and $\alpha_n = 1$ for $n \in T_U$.
- Background and foreground GMMs initialised from sets $\alpha_n = 0$ and $\alpha_n = 1$ respectively.

Iterative minimisation

1. *Assign GMM components to pixels:* for each n in T_U,
$$k_n := \arg\min_{k_n} D_n(\alpha_n, k_n, \theta, z_n).$$
2. *Learn GMM parameters* from data $\mathbf{z}$:
$$\underline{\theta} := \arg\min_{\underline{\theta}} U(\underline{\alpha}, \mathbf{k}, \underline{\theta}, \mathbf{z})$$
3. *Estimate segmentation:* use min cut to solve:
$$\min_{\{\alpha_n:\, n \in T_U\}} \min_{\mathbf{k}} \mathbf{E}(\underline{\alpha}, \mathbf{k}, \underline{\theta}, \mathbf{z}).$$
4. Repeat from step 1, until convergence.
5. Apply border matting (section 4).

User editing

- *Edit:* fix some pixels either to $\alpha_n = 0$ (background brush) or $\alpha_n = 1$ (foreground brush); update trimap T accordingly. Perform step 3 above, just once.
- *Refine operation:* [optional] perform entire iterative minimisation algorithm.

3. Experiments and Results

3.1 Gain data using Kinect

We want to obtain RGB, depth images, skeleton information and so on. Kinect v2 could satisfy these requirements. Kinect v2 is a motion sensing input device by Microsoft for the Windows PCs. The device is a horizontal bar connected to a small base with a motorized pivot and is designed to be positioned lengthwise above or below the video display. Kinect v2 has RGB camera, depth sensor(IR), multi-array microphone and motor to adjust camera angle. It also provides functions such as facial recognition, near mode and skeletal tracking

We install Kinect SDK in our laptop, which enables us to create applications that support gesture and voice recognition, using Kinect sensor technology on computers. Then we connect Kinect v2 to our laptop using Kinect adapter. In our laptop, we use matlab2016a to program.

First , we setup the Kinect V2 for color and depth acquisition. we detect the Kinect V2 devices. The Kinect V2 for windows camera has two separate sensors, the color sensor and the depth sensor. Each sensor has a DeviceID in the Image Acquisition Toolbox. We also create color and depth Kinect videoinput objects using “colorVid = videoinput(‘kinect’, 1)”, “depthVid = videoinput(‘kinect’, 2)”. We also set “EnableBodyTracking” to on, so that the depth sensor will return body tracking

metadata along with the depth frame. The metadata contains skeleton information which is important for us. Then we set “framesPerTrig” equals to 100 and make both “colorVid.FramesPerTrigger” and ”depthVid.FramesPerTrigger” equal to “framesPerTrig” so that we can get 100 color and depth frames. At last we program “start([depthVid colorVid])” to begins acquisition.

Then we access image and skeleton data from the color and depth device objects. We type “[colorImg] = getdata(colorVid)” and “[depthmap, ~, metadata] = getdata(depthVid)” so that make related data appear on the workspace.

Now we get 100 frames color and depth images and metadata(skeleton information) for these images. However, depth images are not aligned with color images because it acquired by different sensor. Therefore, we need to calibrate the depth and color camera. We use a preliminary semi-automatic way to calibrate the Kinect depth sensor and the RGB output to enable a mapping between them. It is basically a standard stereo calibration technique, the main difficulty comes from the depth image that cannot detect patterns on a flat surface. Thus, the pattern has to be created using depth difference. We use a rectangular peace of carton cut around a chessboard printed on an A3 sheet of paper.

First, we make the calibration of the color camera intrinsics using standard chessboard recognition.

Second, we make the calibration of the depth camera intrinsics. We extract the corners of the chessboard on the depth image and storing them. This could be done automatically.

After these two step, we transform raw depth values into meters. Raw depth values are integer between 0 and 2047. They can be transformed into depth in the meters using related parameters.

Then, we select the corners of the chessboard on the color images, which could be easy to extract automatically using the chessboard recognition. We perform standard stereo calibration via this process.

Based on these preparation, we map depth pixels with color pixels. The first step is to undistort RGB and depth images using the estimated distortion coefficients. Then, using the depth camera intrinsics, each pixel (x_d, y_d) of the depth camera can be projected to metric 3D space. We can then reproject each 3D point on the color image and get its color. After these steps, we accomplish the calibration between color and depth camera.

Now we get 100 frames images. We can use matlab to combine these images into a video.

3.2 Two methods to process data

As section 1 mentioned, after get the RGB sequence data, depth images and skeleton data, we have applied two different methods to process the data for segmentation.

In the first method, we first give a rough cut using rectangle based on skeleton data, as figure 2 shows, note the ratio of width and height of the rectangle relies on different body parts and gained from experimental result.

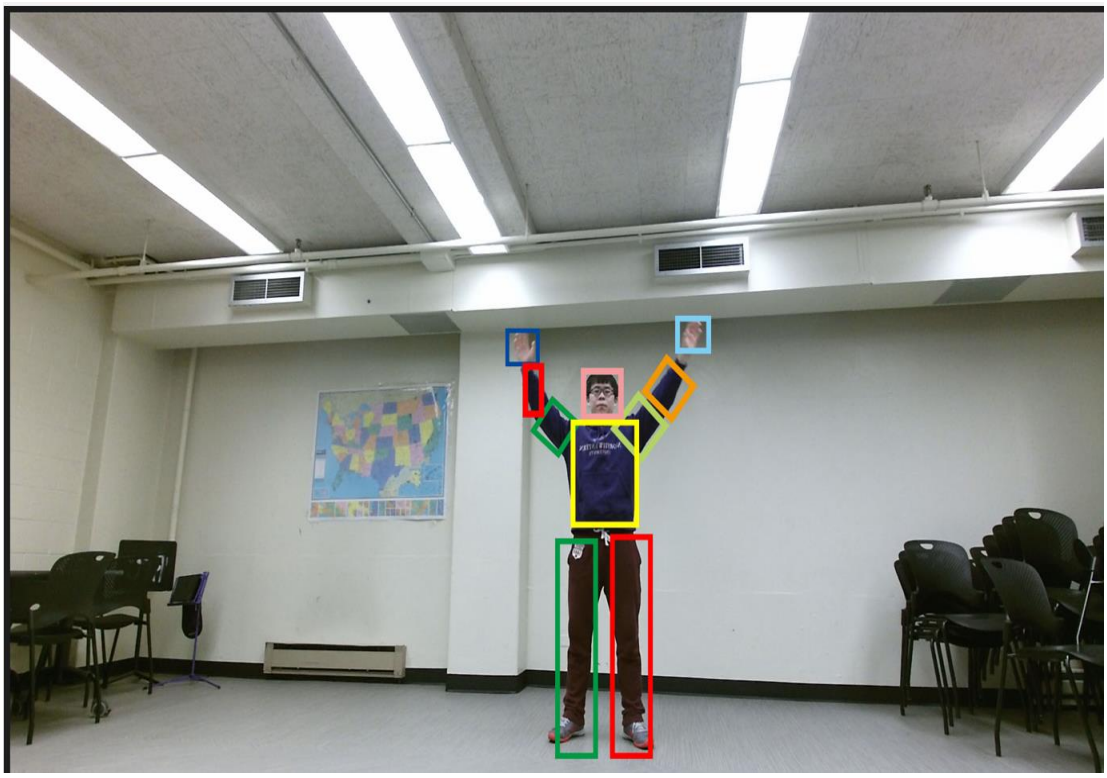

Figure 2 Rough cut on RGB image

Next, we applied grab cut on these rectangles and this algorithm can automatically segment a specific body part like arm, leg ,head from background or other body parts. Then the segmented body parts are painted with different color for labeling as figure 3 shows.

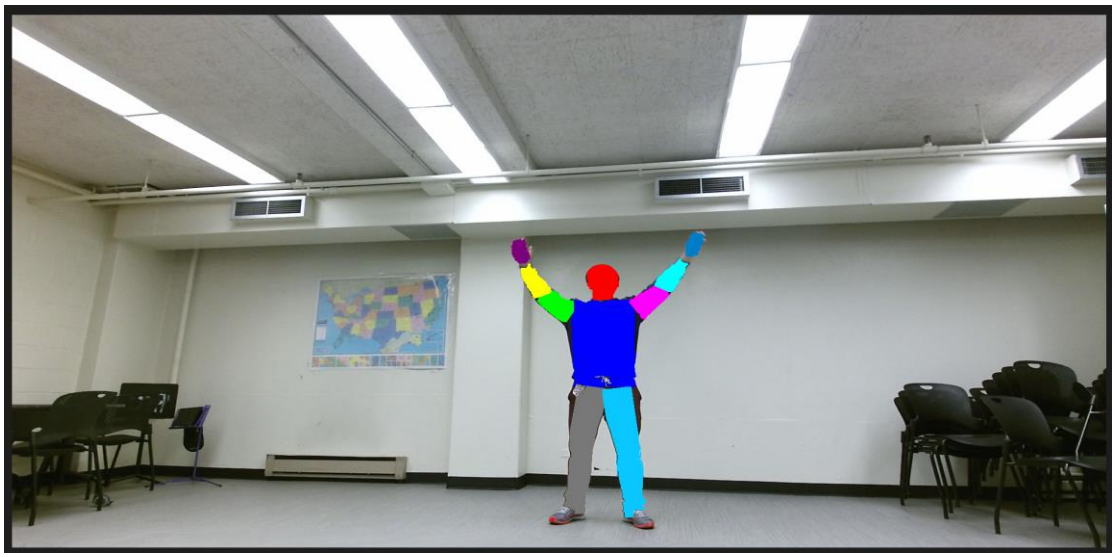

Figure 3 Primary Result

As we can see from the primary result, it's not perfect. Some parts in the body are not colored and some parts in background are colored by mistake. To improve the result, we then applied superpixel combined with depth map as figure 4 and fig 5.

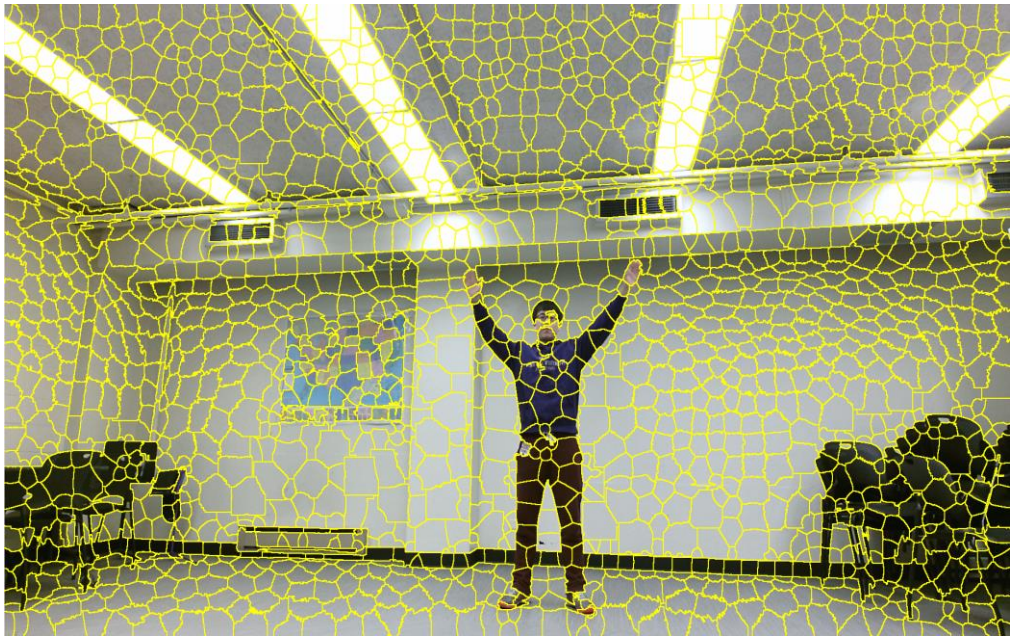

Figure 4 Superpixel Segmentation on RGB image

Figure 5 Depth Map as Reference

Superpixel can segment the human body into thousands of pieces, and in our algorithm , if some pixels in one piece are labeled , then the rest of this piece should be labeled too. To avoid labeling background in the same piece, we use depth map as reference to solve this problem. The revised result is shown in figure 6.

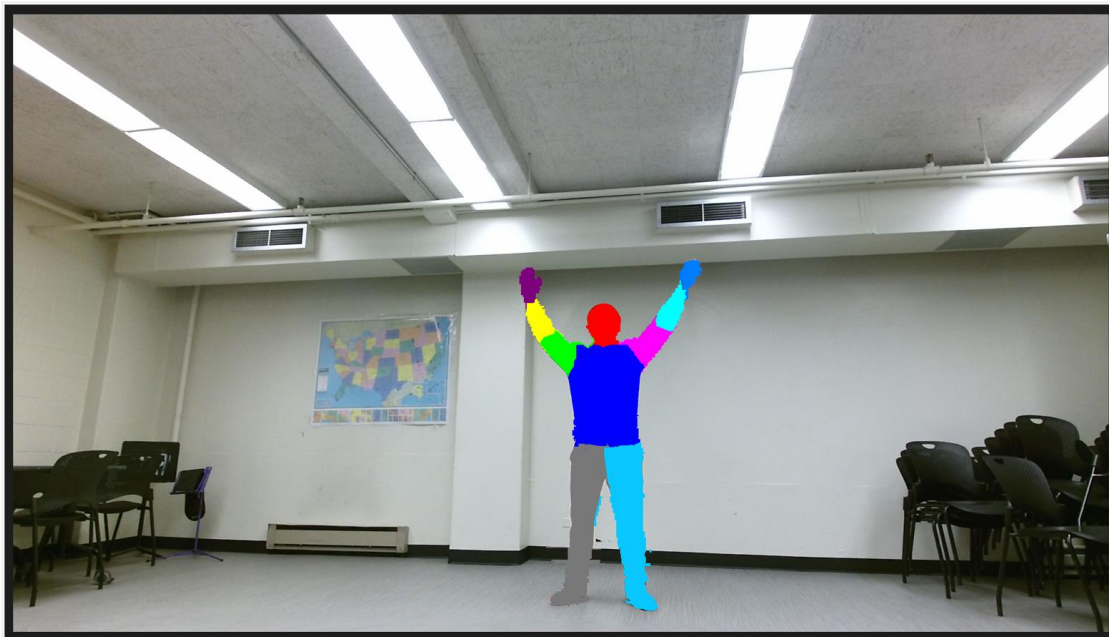

Figure 6 Final Result

The final result looks pretty good, however, when the background's color histogram is similar with human's , this algorithm won't work very well. To solve this issue, here we addressed the second method—using superpixel segmentation on depth map directly.

The first step is the same, a rough cut based on skeleton data is applied to the depth map. And then , the depth image will be segmented using superpixel, like figure 7 and 8 shows.

Figure 7 Depth image

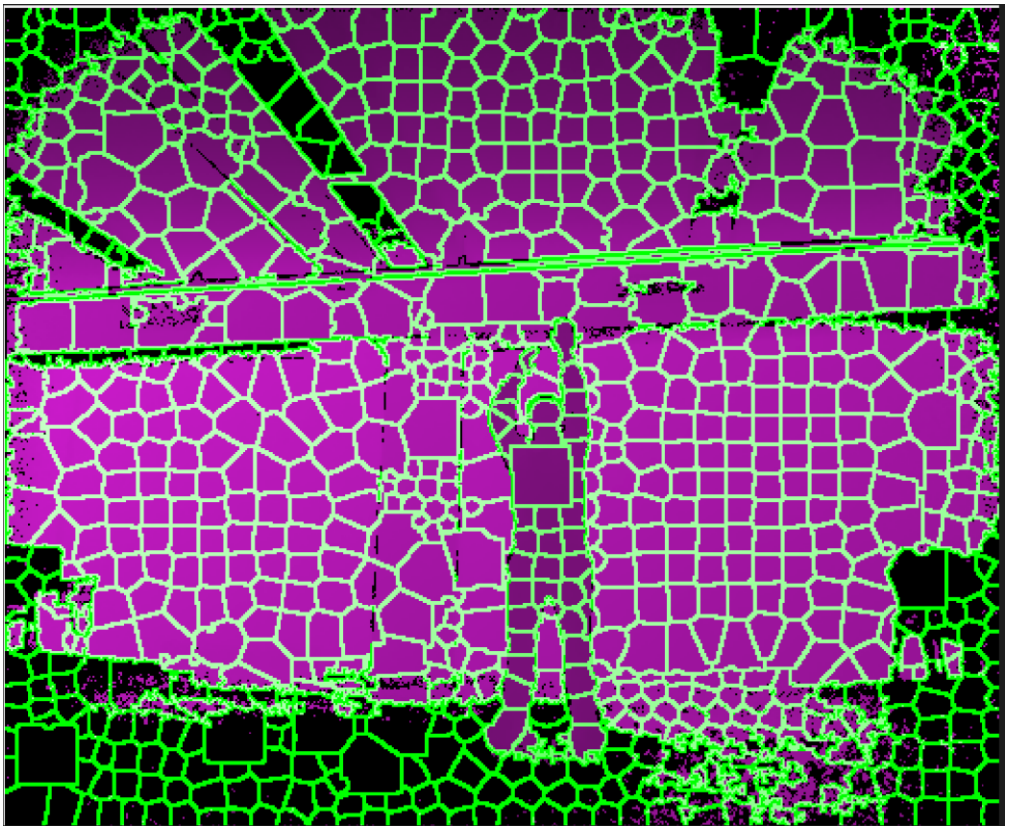

Figure 8 Superpixel Segmentation on Depth Map

Next, the overlap regions of superpixel pieces and rough cut rectangles will be labeled of different color. However, there may be one situation that the arm has an overlap region with the body or head and that area will be labeled twice as a mistake. In this case, we applied color histograms and KL divergence distribution to solve this problem. For example, if one piece is labeled both arm and body, then we will first find other pieces that are labeled body only, gain their average color histograms and do the same thing with arm labeled pieces. Following is comparing the two color histograms of arm and body with the piece needed to be labeled using KL divergence then find the one is more similar with the target piece. Finally, we map the depth image to color image to get the final result, which is shown in figure 9 and 10.

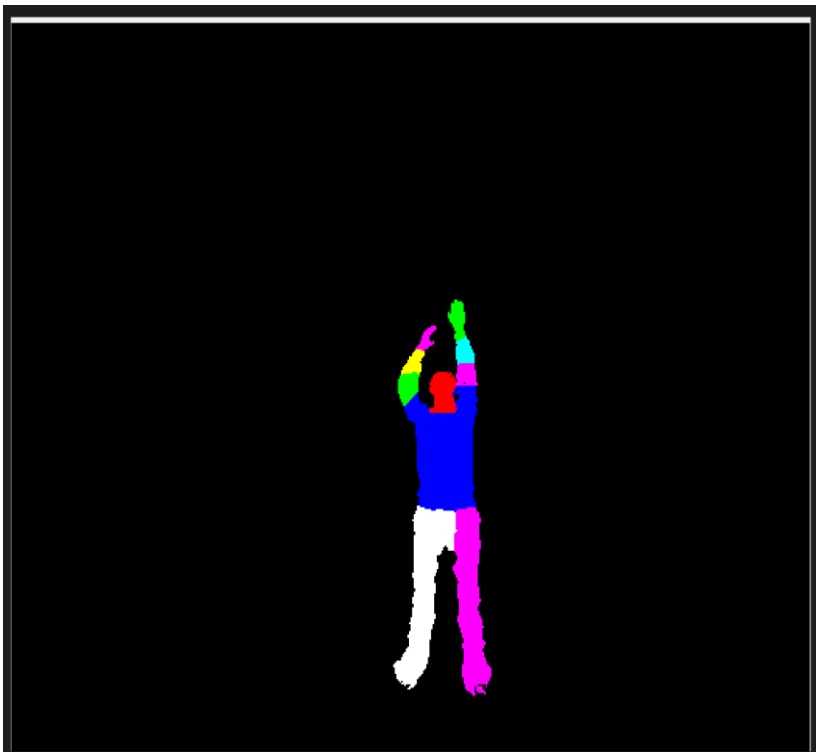

Figure 9 Segmentation Result on depth map

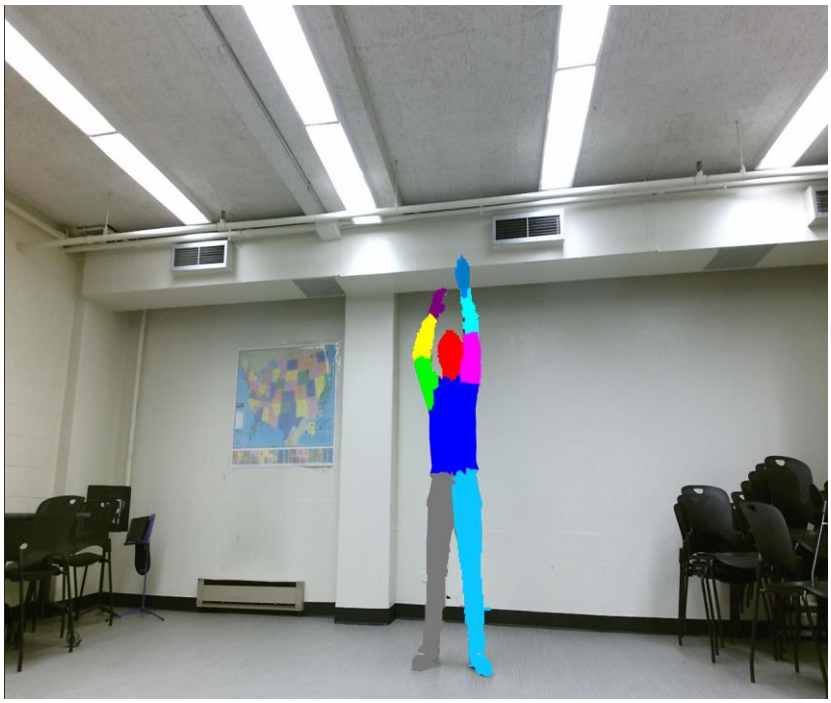

Figure 10 Final Result

3.3 Manual Labeling

Since the automatic segmentation result won't be always perfect, we developed a tool allowing users manual relabel the images that they are not satisfied. This tool basically applied superpixel for area label as well as pixel level label. Figure 11 shows how the tool looks like.

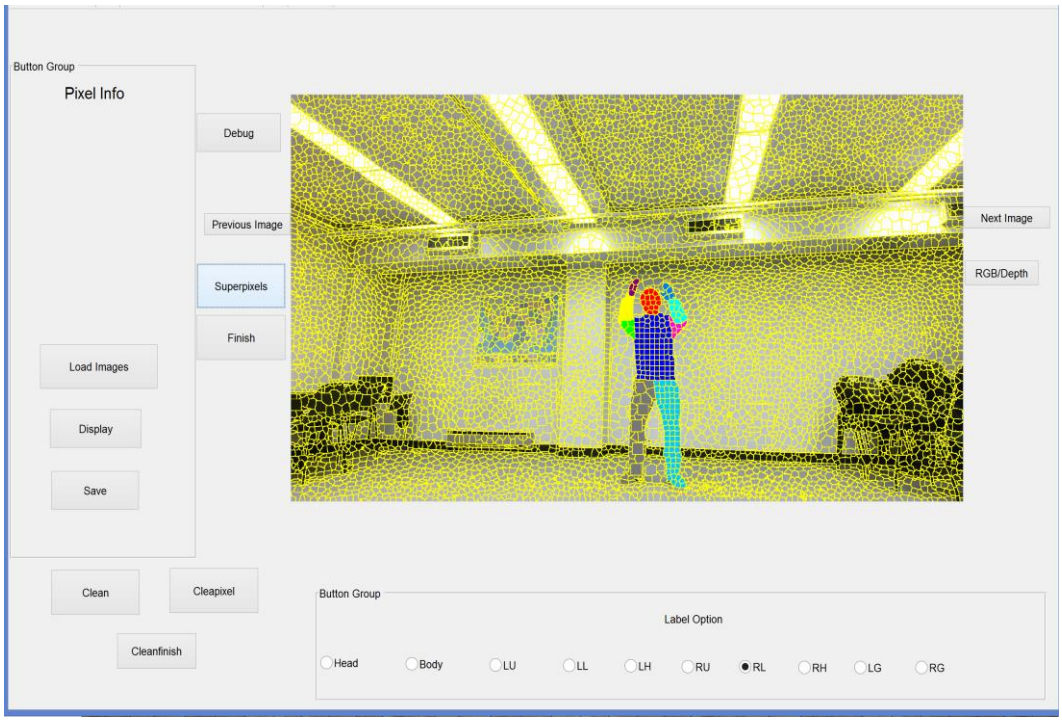

Figure 11 Manual Label Tool

4. My Contribution

In this team project, my main contribution is designing the two different methods for human body segmentation, achieving them in matlab code and then developing the manual labeling tool.

My teammate Peiyuan focused on collecting data using Kinect, aligning depth and RGB images and manually labeled the automatic segmented RGB images for correction.

5. Summary

In this project, our team has extensively studied both the theoretical knowledge of image segmentation, as well as the fundamental skills of applying them on practical applications. . Our team has proposed two different methods which are both quick, accurate and efficient compared to other conventional methods to solve the human body segmentation problem. From the result we can see, our algorithm works very well on human motion without too serious overlap regions. In the future, we will propose a new method to use grabcut on RGB-D data.

References

[1] N. Dalal and B Triggers. Histogram of oriented gradients for human detection. Volume 2,page 886-893, 2005.

[2] N. Dalal , B Triggers and C. Schmid. Human detection using oriented histograms of flow and appearence. *European Conference on Computer Vision*, pages 7-13. 2006

[3] G. Cheung, T. Kanade, J,-Y. Bouguet, and M. H. A. Real time system for human motions. 2:714 -720,2000. CVPR

[4] W. Schwartz, A. Kembhavi and D Harwood. Human detection using partial least squares. Pages 24-31,2009. International Conference on Computer Vision.

[5] Boykov, Y., and Jolly, M. -P. 2001. Interactive graph cuts for optimal boundary and region segmentation of objects in N-D images. International Conference on Computer Vision

[6] L. Ford and D .Fulkon. *Flows in Network. Princeton University Press*, 1962

[7] Casten Rother and Andrew Blake. “GrabCut”—Interactive Foreground Extraction using Iterated Graph Cuts. In *Proc. European Conf. Computer Vision.*