3D Object Tracking and Reconstruction from Videos (CS231A Final Project Report)

Ishan Patil*, Benjamin Konyi**, Yang Yang***
*Stanford University{iapatil@stanford.edu}, **Google{bkonyi@google.com}, ***Adobe Corp.{yangyan@adobe.com}

Abstract

The goal of this project was to track single objects in videos and then reconstruct the tracked object as a 3D model. More particularly, after the object to be tracked is selected by the user in the first frame of the video by drawing a bounding box around it, our framework will segment and find corresponding points in consecutive frames and attempt to simultaneously reconstruct this object in 3D space.

1. Introduction

Over the past few years, object tracking and 3D reconstruction have become increasingly critical in many software applications. These techniques allow for us to gain a better understanding between real-world objects and their surroundings using only a camera, making it possible for computation to be done on the resulting insights. Being able to make decisions based on data collected by cheap sensors has many practical applications in business, including analyzing consumer behavior and creating accessible technologies, such as self-driving cars.

Since object tracking and 3D reconstruction has become more and more important over the past few years, we decided to explore this area of research for this project. Our main goal was to implement an object tracking and 3D reconstruction pipeline similar to that described by Feng et al., and then compare our results with other similar techniques. While industry has shifted their efforts towards using deep learning for computer vision tasks, particularly, tracking and reconstruction. However, we decided to work with traditional computer vision methods for our project in order to expand our understanding of computer vision fundamentals.

The algorithm described in [1] can be broken down into various smaller operations, including segmentation, feature detection, object tracking, and 3D reconstruction. We were able to successfully implement a few of these procedures, but were unfortunately unable to get all aspects of the pipeline working using our own code. In this paper, we will discuss the algorithms we used as part of our pipeline, as well as the successes and difficulties we encountered along the way.

2. Background/Related Work

One of the projects submitted in a previous session of CS231A, 3D Person Tracking in Retail Stores [6], implemented object reconstruction using computer vision techniques learned in class, and used a pre-trained model for detecting and tracking objects in the frame. This project used hand labeled images which contained the true correspondences over multiple key-frames and single view geometry methods to perform reconstruction. Photoshop was also used for distortion correction as part of the image pre-processing procedure.

Another prior work similar to our proposed pipeline is a method described by Feng et al. in their paper On-line Object Reconstruction and Tracking for 3D Interaction [1]. In this paper, the first frame is segmented to isolate an object of interested, and then the Shi-Tomasi algorithm is used to detect the initial feature points. In subsequent frames, feature correspondences are found using a Pyramidal Lucas-Kanade Tracker, which can then be used to find a bounding box in 3D space relative to already known objects in the real-world.

In [1], the Shi-Tomasi algorithm is used to detect feature points in the initial frame. In subsequent frames, correspondences are found using a Pyramidal Lucas-Kanade Tracker. With these correspondences, it is possible to estimate the fundamental matrix, and then with known intrinsic parameters, the camera matrix can be found. Segmentation is achieved by Fast Local Kernel Density Estimation. Instead of reconstructing the object in 3D, the bounding cube around the object was found in 3D, making it possible to detect where the object was relative to other world features.

For this project, we made use of the VOT2013 Dataset [2], which is a challenging dataset used for visual object detection and tracking, and was released in 2013. This

dataset contains many different types of labeled data for various types of object recognition tasks, such as people recognition, face recognition, and object tracking. To evaluate our resultsFor evaluating our object tracker, accuracy and robustness metrics are used as described in [8].

3. Approach

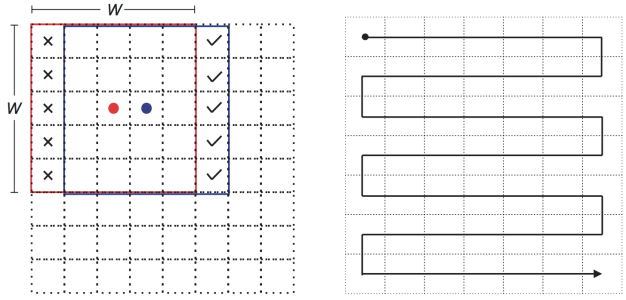
Once the user selects a bounding box around an object of interest in the first frame of the video, our pipeline, which is inspired by [1], comprises of four modules – segmentation and object tracking, feature detection, correspondence tracking across frames, and 3D reconstruction using structure from motion techniques. This section walks through the technical approach that we use to establish each of the above modules.

3.1 Object Segmentation and Tracking

The background points which don't belong to the selected object, in each of the frames of the video, can hinder the final 3D reconstruction process. In order to prevent this, we employ object segmentation as a preliminary "preprocessing" step to extract the silhouette of the object of interest in each of the future frames. As mentioned in [5], there are two possible ways of achieving this segmentation. In the first approach, we could maintain a parametric background model from the frames we have seen up to a certain point of time and use this to enhance our segmentation. This model makes an implicit assumption that the background is stationary. However, in reality, real scenes are not stationary and dynamic shadows, moving background objects as well as camera shaking are common [as also present in the VOT2013 Dataset that we consider for evaluating our approach]. This motivates the second type of method that segments input frames of the video sequence transductively, by using the segmented mask of the previous frame as a non-parametric model to segment the current frame. Using this method, we can make the assumption that the local color distribution at each pixel is consistent between adjacent frames. Fast Local Kernel Density Estimation (FLKDE) is a technique that exploits this particular assumption and can be implemented in a very efficient manner to facilitate online segmentation of the object in the current frame with input as the foreground mask of the object in the previous frame and the previous frame itself. It's naïve implementation works as follows [1]:

- The input binary foreground mask is multiplied element-wise with the previous frame (RGB image) to get the RGB values of the segmented object. Similarly, we get the RGB values of the background from the corresponding background mask (negation of the input foreground mask).
- For each pixel location in the current frame, a foreground color histogram is computed using the RGB values of neighboring foreground pixels of the previous frame in a local window that is centered at the pixel in question. A background color histogram is computed in a similar manner using the RGB values of the neighboring background pixels.
- If the count of the bin to which the RGB value of the pixel in current frame belongs, is higher in the foreground local color histogram than the background local color histogram, the pixel is deemed as a foreground pixel. Otherwise, the pixel is deemed as a background pixel.

The bottleneck in computational efficiency of this approach is the construction of two local color histograms at every pixel. To solve this problem, [5] proposes an interesting method to avoid the repeated computation of the histograms.
As can be seen in Figure 1, the local color histogram at each pixel over the RGB values of foreground and background images of previous frame can be computed by deleting and adding samples. The method is initialized by computing a histogram from scratch for the top-left pixel after padding the previous frame according to the size of the window. In the Figure, this pixel is the red one. Then, the window is slid one pixel to the right, where the blue



(a) (b)

Figure 1: Efficient implementation of the FLKDE algorithm for object segmentation; (a) Updating a histogram as simply adding (marked by tick) and subtracting (marked by cross) RGB samples (b) traversal order of the sliding window.

pixel becomes the center of the window. The color histogram for this new pixel location is obtained by subtracting the counts of RGB samples corresponding to the "cross" pixel locations and adding the counts of the RGB samples corresponding to the "tick" locations. This process can then be repeated for every pixel in the image if we slide the local window in the traversal order as shown in Figure 1(b). In our project we provide our own implementations in Python for both the naïve and this fast

version of the algorithm. The output probability map (binary mask) for the foreground is very sparse and hence some morphological post-processing is carried out. This includes dilation with a rectangular structuring element and small region removal to keep only a single connected component in the output foreground mask.
If we only use this method for segmentation, the estimate of the foreground is very liberal. In addition, since the predicted mask for one frame is given as input to the next one, the region predicted for the foreground (object) keeps growing in every frame and eventually it starts showing

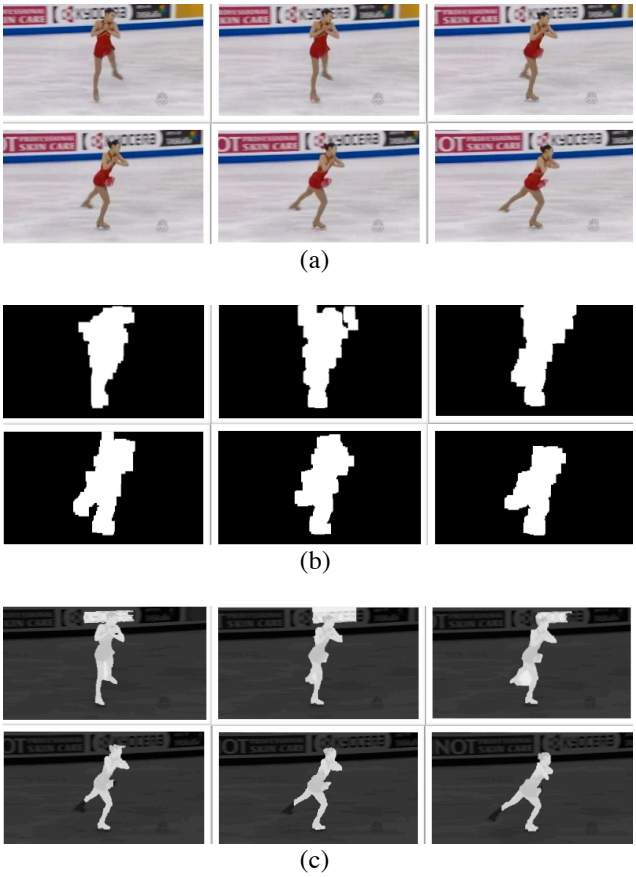
Figure 2: Accumulation of errors in the foreground mask predicted by FLKDE for object (car) selected by user in first frame.

the entire current frame as foreground (after 15-20 frames depending on the size of the object being segmented). This problem can be visualized in Figure 2, where the predicted foreground mask for each frame is blended with the image and shown in grayscale.

To obtain a more realistic estimate, we decided to use the GrabCut algorithm for automatic foreground-background segmentation. Although the original version of GrabCut as introduced in [6] and in the class, is based on an interactive user-guided interface, we used the OpenCV implementation of this algorithm which takes as input the image, number of iterations and a bounding box in which the segmentation is to be performed. Here, we input an axis aligned bounding box that covers all the foreground pixels as predicted by FLKDE. This simple modification (or extension) over the above method gives a very good improvement in the segmentation of the object. As seen in Figure 3, on the "iceskater" video sequence in the VOT2013 dataset, we can observe a good segmentation of the foreground object using this two-step method.

Although there is a small part of the background that is also predicted as foreground in some of the frames, the predicted bounding box has a good amount of overlap with the ground truth bounding box of the object from the VOT dataset and hence, we mark even those frames as correctly tracked (as per the evaluation criteria mentioned in [8]).

Although our original goal was to get these segmentations to aid feature points extraction only within the object area, in this process, we also came up with our own customized object tracker and hence we provide it's qualitative and quantitative evaluation as well in the Results section.



(a) (b) (c)

Figure 3: Object segmentation on 6 consecutive frames from the "Iceskater" video sequence in the VOT2013 Dataset; (a) Input frames; (b) Predicted foreground masks by FLKDE; (c) Refined foreground masks using GrabCut overlaid on top of input images for visualization

3.1. Feature Detection

For this project, we are using the Shi-Tomasi corner detection algorithm [4] to find notable features associated with objects selected by the user. This detector has three hyperparameters that need to be provided: a size for a sliding window, a maximum for the number of feature points to collect, and a minimum distance allowed between adjacent feature points. This step will be

performed after the segmentation stage of the pipeline, which will help isolate the object of interest from the background of the image, ensuring that all feature points are associated with the selected object. It is important that these feature points accurately represent features of our object, as these feature points will be critical when it comes to the next stage of the pipeline: feature tracking.

3.2. Feature Tracking

In order to track feature correspondences across frames, we used the Pyramidal Lucas-Kanade Tracker on the Shi-Tomasi feature points detected in the segmented area as described above. The Pyramidal Lucas-Kanade optical flow algorithm relies on the assumption that objects will only move relatively short distances between frames, and that pixels will generally have similar motion to neighboring pixels in a given window. These assumptions allow for the use of the optical flow equation to model the movements of pixels between frames. This equation can then be solved using least squares optimization, leveraging the information from neighboring pixels.

$$\frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t = 0$$

Figure 4: The optical flow equation, which is used to estimate movements of points over time.

Unlike the standard Lucas-Kanade algorithm, Pyramidal Lucas-Kanade allows for a relaxation of the assumption that pixels will only move short distances between frames. This is done by creating a pyramid of images associated with each frame (see Figure 5 below), where each level of the pyramid is a scaled version of the image, and then estimating movements based on optical flow calculations from each level of the pyramid. This allows for the tracking of features that can potentially move across the scene in a single frame, as this movement can be detected in the smaller levels of the pyramid.

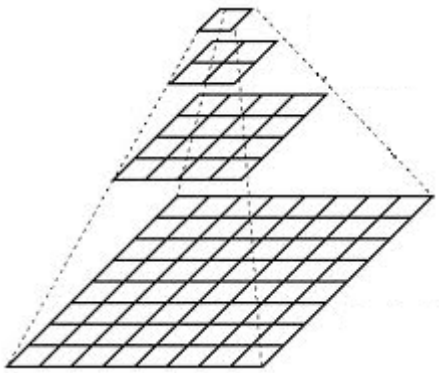


Figure 5: An example of a 4-level image pyramid [9]

3.3. 3D Reconstruction

As the camera or the object moves, new frames together with new 3D points and new measurements of existing points are added. This process expands the structure of the target object and the viewpoints under which the object can be tracked. After detecting the corresponding points in the new frame to all of the ones in the previous frames using the "Feature Tracking" module, and discarding outliers (which results in large errors when reprojecting corresponding 3D points into the object area of the current frame), we will use structure from motion techniques such as affine factorization method and bundle adjustment to recover the object model.

Approach 1: Affine Factorization Method
First, we leveraged a simple naïve method to estimate the 3D points. The factorization method uses only the provided correspondences without any parameter estimation, by assuming the affine transformation. The problem can be solved by computing the SVD of the correspondence matrix. The topic has been covered extensively in lecture, so we will not go into detail here.

Approach 2: Bundle Adjustment
This method comprises of the following steps:

- Estimate fundamental matrix with RANSAC

  After obtaining correspondences between keyframes, we deploy eight-point-algorithm with RANSAC to estimate the fundamental matrix. The steps are processed below in an iterative way:

  -Randomly select eight correspondences
  -Compute the putative model with equation $\mathbf{p}^T \mathbf{F}\ \mathbf{p}' = \mathbf{0}$, where p and p' are the correspondences on two key-frames.
  -Count the number of inliers and record the max inlier iteration and model.

  The process iterates for 100 times, then we save the model with max number of inliers.

- Estimate Camera Matrix

  After using RANSAC to estimate the fundamental matrix, we know from lecture that Fb = 0.So we can estimate b from calculating the SVD of F and get the pair of camera matrices.

- Estimate location of 3D point

  With linear triangulation, we calculate an initial estimate of the 3D coordinates. Using the Newton Levenberg-Marquardt algorithm, we calculate the Jacobian matrix and then finally the the reprojection error. Iterate the above process 100 times and take the final result as the 3D coordinates.

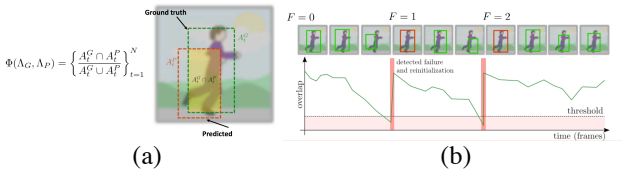
4. Experiments

4.1 Object Segmentation and Tracking

Since our combination of methods used for object segmentation/ tracking, especially FLKDE and GrabCut have not been well evaluated as a whole on this dataset before, we decided to show both quantitative as well as qualitative results on 4 video sequences [Cup, Juice, Iceskater, Torus] which have difficulty of tracking (as per the VOT2013 Benchmark [9]) ranging from simple [Cup] to moderate [Juice] to difficult [Iceskater, Torus]. Qualitative results are shown by simply making a 10fps video of the predicted foreground masks on each frame overlaid over the corresponding actual frame. Below are links to videos for four of these sequences:

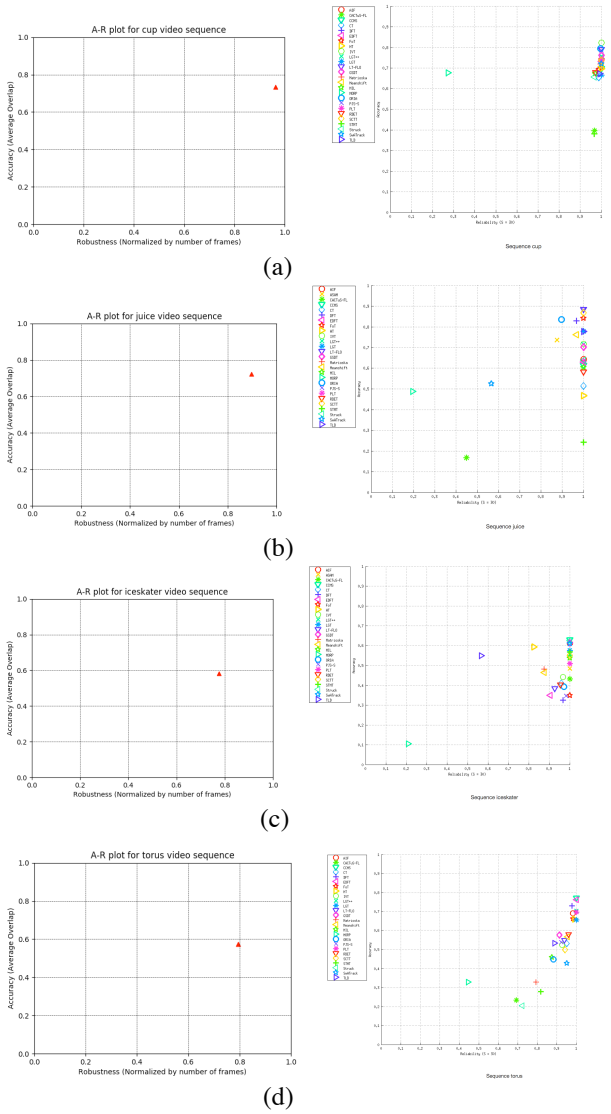
On the “cup” sequence - http://bit.ly/2qX9i1i
On the “juice” sequence - http://bit.ly/2rtBcFI
On the “iceskater” sequence - http://bit.ly/2safBD9
One the “torus” sequence - http://bit.ly/2qPeJQD

There is one important qualification that has to be made here; as per the rules for evaluating an object tracker that are laid out in [8], and are also used when judging the submitted trackers as a part of the VOT2013 challenge, the tracker can be reinitialized with the ground-truth bounding box when the overlap ratio between that and the predicted bounding-box is less than a certain threshold value. This sort of reinitialization is incorporated in our framework to make the evaluation of our tracker more sound. The above videos also incorporate it. This leads us to two important metrics to capture the performance of the tracker – Accuracy and Robustness, as mentioned in [8] and illustrated in Figure 6. Accuracy refers to the average overlap between the ground-truth bounding box and the predicted bounding box for all frames in the video. Robustness is a measure of how many times we needed to reinitialize the tracker when the overlap ration falls below a threshold. If we divide the number of times we needed to reinitialize a particular video sequence by the number of frames in that sequence, we can express the robustness as a values between 0 and 1. Thus a tracker can be visualized as a point in the Accuracy-Robustness space (A-R plot). These comprises the quantitative evaluation of our tracker



(a) (b)

Figure 6: Quantitative Evaluation metrics for object tracking; (a) Accuracy; (b) Robustness



(a) (b) (c) (d)

Figure 7: Comparing Accuracy-Robustness Plot of our tracker with other submitted trackers for the VOT2013 Challenge; On sequences (a) Cup; (b) Juice; (c) Iceskater; (d) Torus

and is shown in Figure 7 (where we have assumed a threshold of 0 for re-initialization as suggested in [8]), where we also compare our tracker in this A-R space to other submitted trackers in the competition. As can be seen from the plot, our tracker is somewhere between the best and worst trackers submitted for the competition. Some of the advantages of this tracker are its high robustness and accuracy for large objects and also a good segmented silhouette of the object (thanks to GrabCut)

which can be fed as input to the feature detection module. But this tracker is not very robust to occlusions and fast camera movement since FLKDE being a probabilistic filtering technique fails completely sometimes (Although this happens quite rarely).

4.2 Feature Detection

Since the Shi-Tomasi corner detection is a relatively straightforward algorithm, we decided to implement it ourselves for use in our pipeline. We were then compared the performance of our Shi-Tomasi implementation against its OpenCV implementation, cv2.goodFeaturesToTrack. For our trials and our pipeline, we used a 3x3 sliding window with gaussian smoothing, requested no more than 25 feature points, and defined a minimum distance of 10 pixels between adjacent feature points. As seen in Figure 8, our implementation of the Shi-Tomasi algorithm performs relatively well when compared to the OpenCV implementation, with many of the same feature points detected by both implementations. The differences in the sets of feature points is likely due to mismatching parameters to the algorithm, as it's not completely clear what some of the defaults in OpenCV are set to. Although our implementation of Shi-Tomasi produces reasonable results compared to the OpenCV version of the algorithm, our version is much slower. OpenCV can find the feature points above in a fraction of a second, whereas our implementation takes over a minute to run. This is due to the fact that our implementation is done completely in Python, while the OpenCV implementation is written in C++ and is parallelized.

(a) (b)

Figure 8: Comparison of outputs from our implementation of Shi-Tomasi Corner Detection in green (a) and the OpenCV implementation in blue (b).

4.3 Feature Tracking

In our midterm progres report, we mentioned that we used OpenCV's built-in Lukas-Kadade implementation, cv2.calcOpticalFlowPyrLK, to perform tracking of features across images. Since then, we spent time on trying to implement a version of the Pyramidal Lukas-Kanade algorithm ourselves so we could have the majority of our pipeline written solely by us. Unfortunately, we were unable to get our implementation of the algorithm to a working state. As shown in Figure 9 below, our implementation of the algorithm fails to track the feature point across frames, failing almost immediately. below, our implementation of the algorithm fails to track the feature point across frames, failing almost immediately.

(a)

(b)

Figure 9: The feature point from the first frame (a) is clearly not tracked properly as seen a few frames later (b)

Since we were unable to get our implementation of the algorithm working, we simply used the built-in version from OpenCV, cv2.calcOpticalFlowPyrLK, to track our feature points across frames. As seen below in Figure 10, the Lucas-Kanade algorithm has no problem tracking our feature points across frames. In order to get the best results, we provided the algorithm with our segmented frames instead of the raw frames from the video.

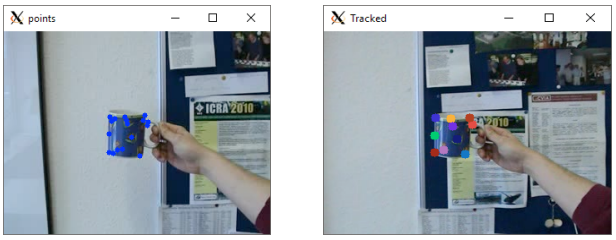


Figure 10: An example of OpenCV's Lucas-Kanade method on our segmented frames in the “cup” sequence. The original feature points are shown in the first frame (left) and the updated feature points in frame 51 (right) are shown.

4.4 3D Reconstruction

Due to the limitations of our dataset, which only included ground truth bounding boxes, we needed to estimate all of the parameters associated with our scene, including the fundamental matrix, camera matrix, and finally the 3D coordinates of the object we are interested in. As we were unable to successfully implement the Lucas-Kanade optical flow algorithm, we used OpenCV's implementation to track feature correspondences across frames, which were then used for reconstruction. Unfortunately, our correspondences did not appear to be very accurate, and would significantly drift away from the object being tracked as the video progressed. As a result, our attempts at creating a 3D reconstruction of the tracked object left much to be desired as we could not accurately estimate the fundamental matrix, camera matrix, or 3D coordinates of our object. An example of our 3D reconstruction results can be seen below in Figure 11, which is supposed to be a 3D image of the cup in the VOT2013.

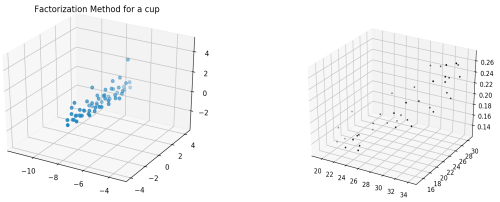


Figure 11: 3D reconstruction with affine factorization method on the left. And bundle adjustment on the right.

5 Conclusion and Future Work

In this project, we set out to complete four main modules in the object tracking and reconstruction pipeline. In the first step of object segmentation/ object tracking from videos, we proposed our own object tracker that employed a probabilistic filtering method called Fast Local Kernel Density Estimation and GrabCut as post processing step. We also provided a comparative qualitative and quantitative evaluation of this tracker to the ones submitted in the VOT2013 Challenge.

We also attempted to implement the Lucas-Kanade method for feature point tracking and tried to create 3D reconstructions of our tracked objects, both of which produced incorrect or mediocre results. The major obstacle for reconstruction was the lack of key parameters, like camera's intrinsic parameters and not perfect correspondences. For good results, some manual measurements such as hand labeling the correspondences or obtain camera intrinsic parameters would hopefully help.

Overall, the major contributions or learning from this project for us have been that we wrote our own implementations (both naïve and efficient) for FLKDE, integrated it GrabCut and incorporated reinitialization from the ground truth data to provide sound evaluation of our own implementation. Further, we also wrote our own implementations for Shi-Tomasi Feature point detector, Lukas-Kanade Optical Flow tracker and estimation of fundamental matrix using RANSAC though not all of them were very successful towards the end. The course notes and lectures were particularly helpful both for inspiration (GrabCut, Structure from Motion etc.) and implementation (Estimating F). All our source code can be found on this GitHub repo: https://github.com/ecilay/3d-Object-Tracking-in-Video

Some directions of improvement particularly in tracking which we evaluated deeply, could be implementation of dynamic global color model along with the local color model in FLKDE to make the segmentation most robust with respect to outliers. Of course, tracking and reconstruction could be also improved upon. Lastly, it would be very interesting to see how this method compares to one based on Deep Learning to track objects as introduced in [10].

Acknowledgments

We would like to thank Prof. Silvio Savarese to introduce us to most of the interesting computer vision techniques that we used in this project and learnt otherwise throughout the course. We would also like to thank our Project TA Boris for his advice on our technical approach and comments on the project.

References

[1] Feng, Y., Wu, Y., & Fan, L. (2012). On-line Object Reconstruction and Tracking for 3D Interaction.2012 IEEE International Conference on Multimedia and Expo. doi:10.1109/icme.2012.144

[2]: Visual Object Tracking (VOT) 2013 challenge dataset.

[3]: Yi, W., Jongwoo, L., Ming-Hsuan, Y. (2013). Online Object Tracking: A Benchmark. CVPR2013, Computer Vision Foundation.

[4]: Shi, J., & T. (1994). Good features to track. Proceedings of IEEE Conference on Computer Vision and Pattern Recognition CVPR-94. doi:10.1109/cvpr.1994.323794

[5]: F. Zhong, X. Qin, and Q. Peng, “Transductive segmentation of live video with non-stationary background,” in Proc. CVPR’10

[6]: R.Kaplan, M, Yu. "3D Person Tracking in Retail Store". Stanford University. 2016 CS231A class project.

[7]: Rother, Carsten, Vladimir Kolmogorov, and Andrew Blake. "Grabcut: Interactive foreground extraction using iterated graph cuts." ACM transactions on graphics (TOG). Vol. 23. No. 3. ACM, 2004.

[8]: Čehovin, Luka, Aleš Leonardis, and Matej Kristan. "Visual object tracking performance measures revisited." IEEE Transactions on Image Processing 25.3 (2016): 1261-1274.

[9]:Image Pyramids. (n.d.). Retrieved June 10, 2017, from http://docs.opencv.org/2.4/doc/tutorials/imgproc/pyramids/pyramids.html

[10]: Xiang, Yu, Alexandre Alahi, and Silvio Savarese. "Learning to track: Online multi-object tracking by decision making." Proceedings of the IEEE International Conference on Computer Vision. 2015.