Shot Boundary Detection in Videos Using Robust Three-Dimensional Tracking

Arturo Donate and Xiuwen Liu
Florida State University
Department of Computer Science
Tallahassee, FL 32306
{donate,liux}@cs.fsu.edu

Abstract

The use of three dimensional information from video is rare in the video analysis literature due to the inherent difficulties of extracting accurate 3D measurements from a single view of a scene. Several methods have been published in recent years, however, that attempt to solve such a problem. They all use the same underlying meaning of exploiting camera motion in order to measure the parallax of visible objects in the scene.

In this paper, we employ the use of such algorithms towards solving the problem of automatic shot boundary detection. The idea is to extract salient features from a video sequence and track them over time in order to estimate shot boundaries within the video. We apply many ideas from previously published SLAM techniques in order to model the inherent three dimensional structure of a scene, and accurately track various salient features across frames. We detect shot boundaries in videos by observing the system's ability to successfully track features across frames.

1. Introduction

With the rapid increase of video databases nowadays, the need for video analysis methods grows every day. One of the most fundamental problems in video analysis is creating hierarchical representations of videos. Such representations start with the actual video at the highest level. The video may be broken down into scenes, each containing some semantic meaning. Scenes can be broken down further into shots, which may or may not contain semantic meaning. These shots can then be broken down even further into the individual video frames that make up the video. In this paper, we concentrate on the specific problem of automatically detecting shot boundaries in videos.

Several works have attempted to solve this problem using various low-level image measurements. In [4], the authors present a method for shot boundary detection which relies on pixel intensities, color histograms in HSV space, as well as edge histograms. These measurements are vectorized and used to train a support vector machine (SVM). The SVM is used to classify the frames into several classes, corresponding to one of four transitions. These transitions may include smooth changes between frames, or abrupt ones. A detection algorithm is finally used on the resulting classifications to locate shot boundaries.

In [2], the authors measure the similarity between frames in order to detect shot boundaries. The authors define a similarity measurement between frames by first de-correlating RGB color information by transforming the image into the Ohta color space. Afterwards, the discrete cosine transform of each channel is computed and aggregated into a feature vector that represents the given frame. Frames are compared using the cosine distance measure. The authors state that shot boundaries will have high self similarities between local frames, and low similarities between frames further away.

In [5], the authors propose an approach for shot boundary detection that models temporal statistics. The method relies of eigenspace models of frames in order to find shot boundaries. An eigenspace model is first trained using histograms of the previous N frames. When a new frame is observed, the histogram of the current frame is first computed, then projected against the existing eigenspace to determine their similarity. If their distance is greater than some threshold, the frame is considered as being part of a new shot.

In this paper, we propose a method of detecting shot boundaries that relies on a robust tracking of salient features in the scene. By using existing simultaneous localization and mapping (SLAM), we are able to track objects in a scene by modeling the relative 3D positions of the features as well as the camera. In doing so, we rely on the notion that certain aspects of an image may change within a shot, but if we are unable to find any features to track between two frames, we must be observing a new shot boundary.

The rest of the paper is organized as follows. Section 2 defines the shot boundary problem, and identifies certain

978-1-4244-7030-3/10/$26.00 ©2010 IEEE

inherent difficulties that must be solved in order to accurately solve this problem. Section 3 provides a background to selected SLAM algorithms, and describes our proposed SLAM-based approach for detecting shot boundaries. Several of our results are presented in Section 4 to show the feasibility of our proposed approach. We conclude the paper in Section 5 and provide a summary of our work as well as future research directions.

2. Shot Boundaries

Most videos can be decomposed according to a hierarchical structure. This structure begins at the highest level with the actual video. The video can then be decomposed into different scenes in a way such that each scene contains semantically related content. According to Xiong et al. [10], scenes typically convey some high level concept and are divided using semantic boundaries. Each scene can then be broken down into video shots, and each shot can be decomposed into image frames.

Shot boundary detection is the problem of automatically detecting the point in a video where one shot ends and the next begins. Several types of shot boundaries exist. In this work, we deal with abrupt transitions (the boundary between two shots is sudden and immediate) as well as fading transitions (one shot fades into the other, sometimes transitioning regions of the image plane before others).

As previously mentioned, this problem is traditionally solved using various 2D image measurements. Such measurements do not capture the inherent three dimensional structure of the scene, and thus are somewhat limited (although still very useful). Our approach is to detect boundaries by robustly tracking certain objects present in the scene. In order to achieve better results, our tracking approach models the relative 3D structure of the scene by using SLAM techniques.

More specifically, we employ the use of the MonoSLAM framework originally proposed by Davison et al. [3]. With minor changes to the MonoSLAM method, we are able to accurately detect shot boundaries of several types with very few false positives.

3. Boundary Detection Using SLAM

In recent years, many methods have been published to extract 3D information from a scene as observed from a single camera. Such techniques include various approaches using SLAM, structure from motion, and 3D stereo techniques. Among these, MonoSLAM [3] has been recognized as a very powerful tool to perform localization and mapping from a single camera.

With some modifications to the original method, we use the MonoSLAM method to perform robust tracking of scene elements in a previously unseen video, in order to detect shot boundaries. The remainder of this section will provide an overview of the originally proposed MonoSLAM algorithm, as well as a discussion of our proposed modifications.

3.1. MonoSLAM

One of the most popular techniques proposed recently for localization using a single camera is the MonoSLAM [3] framework. The method presents an efficient approach capable of localizing a single monocular camera and simultaneously estimating the relative 3D structure of the environment seen by the camera, all with real time performance. The authors model the world using a probabilistic 3D map which includes the current state of the camera, the features being tracked, as well as the uncertainty of the estimates. After initial startup, the map is updated using the Extended Kalman Filter (EKF) [8]. We give an overview of the MonoSLAM framework in this section, but refer the reader to [3, 1] for more specific details on this method.

The map itself is composed of a state vector $\hat{x}$ and a covariance matrix P. The state vector $\hat{x}$ provides an estimate of the camera and visual features being tracked in the world. This vector is composed of the camera vector $\hat{x_v}$ and a feature vector $\hat{y_i}$ for each feature inserted into the map. The covariance matrix P is a square matrix that can be divided into individual sub-matrix elements, allowing the probability distribution to be approximated by a single multivariate Gaussian distribution. This state vector and covariance matrix can be written as

$$\hat{x} = \begin{pmatrix} \hat{x}_v \\ \hat{y}_1 \\ \hat{y}_2 \\ \vdots \end{pmatrix}, \quad P = \begin{bmatrix} P_{xx} & P_{xy_1} & P_{xy_2} & \dots \\ P_{y_1x} & P_{y_1y_1} & P_{y_1y_2} & \dots \\ P_{y_2x} & P_{y_2y_1} & P_{y_2y_2} & \dots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}. \quad (1)$$

Note from Equation 1 that the covariance matrix P is a full covariance matrix that models the relationship between features in the probabilistic map. This allows MonoSLAM to accurately localize the camera and compute the 3D locations of each feature relative to each other very accurately.

As mentioned above, the state vector is composed of the camera and feature vectors, which keep track of the main elements in the scene. The camera vector $\hat{x_v}$ keeps track of the extrinsic camera parameters , while each feature vector $\hat{y_i}$ stores the 3D location and orientation of each feature being tracked. These vectors can be written as:

$$\hat{x}_v = \begin{pmatrix} r^{WC} \\ q^{WC} \\ v^W \\ \omega^W \end{pmatrix}, \quad y_i = \begin{pmatrix} x_i \\ y_i \\ z_i \\ \theta_i \\ \phi_i \\ \rho_i \end{pmatrix}. \quad (2)$$

Here, the r^{WC} parameters stores the 3D location of the camera, q^{WC} stores the relative orientation of the camera, while the v^W and ω^W store the camera's velocity and angular velocity respectively. For each feature vector $\hat{y}_i$, the first three terms (x_i, y_i, z_i) define the 3D location of the camera's optical center at the time the feature was first observed, (θ_i, ϕ_i) is the azimuth and elevation for the ray $\mathbf{m}(\theta_i, \phi_i)$ from the camera to the observed feature, and ρ_i is the inverse depth $(\frac{1}{d_i})$ of the feature along this ray. These parameters keep track of a 3D point located at:

$$\begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} + \frac{1}{\rho_i}\mathbf{m}\left(\theta_i, \phi_i\right). \tag{3}$$

This basic scene geometry is illustrated in Figure 1.

Image features are initially found in the image by searching for salient image regions using the Shi and Tomasi operator [9]. A window of 11×11 pixels is used to extract an image patch at the given location. The patch is assumed to be planar and the surface normal is initially set to be equal to the optical axis of the camera (to be updated later). This image patch is projected as an image and saved as a template for matching. When a new view of this feature occurs during the main execution loop, the new visible image is warped according to the camera position, then compared to this template.

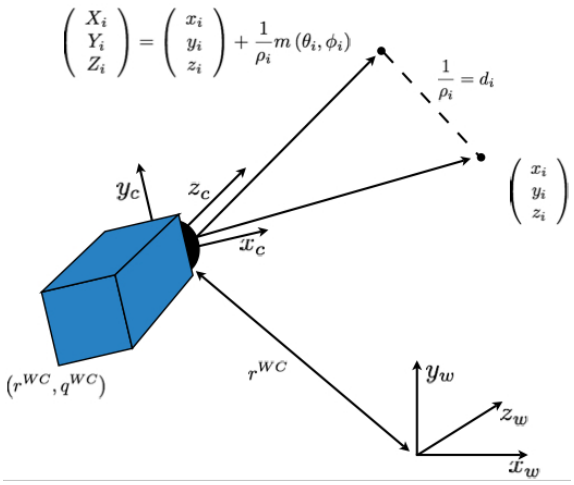

Figure 1. Basic geometry of the scene, for details see [6].

When a feature is first initialized, it is immediately inserted into the map with a large possible depth range of $[1, \infty]$, coded as a Gaussian [6]. As the feature is re-observed over time, the EKF re-estimates the depth of each feature until it converges to a more accurate depth value. For this to occur, there must be enough parallax observed by the camera; otherwise, the system assumes the features are at infinity.

In order to model the movement of the camera, the authors propose a "constant velocity, constant angular velocity model" [3] which assumes the camera motion accelerations occur with a Gaussian profile. The camera vector is updated at each iteration via the EKF according to this model.

3.2. Shot boundary detection via SLAM

As described in the previous section, MonoSLAM uses as input a video stream from a single monocular camera. This data is no different from the data available in a single video of an observed scene, and thus the MonoSLAM algorithm can technically be applied to videos as well. The main idea of our approach is to use this framework to track objects in the scene. If at any point the algorithm fails to track all of the previously-observed objects in the scene, we assume that a shot boundary has been detected.

The primary reason for choosing MonoSLAM for this problem is its ability to successfully localize a monocular camera in a three dimensional environment. Although we are not interested in camera localization for this application, the reason for choosing this method is its ability to accurately track a static scene accurately and robustly. It is important to note that MonoSLAM as originally proposed by Davison et al. [3] cannot be directly applied to this problem successfully.

Our approach begins by finding 16 individual salient image regions and initializing them into the probabilistic map. In order to achieve real time SLAM performance, the authors in [3] limit the number of tracked features to 8. Additionally, they limit the size of each feature to 11×11 pixels. In our proposed approach, we relax the real time criteria in order to improve tracking performance. In this work, we increase the number of tracked regions to 16, as well as increasing the individual region size to 17×17 pixels for a video with 640×480 resolution.

The search for features is limited to the inner rectangle of the image plane. On a 640×480 video, for example, we only search for features within the inner 620×400 pixels. The reason for this is to attempt and evade common locations for logos and text present in videos that sometimes span across shot boundaries. If we take the entire image plane into consideration, the feature detector will attempt to locate and track the text located on the screen. Examples of such a video can be seen in the screen capture illustrated in Figure 2. Note that the upper left corner of the screen contains a logo which is present across several shot boundaries. If our method was to successfully track features at this location, it would fail to detect shot boundaries.

At each iteration, the system updates the camera parameters as well as the feature vectors for all successfully tracked features. If the system fails to successfully track a given feature at any point in time, that feature is marked as one to be possibly deleted. If the same feature is unsuccessfully

tracked for 15 consecutive frames, it is removed from the map and deleted from the system. If at any point the algorithm marks all of the visible features to be possibly deleted, then we assume a shot boundary has occurred.

This process for feature deletion is vital to the performance of our proposed method for several reasons. First and foremost, it drives the detection of the shot boundaries. The underlying assumption is that if the scene has not changed, we should be able to at least track a single object across frames. If we are not able to track a single feature across frames, it is likely that we are observing a new shot.

Secondly, this method takes into account independently moving objects. One of the major drawbacks of MonoSLAM is its inability to track objects moving independently from the camera. Such objects are difficult to track in the probabilistic map because MonoSLAM relies on camera motion to exploit parallax and converge on a depth estimate for a feature. But if the feature is moving, the depth estimates related to camera motion become unreliable. If one of the features being tracked is not part of the static scene, the algorithm will try to locate it for 15 frames and remove it afterwards, so that the feature in question does not corrupt the camera location estimate, or the tracking of the other features.

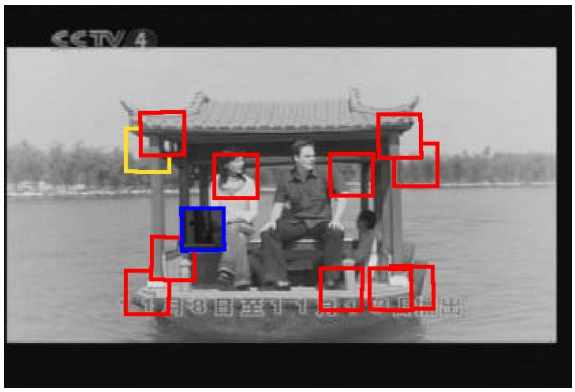

Figure 2. Tracking features in a video. Red features are successfully being tracked, blue features are marked for possible deletion, and yellow features are being initialized.

Finally, this approach also accounts for features being occluded for a short amount of time. Ideally we would like to track static features in the scene. Due to the dynamics of most videos, even static features may be occluded (by either a change in camera angle, or other dynamic objects located between the feature an the camera). If the object is successfully observed and tracked after being occluded, it is no longer considered as one to be possibly deleted until the system fails to track it once again. Note that in order for the system to handle such occlusions, the features must not be occluded for longer than the threshold value for deleting frames (15 frames).

Whenever a feature is completely removed from the probabilistic map, a new feature is immediately initialized. Its location depends on the results of the Shi and Tomasi feature detector [9]. Whenever a successfully tracked feature goes out of the bounds of the frame, it is also immediately deleted and a new one is initialized in order to keep the current number of observed features at 16. This places all emphasis on tracking the currently observed scene. Figure 2 illustrates this process. Here, the red squares denote features being successfully tracked across frames. The yellow squares are features that were just inserted into the map at that iteration. The blue squares are features that the system has failed to track successfully, but have not yet been deleted.

This threshold value of 15 frames was determined empirically using our test data. We ran several experiments on various videos, and determined that the threshold value of 15 provided the best shot boundary results for videos recorded at 30 frames per second. Videos of different speeds may require a different threshold value. The other values regarding the number of features and patch size were also determined empirically. We chose the combination that yielded the best results with the current dataset being tested.

As previously mentioned, the frames-per-second performance of our proposed approach is slower than the original MonoSLAM method proposed due to the increase in data being considered. Namely, increasing patch size from 11×11 to 17×17 pixels, as well as increasing the number of tracked features from 8 to 16. On an Intel Code 2 Duo 2.8 MHz CPU, our method runs at approximately 3 frames per second, only using a single core of the processor.

4. Results

This section presents several of our current results on various video clips obtained from the TRECVID 2005 [7] database. This database contains several challenging videos typically used to test the performance of shot boundary detection algorithms. We illustrate our results on several different videos to illustrate the performance of our proposed approach.

The first example illustrates a scene with a relatively small amount of independent motions. In this scene, we see two people sitting on a boat and talking. Most of the objects in this scene are stationary and do not move independently. The only moving objects are the background (the boat has a small amount of movement from left to right), and the people's heads as they look side to side. The transition between one shot to the other is not an abrupt one. Instead, the screen slowly fades from right to left to denote the change in shots. Figure 3 illustrates four sample video frames from this example.

Notice that in the first frame (top left image of Figure 3), the system is successfully tracking several features in the

Figure 3. Four frames making up a diagonal wipe transition from the lower right of the frame to the upper left. Video was obtained from the TRECVID 2005 database [7].

image. As the transition between frames begins (top right image), the features are no longer successfully tracked and thus are marked for deletion. By the time the transition is almost complete (bottom left image), only a few features remain as successfully being tracked. By the time the transition is complete and the next shot has begun, all of the features have been marked for deletion and thus the system assumes a shot boundary has been found.

The next example (illustrated in Figure 4) shows two people again, but this time one of them is moving around the screen while waving his arms and torso around, making this scene considerably more difficult to track. As before, the transition between frames is a smooth diagonal fade from the bottom right of the screen to the top left.

The first two rows of Figure 4 illustrate the tracking results at four different points in time. The bottom left frame shows the scene after the diagonal fade transition has begun. The bottom right frame illustrates the beginning of the new shot. As in the previous example, our method is successful in accurately detecting the boundary between the two shots.

5. Conclusion

Most videos can be decomposed into a hierarchical structure based on their content. At the highest level is the actual video, which can be broken down into scenes that convey some relatively high level concept [10]. These scenes are divided using semantic boundaries. It is possible to further break down each scene into video shots. Shots may be viewed as small independent video clips which may or may not contain a semantic meaning. Each shot is composed of individual image frames.

Many video analysis tasks require the automatic detection of shot boundaries within a video. In this paper, we present a novel approach towards performing accurate shot

Figure 4. Six frames illustrating the tracking performance on dynamic scenes. Video was obtained from the TRECVID 2005 database [7].

boundary detection. We model the problem as one of tracking individual scene objects. While at least one part of the video can be matched to previous frames, we assume that the corresponding frames belong to the same shot. If at any point the system is unable to track any features between two consecutive frames, we assume that a shot boundary is being observed.

While the literature is saturated with various methods to perform object and feature tracking, we show that SLAM algorithms provide a powerful solution to object tracking. More specifically, we employ the use of the MonoSLAM algorithm. By modeling the scene with a probabilistic 3D map, we are able to track objects in the scene while estimating their relative 3D position. By using the inherent three dimensional structure of the observed scene, we can track different objects in the scene accurately.

Although our experiments show the feasibility of our proposed approach, there is some future work to be done in order to overcome some of the limitations of the system. The main limitation currently is the system's inability to track independently moving objects. If the system only chooses to track features on moving objects, these features will eventually fail tracking and be deleted from the system. If all the features fail at times relatively close to each other, the system may falsely detect a shot boundary. The second limitation is the relatively inefficient performance of such a system. By increasing the size of each feature and the number of features tracked by the algorithm (from those of

the original MonoSLAM method), our proposed approach is slowed down considerably. More research is needed in order to speed the computations up.

Acknowledgements

This work is partially funded by NSF grants CCF-0514743 and DMS-0713012, and the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 RR021813. Information on the National Centers for Biomedical Computing can be obtained from http://nihroadmap.nih.gov/bioinformatics.

References

[1] 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.

[2] M. Cooper, J. Foote, J. Adcock, and S. Casi. Shot boundary detection via similarity analysis. In *Proceedings of the TRECVID 2003 Workshop*, pages 79–84, 2003.

[3] 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.

[4] X. Ling, L. Chao, L. Huan, and X. Zhang. A general method for shot boundary detection. In *nternational Conference on Multimedia and Ubiquitous Engineering*, pages 394–397, 2008.

[5] X. Liu and T. Chen. Shot boundary detection using temporal statistics modeling. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 13–17, 2002.

[6] J. M. M. Montiel, J. Civera, and A. Davison. Unified inverse depth parametrization for monocular slam. In *Proceedings of Robotics: Science and Systems*, August 2006.

[7] P. Over, T. Ianeva, W. Kraaij, and A. F. Smeaton. Trecvid 2005 - an overview. In *TREC Video Retrieval Evaluation Online Proceedings*, 2006.

[8] M. I. Ribeiro. Kalman and extended kalman filters: Concept, derivation and properties, February 2004.

[9] J. Shi and C. Tomasi. Good features to track. In *International Conference on Computer Vision and Pattern Recognition*, pages 593–600. Springer, 1994.

[10] Z. Xiong, R. Radharkishnan, A. Divakaran, Y. Rui, and T. S. Huang. *A Unified Framework for Video Summarization, Browsing and Retrieval*. Elsevier, 2006.