Three-Dimensional Motion Tracking using Clustering

Ryan Shiroma and Andrew Zastovnik

Dec 11, 2015

Abstract

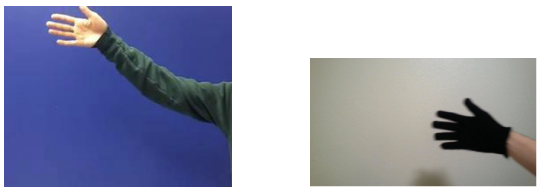
Tracking the position of an object in three dimensional space is a fascinating problem with many real world applications. With applications ranging anywhere from tracking the positions of people on security cameras to finding the distance and relative velocities of stars. In this paper we will be discussing a method of three dimensional motion tracking which uses clustering. The two clustering algorithms we will be using are K-means and spectral clustering. We will then discuss how this information is used in position tracking. Finally, we will discuss our results and ideas for future work.

1 Introduction

Our motivation for this project idea was to identify some of the advantages and disadvantages of clustering techniques to real world applications. Theoretically we assumed it was possible to generate 3D projections from image segmentation results but whether we'd get practical results was unknown. While other, faster, non-clustering motion tracking methods exist, we wanted to check the feasibility of using clustering methods from class to image segment and generate 3D projections from our clustering results. In this paper we explore two different clustering methods and the problems we faced in each.

2 Dataset

The two datasets we will be using are videos of us moving our hands around in three dimensions. In both videos the background is a solid color and free of noise to simplify our methods. Another thing to note is that our hands are in high contrast with the rest of the image. We did this to ensure we would be able to use our clustering techniques to segment the image correctly.



(a) Dataset 1 (b) Dataset 2

Figure 1: Our two homemade datasets

3 Clustering Methods

Before we can calculate 3D projections, we'll first need to image segment the object to track. For the purposes of this paper we only attempt to segment one object within the frame. Given that assumption, we tried two clustering methods, k-means and spectral clustering, each of which has their own advantages and disadvantages. Which method to ultimately use depends on the needs of the application.

3.1 K-Means

We'll first start with the k-means style clustering. K-means in this particular application has two clusters, one object and one background. With just two clusters, k-means reduces to simple thresholding on the intensity values of the image. The value of the threshold is effectively set using k-means on these pixel intensities.

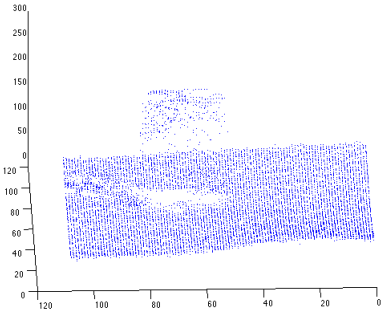


Figure 2: Intensities by pixel location

We would expect to see the object segmented quite easily as long as the object and the background are

homogeneous but of different respective intensities. To run k-means on the image we'll just need to collect each pixel's intensity with each being its own observation in $\mathbf{Z}^1$ (its pixel intensity between 1-255).

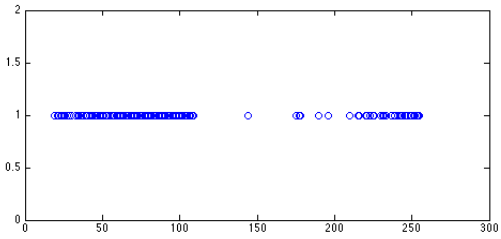


Figure 3: k-means (k=2) → simple thresholding

This can be extended to color images by using all three color values and running k-means on data in $\mathbf{Z}^3$(red, green, and blue intensities).

The main advantages of this method are its speed and its simplicity. We can run k-means on a 100×100 resolution image in a fraction of a second.

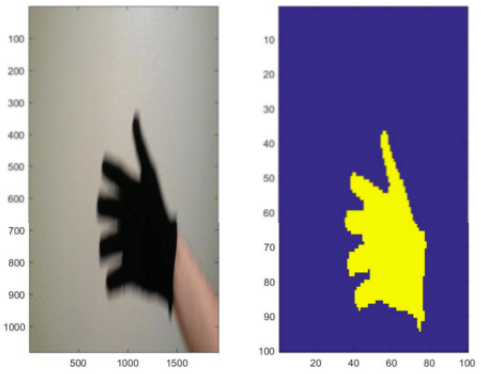


Figure 4: k-means Clustering Results

3.2 Spectral Clustering

Spectral clustering offers more flexibility in image segmentation at the expense of computational time and having to set parameters. Instead of requiring that clusters be homogeneous in pixel intensity, we just require a contrast in pixel intensity at the edges of a given cluster.

3.2.1 Image Spectral Clustering Issues

Since large images contain an enormous amount of total pixels (observations), using spectral techniques become increasingly computationally expensive. If we go from an image size $n \times m$ to $2n \times 2n$, the number of pixels increases from nm to $4nm$. Thus a similarity matrix between all pixels increases from $nm \times nm$ to $4nm \times 4nm$. The computational complexity of calculating eigenvalues/vectors in MATLAB will be at least $O(n^2)$, where $n = 4nm$. As you can imagine, computational time(and memory required) increases faster than what is practical. Due to this dimensionality problem, we've added a preliminary step to reduce the dimensionality of the image before the spectral clustering step.

3.2.2 Algorithm

One way to reduce the number of "observations" prior to the spectral clustering step is to generate super-pixels using the SLIC algorithm [1]. An image suitable for clustering is by nature full of adjacent pixels with similar intensity values. This redundancy allows us to collapse similarly colored areas into individual "super-pixels". The SLIC method essentially runs k-means on the image where k clusters are initially positioned in a regular grid interval. It uses intensity as well as pixel location. Since k-means is generally linear in computational complexity, the issue of dimensionality has been reduced considerably.

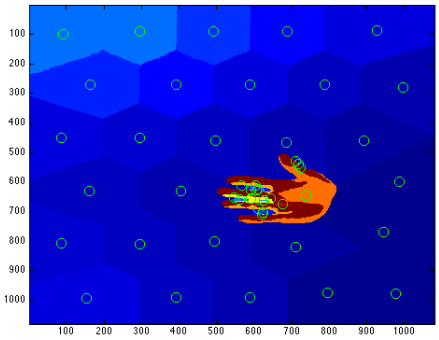


Figure 5: super-pixel locations

Once we have k super-pixels, we now need to compute similarities between each super-pixel. Since we can assume that a cluster will always be contiguous we only need to compute similarities between adjacent super-pixels.

$$S(p_i, p_j) = \begin{cases} e^{-\left(\frac{D_B}{2\sigma^2}\right)} & p_i \text{ and } p_j \text{ are adjacent} \\ 0 & \text{else} \end{cases}$$

We also know that each super-pixel is made up of a sample of pixels with a range of intensities. So one

way to compare similarity is to compute the similarity between the distributions of the pixel intensities. One measure suggested by Tung is to use the Bhattacharyya distance [2].

$$D_B(p,q) = -\ln\left(\sum_{x\in X}\sqrt{p(x)q(x)}\right)$$

where $p(x)$ is the discrete probability distribution of super-pixel p, and $q(x)$ is the discrete probability distribution of super-pixel q.

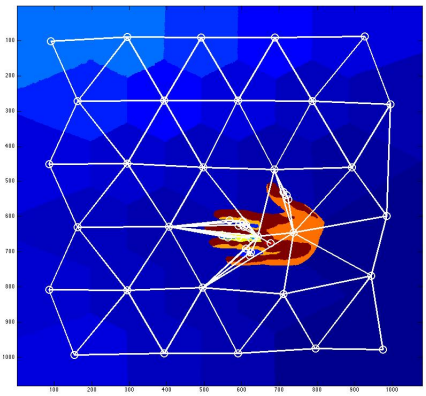


Figure 6: each super-pixel is only "similar" to adjacent super-pixels

When the distributions closely overlap this distance will approach 1, while nonoverlapping distributions will approach 0. With intensity values ranging between 1 and 255, it may be hard to achieve a good idea of the shape of the distribution unless we have a large number of pixels in every super-pixel. To solve this issue we bin the samples into 16 equally spaced bins.

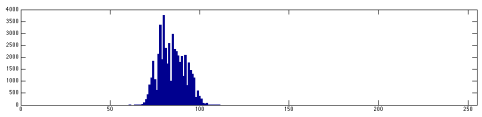


Figure 7: distribution for a "background" super-pixel (before binning)

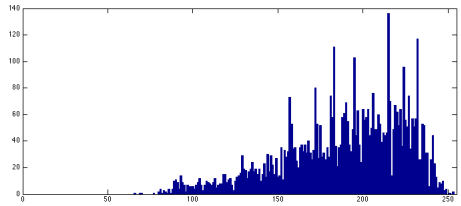
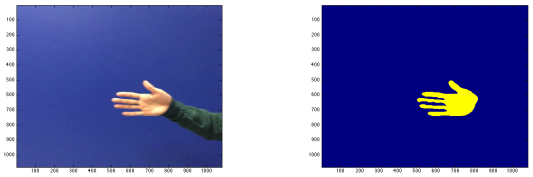


Figure 8: distribution for a "hand" super-pixel (before binning)

We can now create a $k \times k$ affinity matrix which we can pass on to a spectral clustering function, in this case we used normalized cuts [3]. The resulting image segmentation is very clean but takes slightly longer to run than the k-means segmentation.



(a) Original (b) Segmented

Figure 9: Super-Pixel Spectral Clustering Results

Table 1: Computational time of Spectral Clustering methods

Method	**Image Size**	**Affinity Size**	**Time**
SC with eigs	15x15	225x225	0.04 s
SC	15x15	225x225	0.07 s
SC with eigs	30x30	900x900	0.40 s
SC	30x30	900x900	1.96 s
Super-pixel SC	196x196	49x49	3.90 s
SC with eigs	60x60	3600x3600	12.4 s
Super-pixel SC	1080x1080	49x49	31.8 s
SC	60x60	3600x3600	123 s

Using the eigs function in MATLAB takes advantage of the fact that we only need k eigenvalues and eigenvectors. This can reduce computation time significantly as seen here.

We now have two quick methods to segment images. For higher frame-rates and/or less noisy images, k-means would be more suitable. For higher accuracy segmentation but lower framerates, super-pixel spectral clustering is recommended.

4 3-D Projection

Now that we have our clusters, how can we use this information to find the three dimensional position of out object? First let's recap what we information we have at this point. The first thing we know is the field of view of our camera because we are able to look it up online. Another thing we know from our camera is the total number of pixels in the image ($n \times m$ pixels). The next parameter we need is a little strange. If we are interested in returning real measurements we need to know the radius of a circle with the same surface area as our object. Otherwise we can put any number into this parameter and track relative position changes. The final parameter is the number of pixels that out object in occluding in our image which we found out in our clustering step. After we know all these parameters we can plug them into our projection function which we will discuss next.

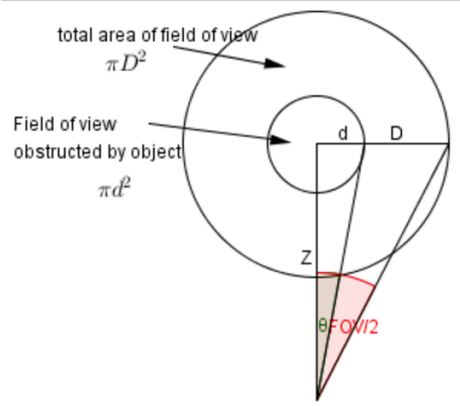


Figure 10: An object obstructing the field of view

The first coordinate we want to find is the z coordinate which is the distance from our camera to the object. We will be using trigonometry to relate the four parameters we discussed previously to the z coordinate.

$$\frac{(\#\text{ pixels in cluster})}{n \times m} = \frac{\pi d^2}{\pi D^2}$$

$$\sqrt{\frac{(\#\text{ pixels in cluster})}{n \times m}} = \frac{d}{D}$$

$$\sin(\theta_1) = \frac{d}{z},\ \sin(\theta_2) = \frac{D}{z}$$

where θ_1 is the angle of the object's obstruction and θ_2 is the angle of the FOV of the camera. d is supplied by the user as an estimated initial wifth of the object (in inches, cm, etc.).

$$\theta_1 = \arcsin(\sin(\theta_2)\frac{d}{D})$$

$$z = d \cdot \tan(\theta_1)$$

The formulas for the x and y coordinates follow similarly.

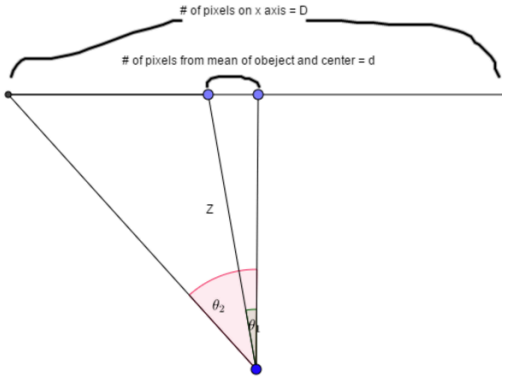


Figure 11: projection in the x direction(similarly in the y axis)

$$\theta_{x1} = \arcsin(\frac{d}{D/2}\theta_{x2})$$

$$x = z \cdot \sin(\theta_{x1})$$

where D is now defined to be $D = n$ (width of the image in pixels)

The y position will be calculated the same way but with $D = m$ (height of the image in pixels)

5 Results

Our final projections show that we can derive a fairly accurate 3D tracking of an object using some basic information from our clustering results.

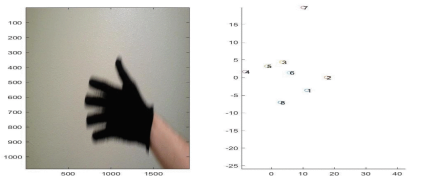


Figure 12: projection results for dataset 1

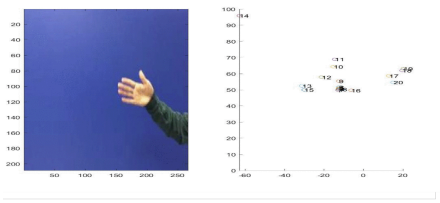


Figure 13: projection results for dataset 2

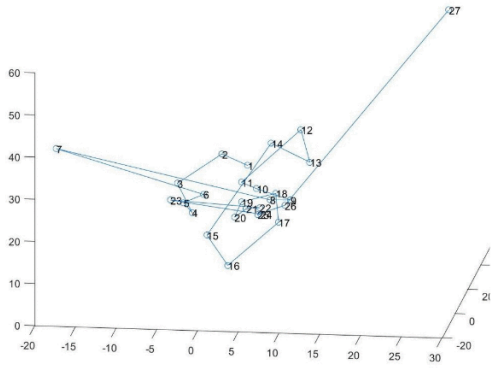


Figure 14: 3D view of the final tracking

6 Future Work

There are many ideas we had in order to improve our model but we unfortunately didn't have time to implement them all. If we had access to actual 3D data points of the object we are trying to track we could optimize our model further. We would also like to be able to detect the number of objects in our video automatically as well as track multiple objects. Another important problem to be solved is correcting for a noisy background so that we can track objects in the real world by creating a more robust clustering model for this type of application. Finally, we would like to use a filter of some kind to account for noise in our position estimates. Lastly, we would like to use these tracking points to predict future object locations within the image frame. If we could predict a future location, we could possibly reduce the clustering phase down to a subimage where the object is likely to be.

References

[1] Radhakrishna Achanta. Slic superpixels. Technical report, EPFL Technical Report, June 2010.

[2] D. Clausi F. Tung, A. Wong. Enabling scalable spectral clustering for image segmentation. *Pattern Recognition*, 43:4069–4076, 2010.

[3] J. Shi and J. Malik. Normalized cuts and image segmentation. *PAMI*, 22(8):888–905, 2000.