# 3D Feature Extraction from Uncalibrated Video Clips

Arturo Donate
Department of Computer Science
Florida State University
Tallahassee, FL 32306
donate@cs.fsu.edu

Xiuwen Liu
Department of Computer Science
Florida State University
Tallahassee, FL 32306
liux@cs.fsu.edu

## ABSTRACT

This paper explores the idea of extracting a dense 3D point cloud corresponding to salient features in a video. The goal is to generate the dense point cloud efficiently, in order to use the information in various other video processing tasks. We present a method that is capable of extracting 3D information of videos with no previous knowledge of the scene, while keeping computational costs low. Our method exploits the movement of the camera while robustly tracking features over time, in order to obtain multiple views of a scene and perform 3D reconstruction. Additionally, our system is able to cope with individually moving people seen in the videos, and can estimate each person's pose and fit a 3D model to it. This 3D model is inserted into the dense point cloud in order to visualize the reconstructed scenes, and does not affect the tracking of the rest of the scene.

## Categories and Subject Descriptors

I.4.8 [**Image Processing and Computer Vision**]: Scene Analysis—*tracking,motion, object recognition, depth cues*

## General Terms

Algorithms, Design, Performance

## Keywords

3d reconstruction, computer vision, mapping, structure from motion, tracking, video processing

## 1. INTRODUCTION

In recent years, the use of 3D in videos has shown a steady incline. Due to the inherent difficult in working with three dimensions, the use of 3D reconstructions and 3D features was not very prominent in the literature until recently. Nowadays, with the rise in computational power of most personal computers, as well as the increased interest in various 3D video applications, the use of three dimensional information is quickly becoming very prominent. The ultimate goal of this research is to extract meaningful 3D representations of a scene in order to improve the performance of various other video processing tasks (e.g., object detection and modeling, video retrieval, shot-boundary detection, etc).

In this paper, we concentrate on the problem of extracting meaningful 3D representations of scenes observed by a single camera. This problem presents many inherent difficulties since we are dealing with a single view (as opposed to a stereo view). Additionally, we assume no knowledge of the underlying camera system or observed scene. In order to achieve an accurate 3D representation, we must track features across frames and perform a temporal matching. In order for this to work correctly, most methods in the literature assume a static scene. In our system, we are able to cope with humans moving across the video by detecting their relative pose and incorporating this into our measurements.

The rest of the paper is organized as follows. In section 2, we will provide a review of recent works in the literature. In section 3, we give an analysis of the various feature extraction methods used. We describe the tracking and reconstruction algorithm in detail in sections 4. The human pose estimation is illustrated in section 5. We illustrate several results in section 6, and conclude the paper in section 7.

## 2. BACKGROUND

Extracting 3D information from 2D images and videos is one of the classic problems of computer vision. As such, we do not have the space to perform a complete literature review, and thus will concentrate on recently published methods related to the problem being addressed in this paper. Most of the works reviewed here fall into one of three categories: simultaneous localization and mapping (SLAM), structure from motion (SFM), and 3D from 2D. All of these problems have similar or related goals, and thus face many of the same problems and limitations.

In [12, 13], the authors describe a real-time structure from motion approach that is capable of reconstructing an observed scene in 3D using only a single camera as input. The method extracts Harris corners from the image and matches them across frames using normalized cross correlation. The authors use Grunert's pose estimation algorithm to retrieve the camera location at each frame in real-time. The 3D structure of the scene is estimated via triangulation. The method is capable of very accurate localization when mounted on a car, as described by their experiments [12].

In [9], the authors present a framework for augmented reality using a single monocular camera. The authors run two separate threads, one for tracking and one for mapping.


*3DVP'10,* October 29, 2010, Firenze, Italy.

The tracking process receives new frames from the camera, projects existing features onto the image according to a prior pose estimate, searches the image for the features using a coarse-to-fine approach, then computes an updated camera pose estimate. The mapping process is responsible for generating the 3D map and maintaining it. Initially, the 3D points are calculated using the 5-point algorithm [15]. Additionally, the system estimates the location of a flat surface in order to perform several augmented reality tasks using the 3D map.

In [4], the authors present the MonoSLAM framework, a system capable of localization and mapping using a single calibrated camera as input. This method finds salient features in the image and stores them in a probabilistic map that keeps track of the covariance between all features. In order to achieve real-time performance, the set of features is kept relatively sparse. The system uses Kalman filtering to estimate the camera and scene locations over time.

In [23], the authors introduce their VideoTrace system. This software package allows users to extract meaningful 3D models of objects from 2D videos. The method uses the Voodoo [21] software package to track the camera, then allows users to outline simple 3D shapes on the video frames. By tracking these shapes over time, the method uses a Levenberg-Marquardt optimization technique to minimize the distance between points and the estimated surfaces, in order to extract NURBS surfaces and generate 3D models of objects present in the scene.

# 3. FEATURE EXTRACTION AND MATCHING

The feature extraction step is one of the vital parts of our algorithm. Not only do they provide the necessary information to achieve robust tracking, matching and reconstruction, but they must also be descriptive enough so that each individual feature can be easily found in subsequent frames. This requirement can become challenging for many existing feature detectors/descriptors in the literature when dealing with image regions with repetitive textures, or videos with low-quality frames. One of the most powerful feature detectors available today is the SIFT detector [10]. This method calculates oriented histograms of features and extracts 128-dimensional histograms which are rotation and scale invariant. The only real downside to the SIFT detector is its computational time. Since we aim to keep computational cost down as much as possible, we decided to use SURF [1] features instead. They behave much like SIFT features, but make use of box filtering and integral images to keep computational costs down. Additionally, we use the 64-dimensional descriptor for performance reasons.

As is their nature, these SURF features tend to extract image patches in high gradient regions. In order to counter this, we also extract features using maximally stable extremal regions (MSER) [11]. These features tend to have more of a blob-like structure, providing different but complementary features to the SURF. We use the 64-element SURF descriptor to describe each MSER patch.

These two features combined provide excellent data for tracking, but alone would not provide a dense reconstruction of the scene. Hence, we also apply the FAST [17] corner detector on the frames. This feature detector is able to extract salient corners in images that are excellent candidates for matching. The computational cost of extracting these features is very low, making them excellent candidates for real-time applications. Figure 1 shows two sample video frames with all three feature types extracted.

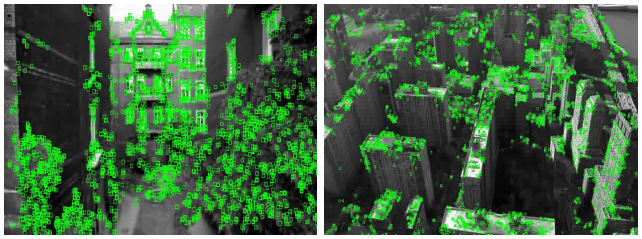

**Figure 1: Detected features on sample frames.**

For efficiency, we do not consider every frame of the video a key frame. During the initialization step, which will be discussed in more detail in section 4.1, we consider the two user-selected frames to be key frames and store these in the map. After this point, we only insert new key frames into the map if the number of tracked features falls below some threshold (in our experiments, our threshold was set to 1000 in order to generate a dense map). Matching each of these features must also be done in an efficient manner, since we will be keeping track of a large number of features across frames (for a 640x480 video, the total number of features may range between 2,000 to 10,000). For the SURF and MSER features, matching is done as described in [1]. For each feature, we find the 2 nearest neighbors of the same type and match the given feature to its closest neighbor if the distance between the two is at least 60 % closer than the distance between it and the second-closest neighbor. Otherwise, we discard the feature. Nearest neighbors are located quickly using the Fast Library for Approximate Nearest Neighbors (FLANN) [14].

For matching FAST features, we use normalized cross correlation (NCC) measurement to determine similarities. This NCC measurement is defined as:

$$NCC(f,g) = \frac{1}{n}\sum_{x,y}\frac{\left(f(x,y)-\bar{f}\right)\times\left(g(x,y)-\bar{g}\right)}{\sigma_f\sigma_g}, \qquad (1)$$

where $f, g$ are the two image regions containing the corners to be compared, $\bar{f}, \bar{g}$ their respective means, and $\sigma_f, \sigma_g$ their standard deviations.

# 4. TRACKING AND TRIANGULATION

This section provides an in-depth description of our algorithm. We will first describe the initialization and self-calibration steps performed at the beginning of each video. Afterwards, we will discuss the tracking and reconstruction steps, as well as human detection in the videos.

## 4.1 Initialization

The initialization of our system requires some human interaction. In order for our tracking to work properly, we need to establish an initial correspondence between two frames near the beginning of the video to estimate the initial structure of the scene as well as the extrinsic camera parameters. While the initialization can also be done automatically, this step is more error-prone and we chose to initialize semi-automatically.

To achieve this, we allow the user to manually select two frames from the video. Ideally, the user will select frames where there is a small translation of the camera. When the user selects the first frame, the system runs all three feature detectors on this initial frame. For the subsequent frames, the system runs all three feature detectors and attempts to match as many features across frames as possible. When the user specifies the second frame, the system first refines the match locally and uses the refined match points between frames to compute an estimate of the fundamental matrix using least squares as described in [8]. From the fundamental matrix, we can extract the camera position (relative to the initial position) and estimate the 3D location of each feature by using triangulation.

## 4.2 Self-calibration and Outlier Removal

Before the triangulation step occurs, the system determines the intrinsic camera parameters and removes incorrect correspondences caused by tracking errors. This self-calibration is done using the method proposed in [20]. Recall that a typical calibration matrix is defined in terms of focal length $f$, principal points $(u_0, v_0)$, and aspect ratio $\gamma$. The method assumes that the focal length is the only unknown, so we assign approximate values to the other parameters. Afterwards, we compute the semi-calibrated matrix $G$ using our previously calculated fundamental matrix $F$ and estimations of camera parameters as:

$$G = \begin{pmatrix} \gamma' & 0 & 0 \\ 0 & 1 & 0 \\ u_0' & v_0' & 1 \end{pmatrix} F \begin{pmatrix} \gamma & 0 & u_0 \\ 0 & 1 & v_0 \\ 0 & 0 & 1 \end{pmatrix}. \quad (2)$$

From the decomposition of this matrix $G$, we end up with two linear and one quadratic equations. For most cases, simply solving for the focal length in the quadratic equation provides us with the necessary calibration parameter. For further details on this method, including proofs, we refer the readers to [20]. This process yields an estimate for the intrinsic camera matrix that, although may not be exact, provide a solution that works well with our tracker.

Before the initial 3D reconstruction takes place, we attempt to remove any incorrect matches from our initial tracking estimate. To do so, recall that for any pair of non-normalized image coordinates, we have

$$(p')^T F p = 0, \quad (3)$$

where $p, p'$ are the image coordinates of the two image frames corresponding to the same feature. The system iterates through all features, computing this value for each matched pair. For any features where the value is not close to zero (in our experiments, we used a threshold of 0.001), the system will remove them from the set.

Afterwards, the system uses triangulation from the two camera locations (corresponding to the two video frames chosen by the user) to calculate the initial 3D location of each feature. These features are stored in our 3D map, and will be used for tracking the camera location at each frame.

## 4.3 Camera Pose Estimation and Tracking

Once the user has selected the initial frames and the system is initialized, the main tracking process can begin. The basic idea behind the tracking is simple: at each frame, project each of the 3D features stored in the map back onto the newly acquired image frame, then match the projected features with the newly detected features. Any of the previously detected features which fail to match are removed from the map. If at any point the number of features falls below a certain threshold, we insert a new keyframe into the system and find new features to add by matching new unmatched features from the latest frame with features detected from previous frames, and performing triangulation.

The system also updates the pose of the camera at each iteration. This is done by implementing Grunert's pose estimation algorithm [7]. This is a very popular algorithm used by several tracking and SLAM methods, capable of very accurate results [12, 13]. In order to remove ambiguities in the solutions, we incorporate the use of RANSAC [6] to find the estimated pose with the minimum number of outliers.

## 4.4 Optimizing Feature Locations

As new image frames are observed, the algorithm re-estimates the camera pose at each frame. If the number of features currently being tracked is above a certain threshold, no new keyframes need to be inserted into the system. In this case, we make use of bundle adjustment to refine the estimated 3D feature locations and camera-pose estimations across previous frames. This process uses a Levenberg-Marquardt optimization to obtain the optimal least-squares solution for the system.

Recall that this method is intended to work with video clips corresponding to a single shot of a scene. Typically, most shots in full-length movies tend to be relatively short. In this case we can safely perform bundle adjustment across all frames in the shot. However, we have observed that the majority of personal video clips available on online public databases (such as YouTube) can sometimes be quite long in duration, while still being composed of only one single shot. Since bundle adjustment can be a computationally expensive operation, we have decided to use a local bundle adjustment scheme that only takes the most recent 20 video frames into account. Our experiments show that reducing this number further yields a negative effect on the quality of the reconstructions, and thus not worth the reduction in computation time.

Let $\alpha_i$ be our cost function at frame $i$ to be minimized by the local bundle adjustment. Let $C_i$ be the set of camera locations for the most recent $N$ positions, and let $F_i$ be the set of feature locations visible at each of the $N$ previous camera locations. We can now define our cost function as

$$\alpha_i = \sum_{C_i} \sum_{f_k \in F_i} d\left(f_k, P_i f_k\right), \quad (4)$$

where $f_k$ is a feature in the set $F_i$, $P_i$ is the projection matrix for the camera position at the current $C_i$, and the term $d\left(f_k, P_i f_k\right)$ defines the Euclidean distance between the previously observed feature $f_i$, and the the projected feature $P_i f_k$ using the current projection matrix. In other words, we are using a least-squares optimization to minimize the re-projection error of feature and camera locations across the $N$ most recently observed video frames.

# 5. HUMAN POSE DETECTION

One of the largest limitations of the majority of SLAM and SFM approaches are their inabilities to correctly handle independently moving objects in the scene. As such, most methods assume a completely static scene with the only motion present corresponding to the camera motion. Our aim

is to allow the system to observe a moving person in the video, while successfully tracking and mapping the rest of the scene. Additionally, we place a generic 3D model of the human in the 3D scene in the same pose and relative location as the person being observed in the video. We divide this process into two steps; first we find persons in the videos and estimate their 2D pose, then we apply the 2D pose to our 3D human model and place the model in the 3D scene.

## 5.1 Detection and 2D Pose Estimation

The first step is to determine if a person is present in the scene. We can do this efficiently by running a face detector on selected frames. If a frontal or profile face is found, we assume there is a person in the scene. Next, we employ the use of the 2D human pose estimator proposed in [15, 16] to estimate the pose of the detected person.

In [15, 16], the authors present a framework for estimating the relative 2D pose of a person across a video sequence; they use a simple 2D model of a human composed of a head, torso, upper and lower arms, as well as upper and lower legs. The system first builds an appearance-based model of each person, then tracks each person by detecting the model in subsequent frames. In order to build the model, candidate parts (for each part of the model) are first found using an edge-based part detector. These candidate parts are then clustered via the mean-shift algorithm in order to find patches with similar appearances across time.

Afterwards, the authors enforce a predefined motion model on the resulting candidate parts. Clusters that are smaller than a certain size, or that never move, are removed (effectively rendering the method useless for tracking people who stand still across the entire video sequence). Each torso cluster is considered a unique person. Once the torso is estimated, the rest of the limbs are inferred according to their model. For further details on this method, we refer the reader to [15, 16]. Figure 2 illustrates a pose estimate calculated using this method. Once complete, we generate a bounding box around the pose estimate, and any previously-observed feature whose 2D projection falls within this box is omitted from the camera pose calculations and bundle adjustment optimizations. Additionally, any new features detected within this bounding box are discarded right away. This prevents the system from using points on a moving person during the camera tracking.

## 5.2 3D Pose Estimation

At this point in the algorithm, the system has successfully detected the human(s) in the videos and estimated their pose in 2D. The next step is to extend this pose to 3D, then place a model of the person in our 3D reconstruction. We employ the use of a simplified 3D human model with similar characteristics to the previously obtained 2D output.

This 3D model consists of several block segments, each one matching a segment of the 2D model. It has a segment for the head, torso, upper and lower arms, as well as upper and lower legs. Now the problem becomes extending the 2D pose onto the 3D model. Note that this is an ill-posed problem, as there may exists more than one 3D pose that project onto the same 2D pose, making the job of going from 2D to 3D inherently more difficult. As such, we apply several constraints to our 3D model.

First, the overall model can only be rotated about the Y axis, changing the direction the model is facing. Each

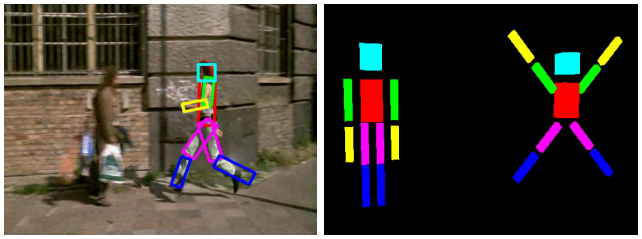

**Figure 2: Left: Estimated 2D pose. Right: Our 3D model in various poses.**

upper arm section has an almost free range of motion, but cannot bend backwards towards the rear of the body, or be rotated so that it collides with the torso or the head. The lower arm is more restricted, only allowing 90 degrees of freedom to bend at the elbow joint. The upper leg segments can be rotated up to 45 degrees forwards, backwards, or away from each other. The legs are not allowed to bend inwards (making it impossible for our model to cross its legs correctly). The lower leg segments have 135 degrees of freedom at the knee. Both the knee and elbow joints are also constrained so that the bending occurs in one direction only (i.e., if the arm is hanging straight down, the elbow can only be bent in such a way so that the lower arm is rotated towards the direction that the model is facing, and not towards the rear; similar constraints are applied to the knee to simulate typical human limitations). Figure 2 shows our 3D model in two possible poses.

First, we must estimate the relative direction of the 3D model, which will be refined later. Since we are limited to a rotation about the Y axis, we need to figure out which direction the model is facing. To do this, we look at the position of the 2D model over time. We assume that the person is always facing the direction in which they are walking. Therefore, if the 2D pose estimation retrieves a person moving in the negative X direction, our initial 3D model orientation is to rotate the model so that it is facing this same direction. This implies that the system is not able to accurately model people walking backwards, which is not common anyways. Note that since this orientation is based on 2D image measurements, the initial orientation of the model will most likely be perpendicular to the optical axis of the camera if the person is moving. If the person is standing still and the 2D pose was able to extract two arms and two legs side by side, we assume the person is facing the camera.

Next, we need to align the sections of our 3D model with the 2D pose estimation. The first step is to calculate the relative angle of the torso (in relation to the image's Y axis). This angle will give us the amount of rotation to be applied to our 3D model's torso. From this, we calculate the relative angle between the torso and each upper arm segment in the 2D pose estimate, and apply them to the 3D model. Next, we calculate the angle between the upper and lower arm segments, and rotate the 3D model's lower arm accordingly. A similar process is repeated for the upper and lower leg segments.

At this point we have aligned our 3D model with the 2D pose estimation calculated from the video frames. The final step is to find the correct placement of the 3D model in

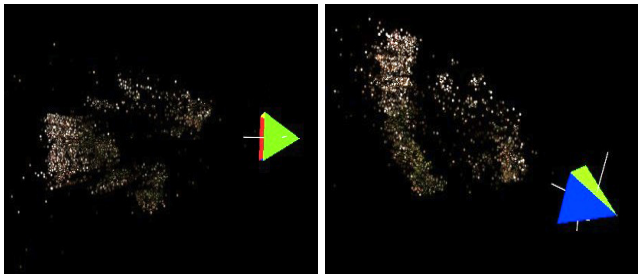

**Figure 3: Reconstruction from various viewpoints.**

**Figure 4: Tracked 2D features and 3D reconstruction.**

the reconstructed 3D scene. Recall from Section 2 that our 3D reconstruction is only able to triangulate static points in the scene. Because of this, the 3D reconstruction cannot accurately recover any points on the moving person to determine their location. Instead, we attempt to place the feet of our 3D model in the 3D scene. We begin with the 2D pose estimation, and attempt to find the image feature with the shortest euclidean distance to the bottom of one of the lower leg segments. When this point has been found, we place our 3D model in the reconstructed scene so that the lower leg segment has the same 3D coordinate as the image feature selected. If more than one feature is found to have the overall shortest distance, the feature is selected arbitrarily.

The final step is to refine the orientation of our 3D model. Recall that the initial orientation is based on the direction of the 2D pose estimate. After obtaining an estimated 3D location for our model across video frames, we refine the orientation by adjusting it to be equal to the direction of the path the 3D model is traveling in (equivalent to the direction between the features closest to our model's feet over time).

## 6. RESULTS

Here we illustrate various reconstruction results generated from major motion pictures, as well as personal hand-held cameras. The videos contain several compression artifacts. The average performance of our algorithm ranged between 5 and 8 frames per second when processing these videos in resolution of 640x480 on a 3.0 GHz quad core processors using two threads to handle the workload, as long as there is no human present in the scene. When a human is located, the pose estimation slows down the system to several seconds per frame; as quad-core processors are commonly available, this performance can be improved substantially using additional threads.

The first set of results in Figure 3 shows the 3D reconstruction of the sample frame from the left image of Figure 1. This reconstruction, obtained from the feature film "Run Lola, Run" shows our method's ability to extract excellent 3D structure from the video. The building in the background is clearly visible from all angles, and the depth difference between the foreground trees and the background building is clearly visible.

The second set of results in Figure 4 correspond to the video frame in the right image of Figure 1. This reconstruction was successful in obtaining a relatively dense amount of points on each building, and even extracting the structure of the skyscrapers on the right hand side of the frame. The structure of the buildings in the middle of the frame is not completely visible in the reconstruction since it was not clearly visible in the video, and hence no features were found in this area.

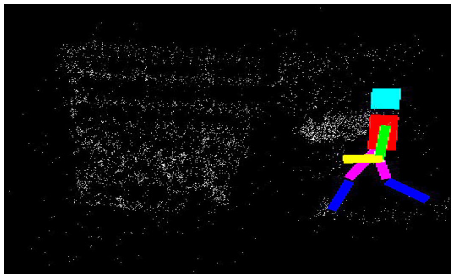

**Figure 5: Reconstruction with 3D human model.**

This third set of results in Figure 5 show the 3D reconstruction of the video frame illustrated in the left image of Figure 2. Notice that not only was the system capable of tracking and reconstructing the scene, but it was also capable of tracking and fitting a 3D model of the human, while not losing the ability to continue tracking the background features.

## 7. CONCLUSION

As video databases become more prevalent, the need for powerful video analysis and mining techniques increases. Many tasks such as shot-boundary detection, video summarization, and content-based video retrieval rely on measurements and techniques based on two-dimensional image features. Although some of these tools can be very powerful, they are not capable of capturing the inherent three dimensional structure of a scene. The goal of our research presented is to estimate the three dimensional structure of a scene, in the hopes of facilitating and improving various video analysis tasks.

As discussed in section 2, there have been several works published recently which deal with the estimation of 3D structure of a scene observed from a monocular camera, while assuming no previous knowledge of the scene or camera. We show that under camera motion, it is possible to estimate the location of the camera in the world, and from this estimate the 3D structure of the scene. We employ the use of accurate and efficient feature detectors to build a dense

3D reconstruction of a scene, while keeping computational time low.

We also show that although most of the related methods require observation of a static scene, it is possible for our system to continuously track scenes containing people. We first detect the people and estimate their relative pose in 2D. This pose is then applied to a 3D human model, which is then inserted into the 3D scene. In order to perform accurate tracking with the person in the video, we place a bounding box around the person and do not use any points within the bounding box in the tracking measurements.

Our experiments show that this method has great promise, but much work remains to be done before the technique can be truly useful. We are currently working on improving the human pose estimation, which is clearly the bottleneck of our system. Additionally, we are experimenting with various approaches to speed up the feature detection and matching, in order to further reduce computation time on videos. Our future research involves applying several techniques to speed up the computation (especially the human pose estimation), as well as incorporating this technique into various video analysis tasks.

### *Acknowledgements*

We would like to extend our gratitude to the authors of the Parallel Tracking and Mapping framework [9], as well as the authors of the MonoSLAM framework [4] for providing their source code online for other researchers. Their fantastic work on these methods has proved invaluable to the development of our own algorithms.

## 8. REFERENCES

[1] H. Bay, T. Tuytelaars, and L. J. V. Gool. Surf: Speeded up robust features. In *European Conference on Computer Vision*, pages 404–417, 2006.

[2] R. O. Castle, D. J. Gawley, G. Klein, and D. W. Murray. Towards simultaneous recognition, localization and mapping for hand-held and wearable cameras. In *Proc. International Conference on Robotics and Automation, Rome, Italy, April 10-14, 2007*, pages 4102–4107, 2007.

[3] J. Civera, A. J. Davison, and J. Montiel. Inverse depth to depth conversion for monocular slam. In *IEEE International Conference on Robotics and Automation*, April 2007.

[4] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse. MonoSLAM: Real-time single camera SLAM. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(6):1052–1067, 2007.

[5] A. Donate and X. Liu. 3d structure estimation from monocular video clips. In *First International Workshop on Three Dimensional Information Extraction for Video Analysis and Mining*, 2010.

[6] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[7] R. Haralick, C. nan Lee, K. Ottenberg, and M. Nolle. Analysis and solutions of the three point perspective pose estimation problem. *International Journal of Computer Vision*, pages 592–598, 1991.

[8] R. Hartley and A. Zisserman. *Multiple View Geometry*. Cambridge Press, 2003.

[9] G. Klein and D. W. Murray. Parallel tracking and mapping for small ar workspaces. In *International Symposium on Mixed Augmented Reality*, 2007.

[10] D. G. Lowe. Object recognition from local scale-invariant features. In *International Conference on Computer Vision*, pages 1150–1157, 1999.

[11] J. Matas, O. Chum, M. Urban, and T. Pajdla. Robust wide baseline stereo from maximally stable extremal regions. In *British Machine Vision Conference*, pages 384–393, 2002.

[12] E. Mouragnon, M. Lhuillier, M. Dhome, F. Dekeyser, and P. Sayd. Real time localization and 3d reconstruction. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 363–370, Washington, DC, USA, 2006. IEEE Computer Society.

[13] E. Mouragnon, M. Lhuillier, M. Dhome, F. Dekeyser, and P. Sayd. Generic and real-time structure from motion using local bundle adjustment. *Image and Vision Computing*, 2008.

[14] M. Muja and D. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Applications*, pages 331–340, 2009.

[15] D. Ramanan, D. A. Forsyth, and A. Zisserman. Strike a pose: Tracking people by finding stylized poses. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 271–278, 2005.

[16] D. Ramanan, D. A. Forsyth, and A. Zisserman. Tracking people by learning their appearance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(1):65–81, January 2007.

[17] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In *European Conference on Computer Vision*, pages 430–443, 2006.

[18] M. Schwarzkopf and C. Richardt. Proteus - semi-automatic interactive structure from motion. In *Vision, Modeling, and Visualization Workshop*, 2009.

[19] J. Sivic and A. Zisserman. Video google: Efficient visual search of videos. In J. Ponce, M. Hebert, C. Schmid, and A. Zisserman, editors, *Toward Category-Level Object Recognition*, volume 4170 of *LNCS*, pages 127–144. Springer, 2006.

[20] P. Sturm. On focal length calibration from two views. In *International Conference on Computer Vision and Pattern Recognition*, pages 145–150, 2001.

[21] T. Thormahlen. *Zuverlassige Schatzung der Kamerabewegung aus einer Bildfolge*. PhD thesis, University of Hannover, 2006.

[22] P. Torr, A. Fitzgibbon, and A. Zisserman. Maintaining multiple motion model hypotheses over many views to recover matching and structure. In *International Conference on Computer Vision*, pages 485–491, 1998.

[23] A. van den Hengel, A. Dick, T. Thormahlen, B. Ward, and P. Torr. Videotrace: Rapid interactive scene modelling from video. *ACM Transactions on Graphics*, 26(3):86, 2007.