Tracking Multiple Moving Objects from an Autonomous Helicopter

David Lieb, Andrew Lookingbill, David Stavens, Sebastian Thrun

{dflieb,apml,dstavens,thrun}@stanford.edu

Stanford Artificial Intelligence Laboratory

Stanford University

Stanford, CA 94305

The algorithm proposed in this paper addresses the problem of tracking multiple moving objects from a moving observation point. The goal was to implement a general approach which attains good results while using a simple transformation to model camera egomotion and no prior assumptions about the nature of the moving objects (size, speed, or visual appearance). The key to this approach is using a particle filter to represent multi-modal probability distributions and a grouping algorithm to interpret the results.

1 Introduction

Tracking multiple moving objects in video has a variety of real world applications. These include autonomous aerial reconnaissance, remote surveillance, and advanced real-time collision avoidance systems.

The problem is difficult in part because the algorithm must compensate for the motion of the camera, *egomotion*, before it can determine which elements of the image are, in fact, moving. After parts of the image have been tagged as moving, a grouping operation must be used to decide which image sections belong to a particular moving object and to calculate that object's position and velocity.

While methods currently exist to track moving objects in video taken from a moving observer, a specific gap has yet to be filled: a completely general algorithm that is robust in a variety of environments. As will be detailed in Section 2, our goal was to produce an algorithm that did not use color, texture, contrast, shape, or size information to track moving objects while keeping the details of the implementation simple.

This project combines feature-tracking, motion compensation, particle filters, and a cognitive algorithm in a motion tracking application able to deduce the number of moving objects in a frame and track them with a high degree of accuracy.

2 Background/Related Work

Previous work on this topic can be broken down into the same sub-problems that the current algorithm addresses: calculation of overall scene motion, egomotion compensation, and some combination of tracking and grouping moving objects.

The majority of current work in image stabilization uses a sparse set of features to calculate the motion of various scene points [1,2]. Some work, however, uses the motion of a single two-dimensional scene region to estimate the motion of the scene prior to performing egomotion calculations [3].

All egomotion compensation techniques attempt to make an estimate of camera motion according to some transformation

model. While perfect estimates of camera egomotion can only be realized with highly non-linear and unwieldy models, robust results are achieved using simpler techniques. [4] uses an affine formulation based on the assumption that the distance from the observer to the targets is large compared to the depth of the scene. Some applications, though, require a slightly more complex bilinear model [5].

Methods for tracking moving objects once they have been detected vary widely. Two dimensional templates of the moving object can be constructed and then tracked in the video stream [1], or some combination of color and texture cues can be used to track objects across frames [6]. In keeping with the current algorithm's goal of generality, a particle filter was pursued instead of solutions that required information such as shape and color.

Probabilistic filtering methods (Kalman and particle filters) have been previously applied to tracking moving objects [7,5]. Particle filters in particular provide a way to model a multi-modal distribution, which is one way to handle the state probabilities of multiple moving objects in a video stream. The distribution resulting from the convergence of a particle filter can then be evaluated using a mixture of Gaussians and a threshold to group particles that belong to the same moving object [5]. As a simpler alternative to a method like this, the current algorithm instead employs a pair of simple image dilations and erosions. The results are then further grouped.

The cognitive algorithm that post-processes the particle filter output combines a variety of techniques related to reasoning about object motion. It could arguably be extensively derived from [8].

3 Our Approach[1]

Our approach is divided into four functional blocks: a sparse optical flow calculation based on tracking image features over consecutive frames, an egomotion compensation that uses the feature information to model camera motion, a particle filter to support multiple hypothesis as to the locations of possible moving objects, and a cognitive grouping operation

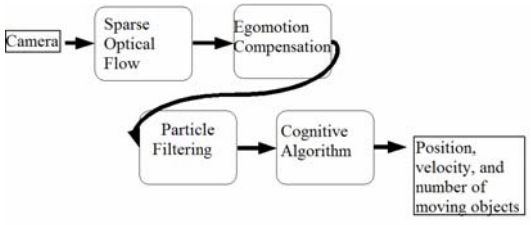


Figure 1: Algorithm Flow

that takes the output of the particle filter and determines the number of moving objects and their locations (Figure 1).

3.1 Feature Tracking

The features found and tracked by this algorithm are corners. This was part of the larger decision to keep the implementation as general as possible. While using Scale Invariant Feature Transform (SIFT) features was an option, it would have limited the ability of the algorithm to track any generic moving object. In order to find corners, the image is first smoothed using a large Gaussian filter to improve feature detection. The matrix

$$C = \begin{bmatrix} \sum \left(\frac{\partial E}{\partial x}\right)^2 & \sum \frac{\partial^2 E}{\partial x \, \partial y} \\ \sum \frac{\partial^2 E}{\partial x \, \partial y} & \sum \left(\frac{\partial E}{\partial y}\right)^2 \end{bmatrix}$$

is then used to find the minimal eigenvalue at each pixel location [10]. Non-maximum

[1] The implementation of this algorithm utilizes Intel's OpenCV library [9].

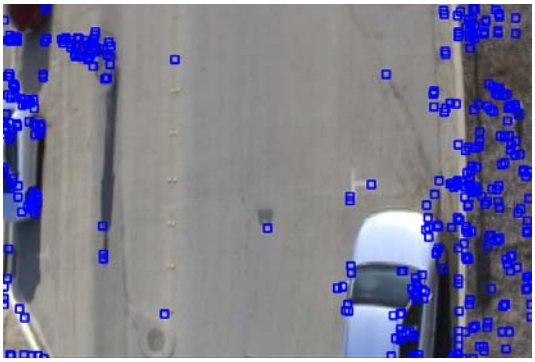


Figure 2: Detected Features

suppression is then performed and eigenvalues are compared to a threshold. Finally, eigenvalues located in close proximity to larger eigenvalues are removed from consideration. Once features are obtained, they are tracked using a pyramidal implementation of the Lucas-Kanade optical flow method [11]. This algorithm has two important benefits for the current application: it is robust to fairly large pixel displacements due to the pyramidal structure, and it allows the tracking of a sparse set of features to calculate camera motion, yielding a faster implementation than a dense optical flow calculation. Figure 2 shows a set of features returned using this approach on a frame of test video taken from an airborne camera.

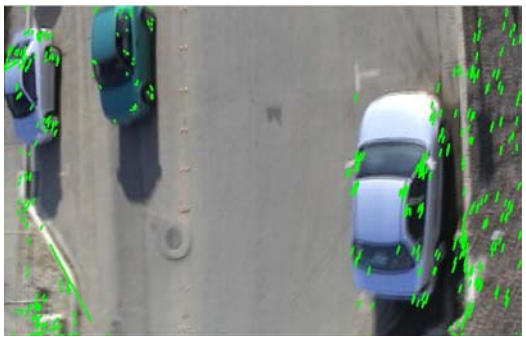
3.2 Motion Compensation

The feature detection and tracking algorithm described in the previous section provides a set of *i* feature locations $\{L_{i,n}\}$ in pixel coordinates $\{x_{i,n}, y_{i,n}\}$ for each frame *n* and the corresponding set of feature locations $\{L_{i,n+1}\}$ tracked to the next frame. After removing outlier features whose motion is greater than an assumed velocity limit, this provides a valid field of motion vectors for each feature *i* from frame *n* to frame *n+1*. Figure 3(a) shows a field of feature motion vectors for a moving car filmed from a moving helicopter. In order to distinguish camera egomotion from actual object motion in the scene, we must predict the motion field resulting from movement of the camera and remove it from the feature motion field.

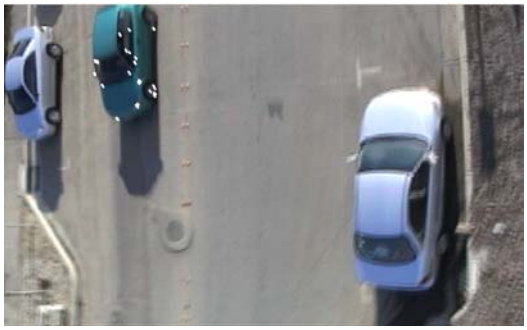
We model feature motion resulting from camera egomotion as an affine transformation between pixel coordinates in frame *n* and frame *n+1* consisting of a rotation **R** followed by a translation **T**:

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} \\ R_{21} & R_{22} \end{bmatrix} \cdot \begin{bmatrix} x_n \\ y_n \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \end{bmatrix}$$

Though the affine transformation is limited to translation, rotation, shearing, and scaling motions, when the time interval between frames is small, it can adequately represent most camera egomotion, as noted in [5]. The optimal transformation [**R**,**T**] can be found by a simple least-squares optimization. However, the optimal transformation contains inherent error since some of the feature correspondences used to estimate



(a) Feature Motion Vector Field



(b) Motion Vector Field after Motion Compensation

Figure 1: Motion Compensation and Moving Feature Detection

the transformation lie on moving objects with motion independent from the camera. We implement a double pass motion compensation algorithm proposed in [5] to ignore features on moving objects in determining our camera motion estimate. Once the initial transformation is estimated, the feature set is partitioned into two subsets – features whose motion vectors are accurately predicted by the transformation and features whose motion vectors indicate a second, independent motion. A second least-squares optimization is then performed on the former subset to obtain a more accurate transformation model.

Thus, by subtracting the motion field predicted by the affine transformation from the actual motion field found from feature correspondences, along with a suitable threshold, we are able to remove the motion of most static objects, leaving only motion vectors corresponding to dynamic moving objects. Figure 3(b) shows the feature motion vectors remaining after motion compensation and thresholding. While the affine transformation works well for predicting and removing most motion vectors resulting from static objects, in some cases features on static objects in consecutive frames may nonetheless move contrary to the predicted static motion field. In some cases this is a result of tracking non-unique features (e.g. features in a homogeneous grass field, specular highlights, etc), while in others the camera motion is complex and cannot be estimated completely using an affine transformation. In order to mitigate these effects prior to sending the moving features as input to a probabilistic filter, we require that each moving feature display motion in two consecutive frames. This eliminates many false, transient motions due to poor feature tracking and complex camera motion, the remainder of which are well handled by the probabilistic filter.

3.3 Particle Filtering

After compensating for the egomotion of the camera, the problem of recognizing potential moving objects in the scene is well matched to the properties of a particle filter. Since the motion of observed objects in the scene can be regarded as a partially observable Markov chain, a particle filter provides an approach that is scalable with the amount of computing time available [12].

The particle filter section of this algorithm is broken down into two subsections. In the first, a single particle filter is used to model a distribution of potential locations of moving objects, and in the second this information is processed to remove outliers and reward dense particle groupings prior to passing this information to the grouping section of the algorithm.

3.3.1 Distribution of object locations

The system state used in the particle filter consists of four parameters, an x-coordinate, a y-coordinate, the component of the velocity in the x-direction, and the component of the velocity in the y-direction. The velocity information is important because it allows the particle filter to make a prediction about the state of the system in the next frame.

In each frame, a set of particles, each with its own state, is placed in the image. By rewarding particles with locations and velocities close to those measured for the moving objects in the scene, in subsequent frames these particles converge to a distribution that describes the possible locations of the moving objects [12]. In this algorithm, the measurements are the motion vectors of features in a frame that are greater than a given threshold. These features are considered to be associated with moving objects, and their location is a measurement

Figure 4: Particle Filter Reward Map

of an object's location. Constructing a dense two-dimensional reward map from this sparse set of measurements is a challenge. Intuitively, it is desirable to reward particles who are close to moving features more heavily than those that are farther away. To accomplish this, a two-dimensional array of size equal to the size of the video image is created and filled with zeros. Non-zero values are placed in the array locations corresponding to the locations of features considered to be moving. The data in this array is then smoothed to create a map analogous to a topological geographic map. A reward map created for one frame of sample video is shown in Figure 4. A moving object is located in the lower-right portion of the image.

At this point assigning the *a posteriori* probabilities of the particles is simple. The *a posteriori* probability of each particle is the value in the reward map at the (x,y) location of that particle's state. Consequently, the particles for the next frame are drawn from the current particles with a probability proportional to the reward map.

This reward map could also be constructed using a difference image created by subtracting from frame *n+1* the image resulting from the warping of frame *n* according to the egomotion between frame *n* and frame *n+1* [5]. This approach is less general than the implementation discussed here for three reasons. First, objects moving quickly (more than their length between frames) are easier to pick up in the difference map than more slowly moving objects because they have a larger number of locations in the map with high difference values. Second, larger objects in a frame are easier to track than smaller ones because of the larger corresponding groups of differences in the difference map. Finally, objects with sharp contrast between internal regions will cause problems. If egomotion compensation is not perfect, a stationary object may be tagged as moving simply because of the large differences in intensity between regions internal to the object.

After using the reward map described above to iteratively reward particles and draw new sets, a multi-modal distribution results. Because a particle filter can support multiple possible state hypotheses, it can, for instance, converge on two moving objects, as shown in Figure 5. Alternatively, it would be possible to use multiple particle filters, one to track each moving object. An advantage of this method would be that each particle filter's overall state could be used to predict the position and velocity of the object it was tracking, and no further computation would be necessary. However, the code to trigger the spawning and killing of each filter when an object enters and leaves the camera's field of view introduces extra complexity in any implementation using this approach.

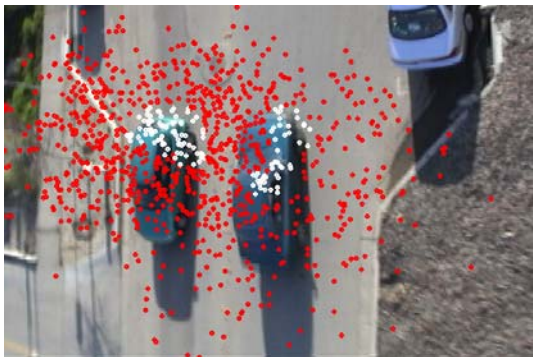


Figure 5: Particle Convergence

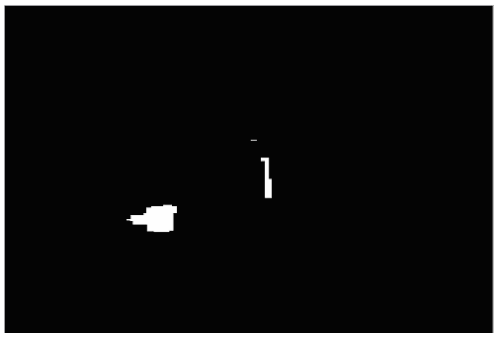
3.3.2 Outlier removal

Once the particle filter has begun to converge to a multi-modal distribution, the information about the positions of the particles that will receive a large reward from the reward map (the white dots in Figure 5) must be processed in such a way that it is robust to outliers in the data and is of maximum usefulness to the grouping part of the algorithm.

This algorithm uses a novel approach to perform this task. A two-dimensional array is filled with zeros, and (x,y) locations corresponding to particles that have received large rewards are set to a large constant value. The resulting array then has a dilating operation performed on it, spreading, and in some cases connecting, the constant non-zero regions in the array. At this point, an eroding operation is performed, whose parameters are set so that "blobs" created by the dilation of a single non-zero value, or a small set of non-zero values are eroded out of existence, while only blobs created by the dilation of large groups of non-zero values remain. Figure 6(a) shows a pictorial representation of an array where the array values have been changed to reflect which particles received high rewards. The video frame in question contains two cars passing each other. Note, this technique should not be confused with some of the more sophisticated blob techniques [13] since additional grouping is done when this information is passed to the grouping section of the algorithm.

(a) Spatially scaled good fit particles

(b) After Dilation



(c) After Erosion

Figure 2: Outlier Removal via Dilate/Erode Operation

Figure 6(b) shows the same array after a dilation operation has been carried out. Finally, Figure 6(c) shows the array after the erosion operation.

As can be seen from these figures, these operations help remove outliers while combining groups of features that are likely to belong to the same moving object. This improves the output of the grouping part of the algorithm.

3.4 Cognitive Tracking Algorithm

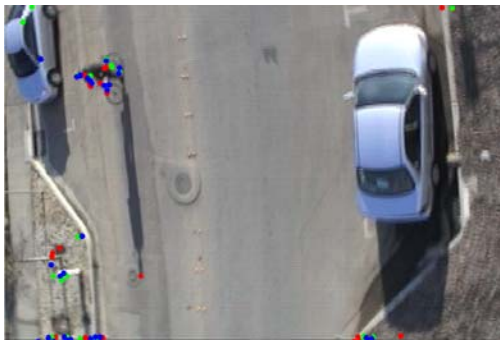
3.4.1 Overview

The final stage of our method is a cognitive algorithm. This algorithm uses artificial intelligence and decision-theoretic principles to add a variety of high-level services that work with the particle filter output. These services include: intelligent filtering of the particle filter output that reduces false positives while leaving other metrics

constant, estimation of the object's structure, averaging to keep that structure constant through momentary miscalculations of the particle filter, and a variety of high-level tracking capabilities such as unique object tagging and velocity and acceleration approximations.

3.4.2 Affinity Grouping

The first part of the cognitive algorithm involves determining the exact location of the moving object from the features and/or particles provided. To this end, the algorithm combines all features from the past three frames and forms as many rectangular grouping boxes as possible using one or more of these features as vertices. Boxes are filtered to eliminate those too large or too small and are only formed among features from a single frame. This process is depicted in the transition from Figure 7(a) to Figure 7(b). For illustration, red represents the most recent frame, green the second most recent, blue the least recent.



(a) Three-frame features

(b) Three-frame groupings

(c) Final grouping

Figure 3: Affinity Grouping

Next, we overlay all grouping boxes remaining from the above, as indicated in Figure 7(b). Each box then votes for the pixel area it covers. The pixel areas with the highest vote representation by frame delegation are considered more likely moving object candidates. Ties are broken based on the total number of votes received. We include a small buffer to correct for object motion between the frames. As a general rule, the algorithm requires a representation of at least two frames of the most recent three before it will conclude that an object is moving. A three-out-of-three mandate causes the algorithm to lose track unacceptably.

3.4.3 Multi-Frame Averaging

In some situations, the particle filter will lose-track temporarily or will vastly reduce its cover of the object. This is distracting to a human or computer observer processing the output. Therefore, before drawing the final grouping boxes, the algorithm averages their size and position with the corresponding box drawn in the most recent two calls. This averaging has varying weight – with five tiers – depending on how much the object has moved. Clearly, if the object has moved substantially, we do not want to average greatly with the previous positions. Even still, a slight lag can be noted in some tracked videos.

Moving objects are matched from frame-to-frame on a best-fit/best-effort basis. If there is a not-yet-averaged-on-this-call moving object record for the past two frames, averaging always occurs, even if the moving object depicted in those two frames is somewhat far off from the current frame. Other approaches cause the algorithm to lose-track.

To provide faster recognition of objects as they enter the field of view, a hull will be drawn around an object that passes the two-representation test described in Section 3.4.2 even if there are not enough moving object records from the past two frames to go around—for example when a new object appears.

3.4.4. Object Structure Estimation

Once final grouping boxes have been decided on, the algorithm extracts the features from the particle filter that are within those boxes plus a modest buffer. A

Figure 4: Structure Estimation

textbook convex hull algorithm creates a convex polygon around each object [14]. Figure 8 shows how the cognitive algorithm perfectly estimates the running woman's structure.

3.4.5. High-Level Track, Object Tags

Based on the hulls drawn, the cognitive algorithm deduces which sequence of hulls, across the entire video as it plays, represent moving objects. The algorithm assigns unique objects unique object tags and tracks them continuously. A list of their positions is maintained. Should an object stop, be lost, or move out of the field of view for a few frames, the algorithm will deduce that the object is the same upon its return to motion, track, or the frame. Naturally, velocity and acceleration are easily inferred.

The algorithm achieves this functionality using the best-fit/best-effort method presented in Section 3.4.3. That is, current object locations are matched to past object paths. A key difference in implementation between this search scheme and the one presented in Section 3.4.3 is that, here, we use a thresholding process that allows for no match to occur. This is a natural extension.

The algorithm correctly identifies the bicyclist in (sebastian.avi) (available online, see Section 4) as one unique object through his entire ride despite video noise and some particle-filter track-loss.

3.5 Net and Embedded Capable

Our implementation is entirely control-separated. We demonstrate this through an extensive graphical user interface (GUI) that talks with the algorithmic portion of the project over TCP/IP. Thus, our system is ready to go on-board the helicopter and be controlled over 802.11 or any other wireless networking paradigm supporting TCP/IP. In addition, our implementation separates the network functionality and the vision functionality onto two separate processors (if available) to maximize performance in an embedded situation: a NIC chip and a DSP, for example.

4 Performance

All input video files and resulting output files used to evaluate this algorithm are posted at www.motiontracking.info.

4.1 Overview

We evaluate the performance of our approach in terms of true positives, true negatives, false positives, and false negatives each with and without the cognitive algorithm. We define these terms as follows. A true positive occurs when a moving object is correctly identified as such by our approach in the current frame. A true negative occurs when there are no moving objects in the current frame and our approach correctly identifies this fact. A false positive occurs when our approach identifies an object as moving in the current frame when it is not. A false negative occurs when our approach does not identify that a specific object is moving in the current frame when, in fact, it is.

Evaluation occurs on both single object and multiple object video. Both the single object video (woman.avi) and the multiple object video (car.avi) are available online.

4.2 Single Object Performance

The single object performance results can be found in Table 1. Based on this data, we can conclude the following. When our approach with the cognitive algorithm identifies an object in the frame as moving, this judgment is correct with probability 0.7948. When the approach with the cognitive algorithm determines that no moving objects are present, this decision is correct with probability 0.9612. Overall, our approach with the cognitive algorithm is correct with probability 0.89855. These numbers without the cognitive approach are 0.587, 0.972, and 0.778.

	Particle Filter	Particle Filter + Cognitive
True Positives	64	62
True Negatives	105	124
False Positives	45	16
False Negatives	3	5

Table 1: Single Object Performance

4.3 Multiple Object Performance

The multiple object performance results can be found in Table 2. With the particle filter alone, the probability of correctness when an object is identified as moving 0.480. The probability of correctness when no moving object is identified is 0.1573. Overall, with the particle filter alone, the probability of correctness is 0.4071. Adding the cognitive approach, these numbers rise to 0.711, 0.16129, and 0.5345, respectively.

	Particle Filter	Particle Filter + Cognitive
True Positives	146	140
True Negatives	14	15
False Positives	158	57
False Negatives	75	78

Table 2: Multiple Object Performance

4.4 Discussion

Not surprisingly, the results clearly indicate that tracking multiple objects is more challenging than tracking single objects. In the latter case, the algorithm can simply take its best guess, threshold to determine if that

guess is likely to represent a moving object, and then identify accordingly. In the multiple object case, that technique is insufficient. The particle filter is likely to be drawn to one object more than others, and the non-primary objects can often have much the same properties as the noise thresholded out in the single object scenario. Thus, it can be very hard to determine whether a primary object with noise is really one object or multiple objects.

The cognitive algorithm performs very well. It universally reduces false-positives by a factor of three, while virtually maintaining the other performance results. Since the particle filter is the input to the cognitive algorithm, it is unlikely that the cognitive algorithm could improve much on the false negatives. That is, if the particle filter input does not include any features/particles on an object, it is virtually impossible for the cognitive algorithm to correct for this.

In some situations, the cognitive algorithm holds track because of the averaging while the particle filter loses track. In these situations, the cognitive approach has fewer false negatives than the particle filter approach alone. However, these situations are outweighed by the fact the cognitive algorithm is a multi-frame system and thus requires a few frames—a few false negatives—to lock onto an object.

5 Conclusion

A general real-time algorithm for tracking multiple moving objects on the ground from a single moving aerial camera was proposed. Feature correspondences from consecutive frames were used in a two-pass process to estimate camera motion as an affine transformation. This camera motion was then applied to each feature to obtain a set of features moving independently from the camera, which was the input to a probabilistic filter. The particle filter converged to a multi-modal distribution with peaks over moving objects, and a grouping procedure determined the number of moving objects and the optimal grouping to segment each moving object. The algorithm was implemented and tested on video from a single camera mounted on the Stanford Autonomous Helicopter. The performance statistics indicate that the system robustly detects and tracks moving objects on the ground.

The current algorithm outputs the position and velocity of moving objects in the two-dimensional image space. Thus, even if the camera intrinsic parameters were known, the unconstrained motion of the helicopter would preclude any sort of three-dimensional reconstruction. However, the Stanford Autonomous Helicopter is equipped with a global positioning system (GPS) receiver as well as an inertial measurement unit (IMU) which in combination can provide an accurate measurement of the camera's pose. With mild assumptions such as the planarity of the ground below the helicopter, our system could reconstruct the position and velocity of moving objects in three-dimensional world coordinates.

Acknowledgments

The authors would like to thank the members of the Stanford Autonomous Helicopter Project for their help in acquiring the video used for testing the algorithm.

References

[1] I. Cohen and G. Medioni. Detecting and Tracking Moving Objects in Video from an Airborne Observer. In *IUW 98*.

[2] A. Censi, A.Fusiello, and V. Roberto. Image stabilization by features tracking. In *Proceedings of the 10th International Conference on Image Analysis and Processing* 99.

[3] M. Irani, R. Rousso, and S. Peleg. Recovery of ego-motion using image stabilization. In *Proceedings of the IEEE Computer Vision and Pattern Recognition* 94.

[4] I. Cohen and G. Medioni. Detection and Tracking of Objects in Airborne Video Imagery. In *CVPR 98*.

[5] B. Jung and G.. Sukhatme. Detecting Moving Objects using a Single Camera on a Mobile Robot in an Outdoor Environment. In *International Conference on Intelligent Autonomous Systems*, Mar 2004.

[6] E. Ozyildiz, N. Krahnst¨over, R.Sharma. Adaptive Texture and Color Segmentation for Tracking Moving Objects. Pattern Recognition, Vol. 35, Nr. 10, pp. 2013-2029, 2002 .

[7] J. Kang, I. Cohen, and G. Medioni. Continuous Tracking Within and Across Camera Streams. In *CVPR 03.*

[8] Newell, A., & Simon, H. A. (1961). GPS, A program that simulates human thought. In H. Billing (Ed.), Lernende automaten. Munich: Oldenbourg KG. Reprinted in E. A. Feigenbaum & J. Feldman (Eds.), Computers and Thought. New York: McGraw-Hill, 1963.

[9] OpenCV: Open Source Computer Vision Library. Reference manual. Intel.

[10] E. Trucco, A. Verri. Introductory Techniques for 3-D Computer Vision.

[11] J. Bouguet. Pyramidal Implementation of the Lucas Kanade Feature Tracker. In OpenCV distribution.

[12] S. Thrun. Particle Filters in Robotics. In UAI 02.

[13] S.S. Intille, J.W. Davis, and A.F. Bobick. Real timeclosed world tracking. In *CVPR97.*

[14] Wayne, K. Lecture notes for COS 226: Algorithms and Data Structures. Spring 2004. Princeton University. Princeton, NJ.