Physical Manipulation of Virtual Holograms Using Proxies

Michael A. Lin
Stanford University
Department of Mechanical Engineering
mlinyang@stanford.edu

Alexa F. Siu
Stanford University
Department of Mechanical Engineering
afsiu@stanford.edu

Abstract

Head-mounted optical see-through displays allow users to interact with digital content in more immersive ways. While much work has been done on the display side, we still rely on abstracted controls for input/output. In this work, we introduce a method for more natural interaction with virtual holograms . Our method uses physical proxies as handles for manipulating holograms. To achieve this, we use a probabilistic Bayesian model for 3D -object tracking and pose estimation. This method only requires a generic model of the object that is being tracked and achieves reasonable real time performance. We show its implementation and system integration with the Microsoft HoloLens.

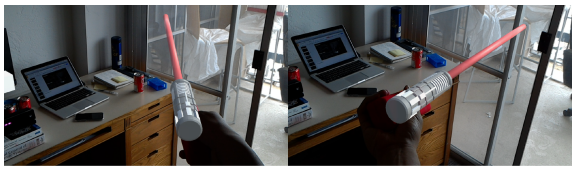

Figure 1. Light saber hologram overlaid on a physical proxy as seen from the HoloLens

Note: Michael Lin is using this project for both CS231A and EE267 final projects.

1. Introduction

Recent technological advances in head-mounted optical-see-through displays (i.e. Microsoft HoloLens), have bridged the gap between the digital and the physical world. These mixed reality devices combine world tracking and head-mounted displays to show 3D image content that is anchored to the physical world. However, these efforts have mostly just increased use of humans visual display channels while still relying on the traditional point-and-click input for manipulating digital content.

In this project, we hope to bridge the gap between the input controls (i.e. pointer, mouse, keyboard) and the graphical output (i.e. the display) such that users can leverage the rich affordances of physical objects. Billinghurst et. al. defined Tangible AR interfaces as those in which: 1) virtual objects are registered to a real physical object (tangible) and 2) tangibles are used to manipulate the virtual objects [1]. Towards this vision of seamless tangible AR, we propose a novel way of interacting with holograms embedded in the real world. We propose using physical objects that are typically found in our surroundings (e.g. a smartphone, an eraser, a water bottle, a coffee mug etc) as handles or physical proxies to which holograms can virtually attach. Instead of using abstracted controls, the user would be able to manipulate the hologram just as they would manipulate any other physical object.

The ideal system would be able to track any arbitrary object chosen by the user as a proxy. It would then superimpose a hologram at the grasping location. For example, if the hologram were a light saber and the proxy was a red box, the system would align and overlay the sabers handle to the box as this is considered the human choice of the sabers graspable region, as shown in Figure 1.

1.1. Design considerations

We cover the main challenges we need to address in order to achieve the system we described. This requirements provided important metrics in guiding our solution implementation.

Real-time performance. The proposed solution should be able to process the RGBD frames in real-time in order to allow true interactivity with the objects.

6-DOF pose estimation. Since the system needs to render a hologram overlaid on the physical object, we must be able to recover the object pose in order to correctly render the hologram.

Uncalibrated environment/minimal overhead setup. If this is a system to be used for quickly testing prototypes or in gaming applications, we want it to have minimal setup requirements such as camera calibration, parameter tunings, and reliance on markers.

Robust to occlusions. We envision users handling the ob-

jects with their hands, therefore the tracking solution should be robust to occlusion otherwise the interaction cannot happen smoothly.

Low computational cost compatible with head-mounted display. The algorithm would ideally run untethered in the head-mounted display. In this project, we use the Microsoft HoloLens. Wearable computing devices are not as powerful as desktop computers nor do they allow easy computation in the GPU therefore the solution should not be computationally heavy.

Based on the aforementioned design considerations, we propose using a probabilistic model-based tracking algorithm as detailed by Yuheng et al. [2]. This algorithm enables real-time tracking on a CPU with enhanced speed if also parallelizing computation on the GPU; but requires a model of the tracking object. In the rest of this paper, we discuss our implementation and integration with the HoloLens.

2. Related Work

2.1. Physical interaction in Augmented Reality

Billinghurst et al. [3] studies the idea of tangible augmented reality where holograms were locked onto markers such that multiple AR headset users would see the same holograms and share the same interaction space.

Garret et al. [4] presented a tracking method that used RGB-D and achieved reasonable accuracy, however, they required of very accurate model of the object being tracked. Also, they limited the experience to just a display of the holograms on a 2D screen rather than using an 3D display such as the HoloLens.

Araujo et al. [5] shows a very interesting experiment where a robotic arm moves its end-effector to where the user is about to interact with a hologram to provide the appropriate kinesthetic feedback. Users wear a virtual reality display. Although it is a very interesting idea, it would not work as well for augmented reality display since seeing the robotic arm moving towards your hand would significantly change the experience. Also, their implementation does not allow the user to have other types of interaction with the object such as holding it or picking it up.

2.2. Methods for real-time tracking with computer vision

Some of the earliest methods for real-time markerless tracking are edge-based. One example is that proposed by Lowe et al. [6]. In their method, parametrized 3D models are iteratively matched to image features. The trade-off is that it requires reliable feature recognition with few outliers to minimize matching errors. Moreover, increasing number of features to match results in higher computational cost.

Koller et al. [7] proposed a similar method but instead of object specific models, they use generic models represented by 3D polyhedrals. They formulate a distance function for matching the model features to the image extracted features. At each step, solve for a pose estimate by minimizing this function using the Levenberg-Marquardt algorithm.

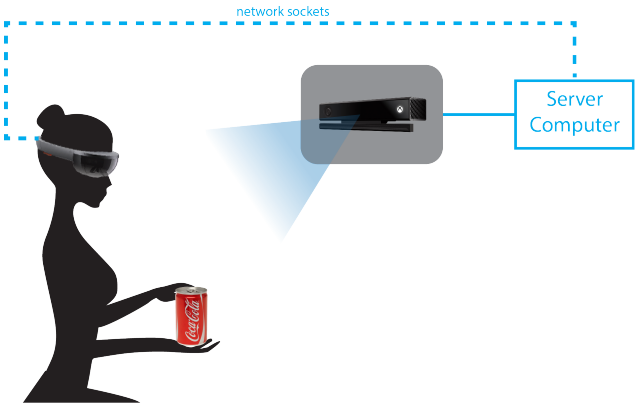

Figure 2. System setup

Rad et al. [8] approached the problem of markerless tracking by training a CNN on RGB-D data set of tabletop objects. Although this method seem to generalize for tracking pose of similar object to those in the dataset, it is not robust to object occlusions.

Particle filtering methods for 3D objects have been proposed. One example is that by Choi et. al. [9]. This method requires a known 3D mesh model. Each particle's likelihood is evaluated on the RGBD images over time. Because of the large number of particles, the computational cost is really high and real-time performance can only be achieved by parallelizing computation on the GPU.

Bibby et. al. proposed a method using pixel-wise posteriors [10]. This method is fast and relies on image segmentation based on an object color appearance model. While their paper shows performance when tracking only in 3D, they mention its extensibility to 3D object tracking.

[11] and [2] extend the method by Bibby et. al. to tracking in 3D. One important difference is that they optimize for likelihoods to find a pose estimate. We adapted this method to use for tracking in the HoloLens.

3. Technical Approach

Our overall system, as illustrated in Figure 2, consisted of three main components: Microsoft HoloLens for displaying 3D information, Microsoft Kinect used for real-time tracking and simple physical objects found in our everyday surrounding, such as coke cans, pencil case or tennis balls.

3.1. System Integration with HoloLens Platform

Although the authors of the tracking algorithm provide a full implementation, there is no documentation or support in

either creating models for new objects or, more importantly for us, no support for integrating with other platforms such as HoloLens.

We developed the algorithm using Unity3D/C# so that it was compatible with the HoloLens. Despite the Hololens having a depth camera, currently developers are only given access to its RGB camera. Since we required depth data for tracking, we used data from an anchored Microsoft Kinect v2 connected to a server computer (Figure 2). The Kinect allows us to obtain both RGB frames (1920x1080) and depth frames (512x424). We align the RGB and depth frames by obtaining calibration parameters from the Kinect cameras which results in frames of 512x424. To obtain the 3D point cloud necessary for the algorithm, we unproject the points to 3D using the camera calibration matrix K.

In our system setup, the server computer handled the algorithm computation and output a pose estimate which was communicated to the HoloLens using network sockets (TCP protocol). This pose estimates ${}^{\mathcal{K}}P^{\mathcal{O}}$ are relative to the Kinect frame of reference $\mathcal{K}$. The HoloLens is capable of tracking the room using Simultaneous Localization and Mapping (SLAM) and then display holograms fixed in the room or world reference frame $\mathcal{W}$. In order to update the pose of the holograms in $\mathcal{W}$ we do an initial alignment to the Kinect with the HoloLens (shown as the white box overlayed on the Kinect in Figure 3). This alignment gives us the isometric transform (pose and rotation) from the world reference frame to the Kinect ${}^{\mathcal{W}}M^{\mathcal{K}}$. Then in order to set the position of the hologram object we transform it by the inverse to obtain the correct position in world reference frame

$$ {}^{\mathcal{W}}P^{\mathcal{O}} = ({}^{\mathcal{W}}M^{\mathcal{K}})^{-1}({}^{\mathcal{K}}P^{\mathcal{O}}) \tag{1} $$

Notation[1]

3.2. Real-time Tracking with the Kinect

3.2.1 Generating the Model as SDF

Yuheng et al. proposed to represent the model as a bag of voxels using 3D Signed Distance Function (SDF) since it is a convenient way of encoding the model surface or distance to the surface. Figure 4 Left shows an example of a 2D signed distance function for a circle. The contour of the circle is marked as 0. In the 3D representation we create a 200mm x 200mm x 200mm cube of voxels in which we center our model as shown in Figure 4 Right. Intuitively this is a good search space setup, since the point cloud data from the sensor is projected on these voxels and the gradient of the distance value informs which way to update the pose.

As mentioned in earlier section, the authors did not include well documented method for generating a bag of voxel for any object. As part of our contributions to this work, we have implemented a Matlab script that takes in any mesh that fits within a 200mm x 200mm x 200mm and outputs a SDF model that can be quickly loaded into the tracking algorithm. With this tool we are able to find any object in our surrounding and import it as a model. Steps to do this are simple:

1. Find an object that has a uniform color. (e.g. a banana that is a uniform yellow color)
2. Measure it roughly and approximate it with a primitive shape such as cylinders, cubes or spheres.
3. Create the mesh quickly in a 3D mesh editing tool such a Blender.
4. Pass it through our custom Matlab script to obtain the model as SDF.

[1]The notation I use here is ${}^{\mathcal{A}}P^{\mathcal{B}}$ the transform of reference frame B from reference frame A

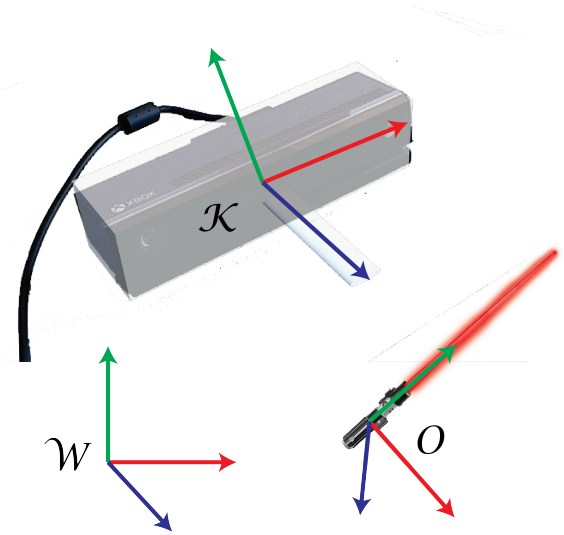

Figure 3. Kinect and HoloLens reference frames

3.2.2 Generative Model

The graphical model for the tracking algorithm is illustrated in Fig. 5. The object shape model Φ and the object pose p inform the observed RGBD image Ω, which we refer to as appearance model. The pose is defined as the 6 degrees-of-freedom of the rigid object we are trying to track which we will denote as just p in this section for simplicity. The object shape model is represented as a bag of voxels ($V = \{in, out\}$) as explained previously. This is an efficient way of representing surface geometry. Finally, our sensor data is obtained as a point cloud representation of the environment.

To solve for a pose using these three components, we want to maximize the likelihood $P(\Omega \mid p, \Phi)$ as a function of p. We know each pixel in Ω is generated from a unique

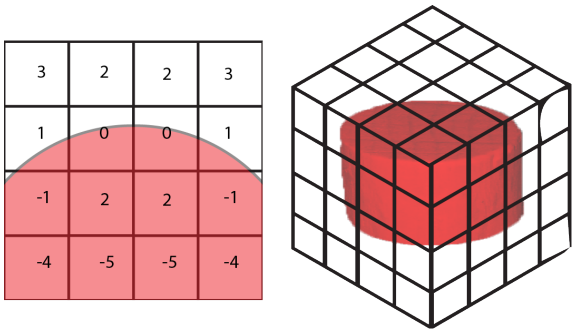

Figure 4. Signed distance function illustration

voxel sampled from Φ. We assume that pixels are conditionally independent. We define the image by its pixels $\{x, y\}$ and its corresponding projected voxel V. Using Bayes and our generative model:

$$P(p, \Phi \mid \{x_i, y_j, V_{i(j)}\}) \sim P(\{x_i, y_j, V_{i(j)}\} \mid p, \Phi) P(p, \Phi)$$

The first term in this equation is the prior, the second term the likelihood, and the third term the prior. The likelihood term usually dominates in this equation. Therefore, even without a good prior and given a hypothesis, the likelihood is a good estimate of how good the hypothesis is. The goal then is to find the maximum likelihood estimate for a given pose hypothesis. Since we assume pixel-wise independence, the likelihood is computed as the product over all pixels. Moreover, if we assume V is fixed and marginalize the equation over V, we obtain the likelihood as:

$$P(x, y \mid p, \Phi) = \sum_{k=in,out} \{P(x \mid \Phi, p, V = k) P(V = k \mid y)\} \tag{2}$$

The object appearance model is represented as color histograms of the foreground and background. The foreground is the region which contains the object of interest while the background is everything else. To capture the foreground and background , we represent these as uniform distributions using the Dirac delta and Heaviside functions respectively. The Heaviside function is positive for any value above 0 and the Dirac delta is positive only at 0 which is how our SDF representation of the model illustrate on the object surface ($SDFvalue = 0$) and outside the object ($SDFvalue > 0$). The Heaviside and Dirac delta functions definitions are:

$$H(z) = \frac{1}{1 + e^{-z/2}} \tag{3}$$

$$\delta(z) = \frac{2e^{-z/2}}{(1 + e^{-z/2})^2} \tag{4}$$

Using these definitions, the likelihood of a single pixel x, y on the object surface (foreground) is:

$$P(x, y \mid \Phi, p, V = on) = \frac{\delta(\Phi)}{\sum_{\Omega} \delta(\Phi)} \tag{5}$$

Similarly, the likelihood of a single pixel outside the object surface (background) is:

$$P(x, y \mid \Phi, p, V = out) = \frac{H(\Phi)}{\sum_{\Omega} H(\Phi)} \tag{6}$$

Substituting the likelihoods from Eqns. 5 and 6 into Eqn. 2 and taking logs, we obtain the energy function:

$$\frac{\partial E}{\partial \mathbf{p}} = \sum_{\Omega} \{ \frac{(P(y|V = in) \frac{\partial \delta}{\partial \Phi} + P(y|V = out) \frac{\partial H}{\partial \Phi})}{P(x, y|\Phi \mathbf{p})} \frac{\partial \Phi}{\partial X} \frac{\partial X}{\partial \mathbf{p}} \} \tag{7}$$

This algorithm is set as an optimization problem solved using the gradient-descent method of Levenberg-Marquardt (LM) to solve for $\mathbf{p}$. At each frame the goal is to maximize the posterior probability of the pose given the current depth image and 3D shape represented as the SDF. This probability is calculated at a per-pixel likelihood and its energy function is defined as the sum of log-likelihoods of this probability over all pixels. Finally, the gradient that is iteratively minimized, is given by the change in energy over the change in pose.

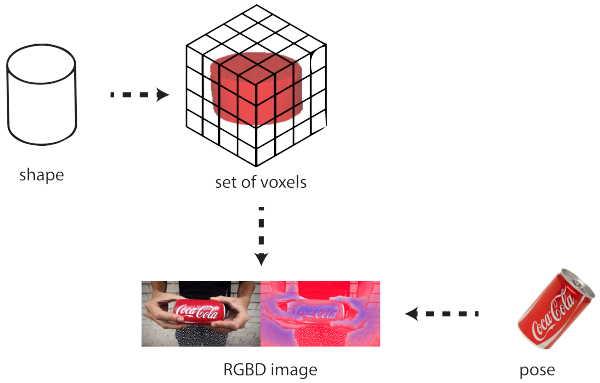

Figure 5. Graphical model of the probabilistic model

4. Results

Figure 6 shows a sequence of frames when tracking a can and the resulting foreground/background segmentation and posterior probability estimations. The foreground is defined as everything inside the blue box and the background is everything outside of it. As the can moves around , it can be seen that the box moves with it. The green pixels highlight

Figure 6. Sequential frames from tracking a red can. The region inside the blue box shows the segmented foreground; everything outside the box is labeled as background. As the can moves around, this box moves with as a result of the algorithm tracking. The green overlayed pixels highlight regions where the posterior probability estimate was greater than 0.5.

regions where the estimated posterior was above 0.5. It can also be seen from the image that most green pixels are concentrated on the red can. Figure 6 shows the results when using the pose estimate to overlay an object.

As a first pass implementation, we used the Kinect's full frame data (512x424 = 220000 pixels). This resulted in slow computation. We made two optimizations to increase speed:

1. Each frame was downsampled by four; resulting in 13000 pixels per frame. Moreover we added a smoothing filter so that each resulting pixel was the result of averaging its four neighboring pixels.

2. Since we segment the image with foreground and background, we limited most computations to be performed only on the bounding box region. This highly sped up the process since the box region is only a fraction of the entire frame.

4.1. Tracking Quality

We tested the tracking algorithm with a set of simple objects that can be found in your surroundings such as a pair of sunglasses, a tennis ball, a banana and a IKEA mug as shown in Figure 7. The models for these were made quickly following the steps specified in section 3.2.1. Our results from tracking the objects are shown in Figure 8. One issue we noticed is that objects that present axis symmetry such as the coke can, the mug and the tennis ball yield a pose that is not stable in their axis of symmetry.

Another test we performed is shown in Figure 9. These images show the whole pipeline running, so they show the view from the HoloLens with the light saber hologram superimposed. In this sequence of images we were aiming at testing occlusion with the hands. As can be seen, the algorithm was very forgiving even with two hands holding onto the object being tracked.

Figure 7. Sample of objects that were tracked.

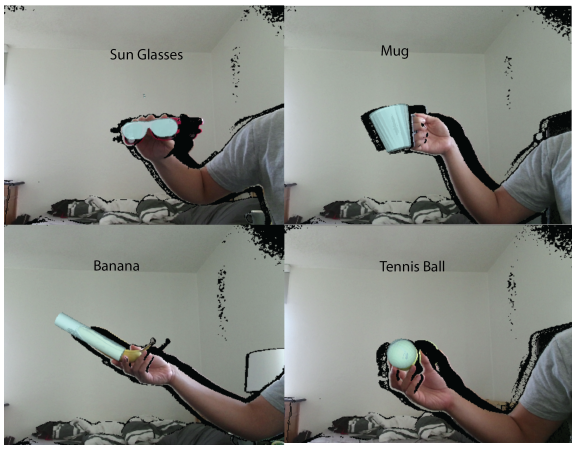

Figure 8. Tracking result on multiple different objects.

5. Limitations & Future Work

Parallelizing computation on the GPU was not done for our implementation therefore there is a limit to how fast the user can interact with the objects. When the user moves too fast the calculated energy may drop and the algorithm loses track of the object. To recover, this requires the user to place the object back on the last pose before it was lost. Possible future work could look into how we can minimize losing track of the object. One possibility is to use not only the current frame for pose estimate but also incorporate previous frames. We know that it is not physically possible for the object to jump a large distance from one frame to the next so based on the previous frame where the object was last seen we could perform a search to recover the object within that region.

Since we couldn't run the algorithm from the HoloLens directly but instead data was sent over the network, this increased the latency in tracking. Moreover, taking images from the head-mounted display doesn't result in images that align with what the user wearing it would see. This is simply due to camera misalignment and differences in how users may wear the headset. In future work we would like to improve the way we align the Kinect with respect to the HoloLens.

6. Conclusion

In this paper we have shown real time markerless tracking and 6-DOF pose estimation robust to hand-occlusion implemented for use in the HoloLens. By providing the implementation of the tracking algorithm in Unity we open to the possibility of having an all-in-one package where user can create models of objects they wish to track using the Unity user interface and immediately load it into the algorithm. This platform enables new opportunities for interaction input and output in AR. We have merely introduced the concept and believe there is much more to explore. For example, what if we could also make virtual holograms interactive? If we could also track the users hands, we could also give holograms behaviors that react to user input. You could have a hologram with a button and press the button by approaching it with your finger.

7. Github Repository

Source code for the HoloLens Unity 3D application, TCP protocol script in C++, and SDF generation script in Matlab can be found in: `https://github.com/michaellin/HoloProxies`.

References

[1] Mark Billinghurst, Raphaël Grasset, Hartmut Seichter, and Andreas Dünser. Towards ambient augmented reality with tangible interfaces. *Human-Computer Interaction. Ambient, Ubiquitous and Intelligent Interaction*, pages 387–396, 2009.

[2] Carl Yuheng Ren, Victor Prisacariu, Olaf Kaehler, Ian Reid, and David Murray. 3d tracking of multiple objects with identical appearance using rgb-d input. In *3D Vision (3DV), 2014 2nd International Conference on*, volume 1, pages 47–54. IEEE, 2014.

[3] Mark Billinghurst, Hirokazu Kato, and Ivan Poupyrev. Tangible augmented reality. *ACM SIGGRAPH ASIA*, 7, 2008.

[4] Timothy Garrett, Saverio Debernardis, Rafael Radkowski, Carl K Chang, Michele Fiorentino, Antonio E Uva, and James Oliver. Rigid object tracking algorithms for low-cost ar devices. In *ASME 2014 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages V01BT02A043–V01BT02A043. American Society of Mechanical Engineers, 2014.

[5] Bruno Araujo, Ricardo Jota, Varun Perumal, Jia Xian Yao, Karan Singh, and Daniel Wigdor. Snake charmer: Physically enabling virtual objects. In *Proceedings of the TEI'16: Tenth International Conference on Tangible, Embedded, and Embodied Interaction*, pages 218–226. ACM, 2016.

[6] David G Lowe. Robust model-based motion tracking through the integration of search and estimation. *International Journal of Computer Vision*, 8(2):113–122, 1992.

[7] Dieter Koller. Moving object recognition and classification based on recursive shape parameter estimation. In *Proc. 12th Israel Conf. Artificial Intelligence, Computer Vision*, volume 2728, page 2, 1993.

[8] Mahdi Rad and Vincent Lepetit. BB8: A scalable, accurate, robust to partial occlusion method for predicting the 3d poses of challenging objects without using depth. *CoRR*, abs/1703.10896, 2017.

[9] Changhyun Choi and Henrik I. Christensen. RGB-d object tracking: A particle filter approach on GPU. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 1084–1091, 2013.

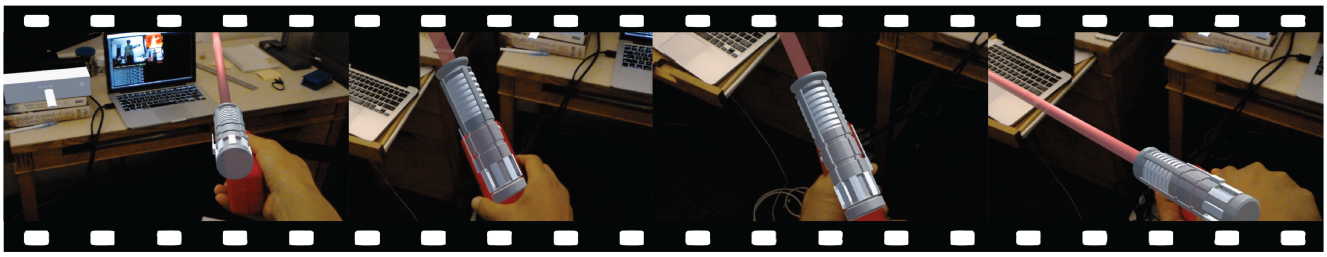

Figure 9. Sequential frames from tracking a red box while being heavily occluded by user's hand.

[10] Charles Bibby and Ian Reid. Robust real-time visual tracking using pixel-wise posteriors. *Computer Vision–ECCV 2008*, pages 831–844, 2008.

[11] Carl Yuheng Ren, Victor Prisacariu, David Murray, and Ian Reid. Star3d: Simultaneous tracking and reconstruction of 3d objects using rgb-d data. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1561–1568, 2013.