Applying Robust Structure from Motion to Markerless Augmented Reality

Jonathan Mooser, Suya You, Ulrich Neumann, Quan Wang
University of Southern California
{mooser, suyay, uneumann, quanwang} @graphics.usc.edu

Abstract

We demonstrate a complete system for markerless augmented reality using robust structure from motion. The proposed system includes two main components. The first is a means of learning the appearance of complex 3D objects and augmenting them with virtual annotations. Its output is a database of recognizable landmarks along with 3D descriptions of accompanying virtual objects. The second component uses this data to recognize the previously learned landmarks, recover camera pose, and render the associated virtual content.

Both components make use of the recently developed subtrack optimization algorithm for structure from motion, which we demonstrate to be a useful tool for both learning the structure of objects and tracking camera pose after recognition. The complete system is demonstrated on several complex real-world examples.

1. Introduction

As interest in augmented reality applications has steadily grown, so has the demand for versatile systems that work in a wide variety of environments. Systems based on artificial markers, for example, provide fast and reliable tracking but are infeasible in situations where the environment cannot be easily prepared with visual landmarks. Recognizing and tracking natural features is theoretically more powerful, but no known algorithm can reliably recognize all types of objects in all cases.

The present work takes a step toward the ultimate goal of recognition and tracking of arbitrary 3D objects in the context of augmented reality. With this in mind, we propose a system comprised of two distinct but interconnected components.

The first component, the offline stage, takes a video stream as input and builds a sparse point cloud model of the target environment. Each keypoint in the cloud is stored with a set of local descriptors that will serve as landmarks for recognition. This process is fully automated and represents the primary function of the offline stage.

As a secondary function, the offline component also includes an interface for creating virtual annotations. A user can interactively manipulate simple geometric primitives and text labels either on top of the point cloud model or directly on the frames of the input sequence. The vertices of the virtual objects are thus defined in the same coordinate frame as the keypoint database. The inclusion of an annotation interface is one of the unique aspects of this work.

While object modeling in general has been considered in the past (see section 2) the problem of positioning virtual content within a natural environment has seldom been addressed.

The second part, the online stage, processes frames sequentially and matches detected keypoints against the keypoint database. The resulting 2D-3D correspondences are used to compute a camera pose and render virtual content.

In its simplest form, the online stage does not require a full structure from motion system; the known 3D locations of the keypoints provide enough information to determine camera pose. However, there may be frames in which there are too few matches to compute a pose. Rather than simply failing at these frames, keypoints that have not been matched to the database can be incorporated and used to determine camera pose, as described in section 4.2.

Because both stages rely on the ability to localize tracked points in 3D space, an accurate structure from motion system is essential. One of the goals of this paper is to demonstrate that the subtrack optimization algorithm, introduced in [10], fulfills this need. Numerical results in section 5...
demonstrates the improvement that subtrack optimization provides.

The key contributions of this paper are as follows:

- We show that the subtrack optimization algorithm may be used to build a keypoint database that is both larger and more accurate than could be achieved using conventional structure from motion techniques.
- We propose an interface for populating the sparse point-cloud model of the real world scene with virtual annotations.
- We describe a method of utilizing keypoints that were not successfully matched to the database. This allows the system to make pose estimations even after the camera moves into a view that was not covered during the model building process.

2. Related Work

The drive to build applications without relying on artificial markers or fiducials has placed tremendous focus on natural feature recognition and tracking in the AR community [4, 5, 7, 8, 12, 13, 14, 17, 18, 19, 21]. Proposed solutions often target a specific class of objects such as urban buildings [4] or planar objects [5, 17]. Others make use of specialized hardware such as calibrated multi-camera rigs [15, 21]. Here, we set out to alleviate these constraints by targeting arbitrary object geometry and using a single off-the-shelf camera.

It is also important to draw a distinction between our system and those that depend on an existing polygonal CAD model of the target objects. Bleser, et al. [2] describe such a system in which edges matched to a CAD model are used to initialize tracking. Park, et al. [12] demonstrate tracking of multiple objects for AR using keypoints. In order to learn the appearance of the target objects, however, a user must manually register correspondences between keyframe images and a pre-built 3D model. Similarly, Platonov, et al. [13], describe an AR maintenance and repair application in which the initial training process requires artificial markers.

Neubert, et al. [11] describe a system that models objects as sets of coplanar edgels that can later be tracked and used to estimate camera pose. The model building phase of that system requires a user to manually identify planar polygonal regions of the target object. Reitmayr, et al. [14] discuss a similar method that tracks planar structures such as rectangles and ellipses. That system constrains the virtual objects to lie in the same plane as the tracked landmarks. In our system, virtual annotations can be freely positioned anywhere in the 3D space of the scene.

Klein and Murray [7] demonstrate another approach to tracking using collections of keypoints. As with our approach, they use structure from motion techniques to build a point cloud and estimate camera poses. They only discuss mapping new scenes however, and do not take on the problem of recognizing previously learned objects.

Zhu, et al. [21], describe a complete system that, like the one discussed here, incorporates both the creation of a landmark database and the process of subsequent recognizing and tracking. The focus of that work, however, is on maximizing the efficiency of a large landmark database and incorporating additional odometry hardware such as an inertial measurement unit. Our focus is on implementing and incorporating robust, single camera structure from motion into an AR application.

Another system that addresses both model creation and object tracking is that of Skrypnyk and Lowe [18]. There, only keypoints that have been matched to the original model are used for pose recovery. Unlike our system, they do not track unmatched points to improve pose estimates or handle previously unseen views. That system also includes an interface for adding virtual content. As described, however, it only allows the user position objects on top of the captured images, and not on top the model. The ability to simultaneously see virtual content on top of the captured images and the model is a key advantage of our system, as explained in section 4.1.3.

Perhaps the work most similar to ours is that of Takomii, et al. [19]. They describe a system in which a scene is modeled as a collection of point landmarks that can later be used to establish camera pose. Several key differences distinguish our work from theirs. In order to build the keypoint database, for example, they propose the use of specialized omnidirectional camera hardware. Here, we use only a single monocular camera. Then, rather than simply applying traditional structure from motion, our system models its scene using the subtrack optimization algorithm. The benefits of this improvement are demonstrated in section 5. Finally, we show how to incorporate unmatched points during the online stage. This provides a measurable improvement in the final results as well.

3. Structure from Motion Using Subtrack Optimization

The subtrack optimization algorithm has previously been shown to provide accurate structure from motion on video sequences [10]. The goal of this paper is to show how it can be applied to an AR application. Because it figures prominently in the system described in section 4, this section provides an overview of the subtrack optimization algorithm.

Given an ordered sequence of captured images, keypoints can be extracted from the first frame using the Shi-Tomasi operator [16] then tracked using standard optical flow techniques. Here, we use a variant of Lucas-Kanade optical flow based on image pyramids [3].

The optical flow process generates a set of keypoint tracks, each consisting of a series of keypoints spanning...
two or more consecutive frames. Each track will continue to grow until optical flow fails or the keypoint drifts out of view, so it may span a few frames or several hundred.

Ideally, all of the keypoints in a given track will correspond to the same 3D point in space (referred to as a structure point), in which case the keypoint track is deemed consistent. Over a long sequence, however, this seldom holds true. Keypoint tracks may be stable temporarily, then drift, then become stable again. Figure 2 illustrates this phenomenon.

A simple solution might be to identify keypoint tracks that cannot be fit to a single structure point and remove them from the computation. Traditional outlier detection schemes such as RANSAC may be used to this end. But such a brute force approach, which would simply label keypoint tracks as inlier or outlier, ultimately discards useful data. A long keypoint track is generally stable over some portion of its lifetime and a more powerful approach would seek to identify those sections and use them.

At the same time, identifying those sets of frames during which a keypoint track remains stable is nontrivial. Simply splitting the track up into fixed sized partitions, for example, would only partially address the problem. Those partitions, no matter how small, may be individually inconsistent. Moreover, if multiple consecutive partitions are all consistent, it would be preferable to consider them as a whole.

The motivation behind the subtrack optimization algorithm is to solve this partitioning problem optimally. That is to say, it sets out to identify the longest possible subtracks that can be deemed consistent. Favoring fewer, longer subtracks is important because it ensures that they span as wide a baseline as possible. If overly aggressive in partitioning a keypoint track, we lose valuable information, and the accuracy of the resulting structure will suffer accordingly.

This idea is illustrated in figure 3. As a hypothetical camera moves from top to bottom, a keypoint is tracked representing a ray in space at each frame. Because those rays do not meet at a common structure point, the six frame track is, by definition, inconsistent. The subtrack spanning frames 1-3 and frames 4-6 are consistent, however, and thus represent an optimal partitioning.

Each subtrack corresponds to a single structure point with its consistency determined by average reprojection error of that point. For keypoint track \( k \), let \( k_j \) and \( P_j \) be the keypoint and camera at frame \( j \), and \( k_{a,b} \) be the subtrack spanning frames \( a \) to \( b \) inclusive. The consistency of \( k_{a,b} \) is given by the error function

\[
E(k_{a,b}) = \min_{\mathbf{X}} \left( \frac{1}{N} \sum_{j=a}^{b} d(P_j \mathbf{X}, k_j)^2 \right) \tag{1}
\]

where \( d \) is Euclidean distance in pixels, and \( N \) is the length of the subtrack. The argument, \( \mathbf{X} \), that minimizes the right side of (1) is the structure point corresponding to \( k_{a,b} \).

In general, the optimal partitioning, \( \hat{p} \), of keypoint track \( k \) is defined in terms of a cost function

\[
C(p) = \sum_{k_{a,b} \in p} (\delta + E(k_{a,b})) \tag{2}
\]

where \( \delta \) is a constant penalty term ensuring that the optimization favors longer subtracks whenever possible.

The number of possible partitionings is exponential in the length of \( k \), so a brute force search would be intractable. As it turns out, however, given an estimate of the camera pose at each frame, the optimal partitioning can be found in low order polynomial time.

The trick is to define the cost function recursively as

\[
C(\hat{p}_0) = 0 \\
C(\hat{p}_1) = \delta \\
C(\hat{p}_n) = \min_{1 \leq a < n} [C(\hat{p}_{a-1}) + \delta + E(k_{a,n})] \tag{3}
\]

where \( \hat{p}_n \) is the optimal partitioning of a track only up to frame \( n \). This recursion can be computed efficiently from the bottom up using dynamic programming. See [10] for a formal proof of its correctness and analysis of its run time.

Although the final partition is optimal in that it minimizes (2), it is not necessarily the case that each subtrack is consistent. Recall that finding long consistent subtracks.

---

**Figure 2.** A point that tracks accurately over some frames, but not over the entire sequence. For a period of time the keypoint accurately tracks a point on the ground. As part of the motor crosses it, however, it begins to drift. It eventually regains stability, but now associated with a different 3D structure point.

**Figure 3.** A hypothetical keypoint track with six keypoints. On the left are six locations of a camera as it moves from the top downward. Each keypoint corresponds to a ray in space. The six rays do not meet at a single point, so there is no structure point that is valid for the entire track. However, subtracks \( k_{1,2,3} \) and \( k_{4,5,6} \) do have valid structure points. The goal of the optimization algorithm is to reliably perform this partitioning.
is the ultimate goal. After optimizing each keypoint track, those subtracks spanning at least three frames and having $E(k_{n,b}) < 1.0$ are deemed consistent; all others are deemed inconsistent. Only the structure points corresponding to consistent subtracks are included in the final structure.

This process forms the basis for building the keypoint database during the offline training stage. It also serves an important role in the online stage, as will be show in section 4.2.

4. System Components

This section details the components that make up the overall architecture shown in figure 1.

4.1. The Offline Stage

For a given application, the offline stage would typically be run only once. Its purpose is to learn the appearance of the environment or target objects and populate the scene with virtual annotations. As input, the offline stage takes a video sequence that ideally captures the environment from as many viewpoints as possible. While it need not exhaustively cover every possible viewpoint, it should roughly cover those from which the system is likely to be initialized.

4.1.1 Building the Point Cloud

The keypoint database consists of a cloud of 3D points, each associated with several descriptor vectors. As keypoints are tracked through the input sequence, the subtrack optimization algorithm is used to determine their locations and incrementally build the complete cloud.

In order to partition a keypoint track, the subtrack optimization algorithm requires an estimated camera pose at each frame. Initially, no poses are known so some method is needed to bootstrap the process. To achieve this, the first frame is selected along with one other early frame, and $E(k_{n,b}) < 1.0$ are deemed consistent; all others are deemed inconsistent. Only the structure points corresponding to consistent subtracks are included in the final structure.

This process forms the basis for building the keypoint database during the offline training stage. It also serves an important role in the online stage, as will be show in section 4.2.

4.1.2 Extracting Keypoint Descriptors

The point cloud, by itself, contains only geometric information, namely a 3D location for each structure point. To complete the keypoint database, each point must be associated with a visual descriptor that can be matched during the online stage.

For every inlier keypoint in every frame, a 32x32 pixel image patch is projected onto a 20-dimensional Walsh-Hadamard Kernel [6]. Existing work has shown that Walsh-Hadamard descriptors can be extracted quickly and are effective for keypoint matching [20, 9].

Tracking hundreds of keypoints over hundreds of frames, the resulting database can grow quite large (as many as 50,000 individual descriptors in our experiments). This is not a problem, in our experience, because the matching process will use an approximate nearest neighbor search that is, at worst, logarithmic in the size of the database (see section 4.2.1).

4.1.3 Adding Virtual Content

Once the database has been constructed, an application developer needs some means of adding virtual content. In our proposed interface, the user is presented with two side-by-side views. One, called the model window is a rendered version of the point cloud that can be freely rotated and translated to display the environment from varying viewpoints. The second view, called the image window is the input video
sequence itself, and can be advanced and rewound to show any individual frame.

Newly created objects can easily be rendered and manipulated in the model window. Because a camera pose has been recovered for each video frame, it can be rendered in image window as well. The image window thus offers a preview of how the object will look when it is ultimately displayed during the online stage.

In principle, the model window alone could suffice as an editing interface. However, the sparse nature of the model often provides only a rough outline of the scene structure. One of the principal goals of this work is to avoid creating a fully textured model. The model window can be viewed from any angle and is useful for the initial object placement.

The image window, on the other hand, only offers those viewpoints covered by the input video. Yet it provides a realistic view of the scene, and thus a valuable tool for fine tuning the object placement.

Figure 5 illustrates the advantages of the two view system. A virtual annotation is first positioned in the model window, but must be adjusted in the image window to achieve accurate positioning.

4.2. The Online Stage

The online stage will take a previously unseen video stream as input and, using the keypoint database for pose recovery, augment the scene with the annotations created during the offline stage.

4.2.1 Incremental Keypoint Matching

Each frame of the input sequence contains a set of keypoints. As in the offline stage, these keypoints are initially generated by a detector and tracked using optical flow, exactly as in the offline stage. Initially only the 2D image locations of these keypoints are known.

A descriptor is extracted at each keypoint using the same 20-dimensional kernel projections described in section 4.1.2, and matched against the database using a Euclidean nearest neighbor search. To improve performance, we use the approximate nearest neighbor algorithm described by Beis, et al. [1], called Best Bin First. It returns the exact nearest neighbor with high probability while only requiring time logarithmic in the database size.

Using the Best Bin First algorithm, the nearest and second nearest matches are retrieved. If both of these are associated with the same structure point or the ratio of their distances to the query point is smaller than 0.75, the match is accepted, otherwise it is rejected.

Even using the fast approximate nearest neighbor searches, the matching process is time consuming when applied to every keypoint in an image (typically 300-500). In order to spread this computational cost evenly across frames, we adopt the incremental keypoint matching approach described by Mooser, et al., [9]. While it was originally applied to planar rectangular objects, we show here that incremental matching can be applied to objects of arbitrary geometry as well.

Incremental keypoint matching only attempts to match a few keypoints at each frame so that the set of matches gradually accumulates. Generally, the set of matches grows large enough to recover a camera pose within about ten frames.

4.2.2 Camera Pose Estimation

Each successful database match produces a correspondence between a 2D image point and 3D structure point. RANSAC is used to fit a camera pose. Outliers are removed, as their associated 3D points are deemed incorrect.

Keypoints are tracked through the sequence so that poses can be recovered for subsequent frames. Some of these points will be lost as they drift out of view or due to track failures. At the same time however, incremental matching will add new matches at each frame, so that the set remains large enough to compute a reliable pose.

4.2.3 Incorporating Unmatched Keypoints

The process described thus far only uses points whose 3D locations are known in advance. Most of the keypoints in the image will never be successfully matched against the database. We will demonstrate that incorporating these points into the pose estimation process significantly improves the quality of the final results.

Once a pose has been recovered for several consecutive frames (20 in our implementation), unmatched keypoints can be localized in space by applying the same structure
from motion techniques used during the offline stage. Keypoint tracks are passed to the subtrack optimization algorithm, partitioned into subtracks, and identified as inlier or outlier. The inliers now have associated 3D points applied to the pose estimation of future frames. The result is two distinct sets of matches. Those that were originally matched to the database are called database matches. Those that were matched using subtrack optimization are called calculated matches.

The advantages of including calculated matches in the pose estimation process are twofold. First, there will be times that the camera comes into a view that was not represented in the model building phase, in which case the database matches alone will be insufficient to recover a pose.

The second advantage is that even when the database matches alone are sufficient to compute a pose, having more matches results in a smoother more reliable pose. Typically, there are many times more calculated matches than database matches, and a larger set makes the final pose estimation more reliable.

Both of these advantages are demonstrated numerically in section 5.

5. Results

We demonstrate the functionality of both the online and offline stages on three test cases. In all cases, one video sequence was captured for the offline stage and a separate, longer video was used for the online stage.

The first test, the Fuse Box sequence, shows the exterior of an electrical fuse box in an industrial environment. The scene contains a mixture of planar and non planar surfaces.

The second test, the A/C motor sequence, targets an irregularly shaped object in an industrial setting. Although the ground surrounding the motor is flat, it is mostly covered in gravel, and does not contain many easily identified textures.

Finally, the campus building sequence, shows the corner of a building on the USC campus, an environment containing both natural and man-made objects. The goal is to populate the scene with labels showing the way to nearby campus points of interest.

The primary focus of all of these tests is to show that the subtrack optimization algorithm makes a substantial, measurable difference in the end results. Both stages were thus run with and without the optimization.

Table 1. Offline Stage Error Measurements

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Fuse Box</th>
<th>A/C Motor</th>
<th>Buildings</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Opt.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Avg. Subtrack Len.</td>
<td>15.48</td>
<td>24.97</td>
<td>20.89</td>
</tr>
<tr>
<td>Reproj. Error</td>
<td>1.17</td>
<td>0.45</td>
<td>0.85</td>
</tr>
<tr>
<td>With Opt.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Avg. Subtrack Len.</td>
<td>21.36</td>
<td>26.23</td>
<td>25.28</td>
</tr>
<tr>
<td>Reproj. Error</td>
<td>0.64</td>
<td>0.37</td>
<td>0.54</td>
</tr>
</tbody>
</table>

Table 1 shows the RMS reprojection error of all keypoints in the database after running the offline stage. When no subtrack optimization is applied, keypoint tracks are never partitioned; tracks are simply terminated when their error exceeds δ. Every track will thus correspond to exactly one structure point. In all three cases the average error is significantly worse without optimization.

One might suspect that the reduction in total error was simply the result of creating shorter subtracks. If one test tends to generate subtracks that are much shorter, on average, than another test, then the first test will almost certainly return a smaller error. However, as shown in Table 1, the average subtrack length is actually longer when using subtrack optimization. It thus offers the dual advantages of producing subtracks that are longer (and thus span a wider baseline) yet more consistent in terms of reprojection error.

Figures 6 shows the point cloud models for building sequence along with the computed camera poses.

Moving to the online stage, table 2 shows the results of recognizing and tracking all three scenes. When testing the system without subtrack optimization, the pose refinement described in section 4.2.3 is completely eliminated; that is to say only database matches and no calculated matches are used. As with the offline stage, pose accuracy is measured by average reprojection error of all keypoints in all frames.

Table 2 also compares the average number of inlier keypoints available with and without subtrack optimization. As the results show, including calculated matches greatly increases the total number. Using a larger number of observations to estimate pose makes the computation more reliable.
and more robust to individual errors, as discussed in section 4.2.3.

Note that two of the three tests involved moving the camera to an area of the environment that had not been covered during the offline stage. The system, therefore, is unable to maintain database matches. Without the use of calculated matches, there is no way to recover a pose and the entire tracking process fails. Using the calculated matches, however, the system is able to continue for over 100 additional frames.

All tests were run on a 3.4 GHz Pentium 4 machine running Windows XP. Without incorporating calculated matches, the online stage processes each frame in approximately 100 ms. When using calculated matches, the processing time increased to approximately 500 ms. This trade-off between speed and accuracy is adjustable; as discussed in [10], the maximum subtrack length can be adjusted to meet the needs of a particular application.

Figure 7 shows the final augmentation results for all three test cases. Despite slight visible errors, the virtual objects clearly remain correctly aligned with the real world.

6. Conclusion

This paper has presented a complete system that allows scenes and objects of arbitrary geometry to be captured and later recognized, tracked, and augmented. Besides the ability to automatically create a keypoint database, the system also includes a means of manually creating and positioning virtual annotations.

Most importantly, we have shown that both the offline and online stages benefit greatly from robust structure from motion, specifically the subtrack optimization algorithm. During the offline stage it makes the model creation more accurate. During the online stage it makes pose recovery more reliable. It thus provides a substantial improvement to the final AR product.

Acknowledgements

This study was funded by the Center of Excellence for Research and Academic Training on Interactive Smart Oilfield Technologies (CiSoft); CiSoft is a joint University of Southern California-Chevron initiative.

References


Figure 7. The final AR output for all three test cases. In (a) the Fuse Box is tracked from a variety of orientations, not all of which were covered in the training process. In (b) the A/C motor is tracked through a nearly 180° rotation. Note that the keypoint dataset was built from only one side of the motor. The last example, (c), shows directions to various buildings and streets on the USC campus. In all cases the virtual objects remain accurately aligned with the image of the real world.