Tartalomjegyzék

Kerítés fölött átdobott objektumok érzékelése kültéren

Róbert Csordás, László Havasi, and Tamás Szirányi

Elosztott Események Elemzése Kutatólaboratórium, MTA SZTAKI
{havasi.laszlo,sziranyi}@sztaki.mta.hu

Kivonat Kidolgozásra került egy módszer, amely alkalmas kültéri, monokuláris kamerkaképeket adott terület fölött átdobott objektumok érzékelésére. A módszer valós időben fut egy kültéri térfelügyeleti rendszer részeként. Más algoritmusokkal ellentétben ez a módszer optikai áramlás alapú, amely segítségével az eldobott objektumok pályája követhető, majd a pályákra parabolát illesztve az az esemény detektálható. A rendszer sikeresen észleli a különböző méretű elhajított objektumokat, és nem érzékeny azok forgására.

Abstract. We present a new technique for detecting objects thrown over a critical area of interest in a video sequence of monocular cameras. Our method was developed to run in real time in an outdoor surveillance system. Unlike others, we use an optical flow based motion detection and tracking system to detect the object's trajectories, and for searching parabolic paths. The system successfully detects thrown objects of various sizes and is unaffected by the rotation of the objects.

1 Introduction

Thrown object detection is widely used in surveillance systems. We introduce a robust method for detecting of objects thrown over a fence in outdoor environments with monocular camera system.

The challenge in developing such a method for outdoor scenes is to detect object trajectories reliably and to filter out motions that are irrelevant, such as a walking person or a waving branch of a tree. The thrown object can be small and blurry, making some of the feature extraction based tracking algorithms perform poorly.

The method described here is intended to work with specific camera placement. Cameras should be placed on the end of the fence and they should be looking right on the line of the fence. This would make the camera see a whole and clean image of the area of interest. The camera placement criterion is easily to meet in real world surveillance scenarios.

Detecting thrown objects by looking on their trajectories can be easily accomplished. They always follow a parabolic path, even after perspective transformation. After filtering of the parameters of the detected parabolic trajectories (like size, length and slope) the thrown objects can be identified.

Unlike other researches in the field, we use optical flow for finding object trajectories instead of tracking points of interests. This method is more reliable on tracking blurry, deformable or small objects. It is also more immune to object rotation around the axis in the image's plane that makes points of interest disappearing and re-appearing. It can be easily used to track multiple objects and is robust to motions that are not interfering with the position of the tracked object.

1.1 Related work

There are plenty of systems with the capability of detecting thrown objects. These systems are commercial, and their theory of operation is hardly known. There are very little publicly available studies in this field; most of the methods are based on change detection over a dedicated zone, and the path of motion is not considered. The most related study is [1]. The authors use inter-frame differentiating method for identifying areas of rapid motion in the image. They filter them in advanced ways, keep track of their centroids, and are using expectation maximization to find parabolic trajectories. Our method uses a different way of object tracking which are more suitable for outdoor use and different trajectory matching.

The method described in [2] uses interesting point detection based object tracking methods and measures of salience that are not related to parabolic trajectories of falling objects.

We use ideas described in [3] to improve the performance and robustness of the algorithm. We use background segmentation methods to determine areas of interest, and we run optical flow on these areas. These areas are usually much smaller than the area of the possible actions, thus reducing the resources needed for optical flow computation. The background segmentation also filters out some types of background clutter, which can otherwise interfere with the optical flow algorithm.

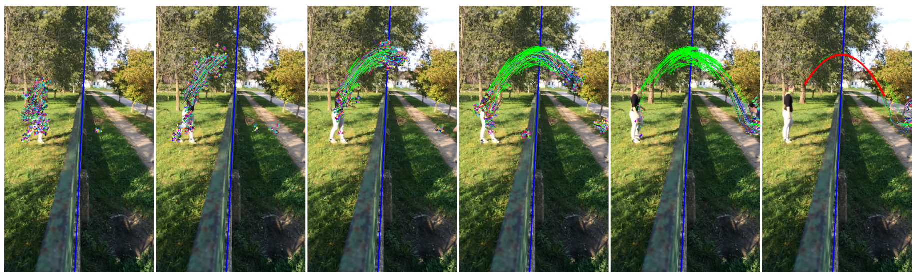
Fig. 1. The sequence of a throw and the output of the algorithm

2 PROPOSED METHOD

The goal of the project was to reliably detect objects thrown over a fence. The object can be deformable, and the algorithm must work in real world applications. Our method consists of four major steps: Frame capturing and preprocessing, moving point detection, tracking points and trajectory matching. The structure of the system is shown on Figure 4. An example sequence with the output of the algorithm is shown on Figure 1.

2.1 Frame capturing and preprocessing

The frame captured by camera is needed to be preprocessed. Firstly, it is transformed to monochrome image. The image size is scaled down to a smaller size that is enough for the accurate operation of the algorithm but yet it is less resource hungry. A Gaussian blur is applied to the resulting image. This makes the algorithm more immune to noise and makes optical flow algorithm perform better.

2.2 Moving point detection

After the preprocessing step, the moving points, that are possibly worth tracking, are detected. This is done in two steps.

The first step is foreground segmentation using Gaussian mixture-based background/foreground segmentation algorithm [4]. The resulting foreground map is eroded and dilated to get a rough map of foreground objects and their close neighborhood. A grid is placed on the foreground image. In our implementation the grid points have a fixed distance, but placing more points is smaller blobs could possibly increase the algorithms performance on small objects. In some cases the foreground segmentation step can be omitted. The algorithm performs reasonably well with a static, fixed grid of points.

After the grid is found, the optical flow is calculated between the current and the next frame. This gives a motion vector field. The optical flow is computed by Lucas-Kanade algorithm [5]. Is some cases when the flow vector is miscomputed with a large error, which makes the trajectory search difficult. To filter out the most of these errors, the optical flow is calculated twice: once in forward and once in backward direction. In backward direction we use destination points calculated by the forward step. If the resulting position vector differed from the initial one by a certain measure, the flow vector would be considered invalid.

Figure 2. shows the vector field calculated in this step. Only vectors with length greater than a threshold (here: min. 2) are shown. The beginning of the vectors are marked with points.

After the flow is calculated, it is needed to determine whether the particular point should be tracked. New points are added to the list of tracked points in two cases: if there is no tracked points near them, or if the direction of flow vector calculated on the tracked point and the one calculated on the grid differs. This ensures that there are no unnecessary points added to the list of tracked

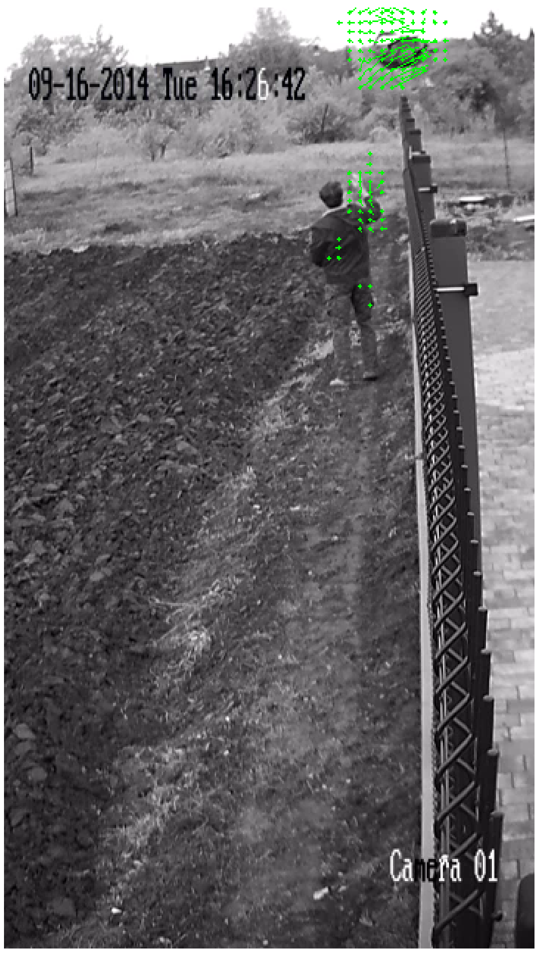

Fig. 2. Detected optical flow on the foreground of the image. Only flow vectors with length greater than 1 are shown.

points, but also no important points are missed. Only points with certain flow vector length are tracked. The limits of the vector length are determined based on typical speed of thrown objects in the scene.

2.3 Tracking points

The moving points detected by the previous step are added to the list of tracked points. The position history for every point is kept. On every step, the motion of the tracked points are calculated with the same dual direction optical flow algorithm used in the previous step.

The number of tracked points should be kept low in order to save resources. Practical constraints are used to discard inappropriate paths: long, straight lines are discarded. The paths which are nearly parallel with the fence are also discarded. The x coordinate of the path should not change direction. The reason for this is that the trajectories of the drops should go either from left to right or from right to left while in flight. There are some kinds of motions that have near parabolic trajectories. The best example is a waving branch. These distractions are also filtered by this method.

As a result of the object's rotation and its deformability, the tracked points can be lost relatively often. When the track of a point is lost, the moving point detector detects new motion vectors in the next frame, therefore a new tracked point is created. This results in multiple, shorter trajectories. The trajectory matching algorithm deals with this problem.

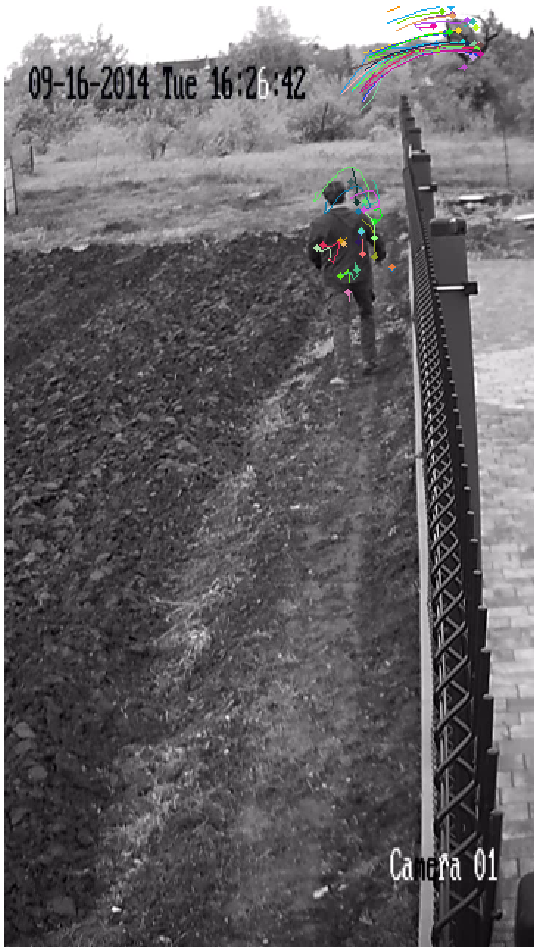

Fig. 3. Trajectories of tracked points

Figure 3 shows an example of motion history for the tracked points.

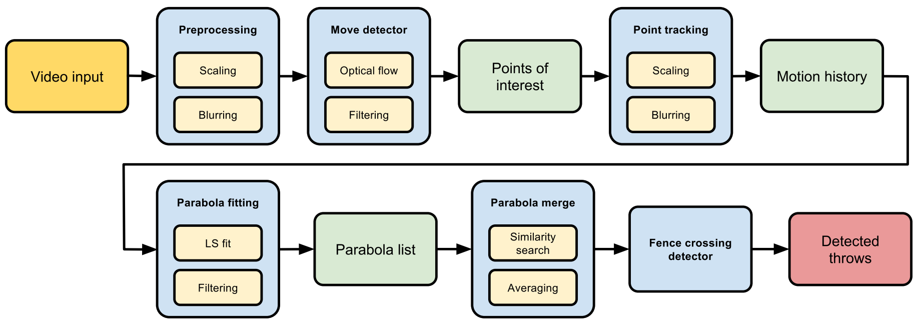

Fig. 4. The structure of the system

2.4 Trajectory matching

Trajectory matching consists of two separate steps. The first step finds a large number of relatively small parabola fractions. This step have a large number of false positive detections. The second step is to merge these parabolas to a final parabolic path belonging to actual thrown objects.

Finding parabolic trajectories The first step is trying to fit parabolas on the trajectories found by the previous step. LS fitting is used for this: a matrix is built from the coordinates of the points on the path, and the Moore-Penrose pseudoinverse [6] is taken. From this, the parameters of the parabola is easy to compute. Let us assume we have N points along the path. Name the coordinate pairs belonging to point i x_i, y_i, where $i = 1..N$. The equation of the parabola is given by:

$$y_i = ax_i^2 + bx_i + c \tag{1}$$

The unknown parameters are a, b and c. This equation holds for every point along the path. If we want to use linear regression, we have to get rid of the x_i^2 part. Because the value of every x_i known, we can easily rename them to z_i:

$$z_i = x_i^2 \tag{2}$$

Define matrix X as follows:

$$X = \begin{bmatrix} z_1 & x_1 & 1 \\ z_2 & x_2 & 1 \\ \vdots & \vdots & \vdots \\ z_i & x_i & 1 \\ \vdots & \vdots & \vdots \\ z_N & x_N & 1 \end{bmatrix} \tag{3}$$

vector Y^T as follows:

$$Y^\mathsf{T} = \begin{bmatrix} y_1 \; y_2 \; \cdots \; y_i \; \cdots \; y_p \end{bmatrix} \tag{4}$$

and vector P^T as:

$$P^\mathsf{T} = \begin{bmatrix} a \; b \; c \end{bmatrix} \tag{5}$$

We can write the following equation:

$$X \cdot P = Y \tag{6}$$

The least-square approximation of the P parameter vector can be calculated from equation

$$P = X^+ \cdot Y \tag{7}$$

where X^+ denotes the Moore-Penrose pseudoinverse.

The finite length parabolas have two more properties: their beginning and ending x coordinate. These are just the x coordinates of the leftmost and rightmost points in the position history.

After fitting a parabola to the position history, the square error of the fit is calculated at the points in the history. They are summed and normalized by the number of points, giving an error measure that is independent from the number of points. If the error is within certain limits, the trajectory is assumed to be a parabola.

There are many short trajectories found because of background clutter or other objects in the scene. To avoid unnecessary matching step, the approximate arch length is calculated prior to the fitting process. If the length is within certain limits the fit is calculated.

In some cases the paths contain only parabolic parts. Most of these situations arise from the fact that someone throws the object with his/her arm. If the camera have a clear sight on the object during the fling of arm, the points can be tracked before it leaves the hand of the thrower. In these situations the initial part of the path is closer to an arc of a circle than to a parabola. Nevertheless, the part of the trajectory is still a parabola after the object leaves the hand providing the initial force. In these situations the simple parabola matching fails because of big fitting error. Thus, these situations require a special treatment: only the part that is most likely parabolic is cut out from the position history and is used to a second try of fitting a parabola. In order to do this, the most likely part of the path must be defined: the path of object during the flight should have a maximum. This is most likely during a free flight, so it must be a part of a parabola. The maximum of the path is considered as a clear maximum if there are points on its both sides in x direction, and if these points are all closer to the bottom than the maximum. If the maximum of the path is clear, points in a window of specific length centered around the maximum is used. If there is no clear maximum found, the window is placed on the end of the path that is closer to the top of the image. The length of the window can be approximately determined by the frame rate, the average distance and the average speed of a thrown object.

The parabolic trajectories that belong to a thrown object must be oriented in the right direction: it must have a maximum and not a minimum. Therefore, the parameter a should be positive. They is also a certain upper limit for this parameter, because the parabola cannot be very narrow. This helps to filter out some false detections.

The parabolas passed all the tests are saved in a queue. The parameters a, b, c, x_{start} and x_{stop} are saved, along with the frame index of the detection which serves as a timestamp. x_{start} and x_{stop} defines the x coordinates of the ending points of the parabola.

Figure 5 shows an example of detected parabolas after throwing a backpack through the fence.

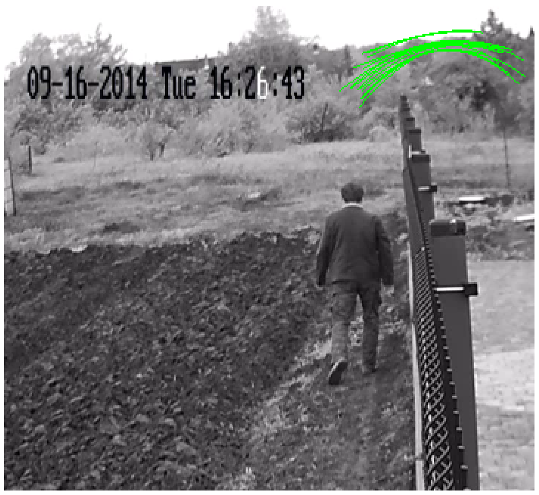

Fig. 5. Parabolas found based on motion history of tracked points

Merging trajectories The final step of the algorithm is run when parabola with a certain age is present in the list of detected parabolas. The age is determined by the associated frame index of the parabola. This introduces a small delay before the algorithm produces its final output, but it also allows that all the parabolas related to a single drop are detected before the final step runs. This delay is about 1 seconds.

The main function of this step is to find similarities in the list of detected parabolas. In order to do this, a measure for parabola similarity must be defined. The parabolas that looks similar by looking at them and are near each other should be considered similar. For this, three major conditions must be met: first, the maximums of the parabolas (which is a point $(x_{max,i}, y_{max,i})$) should be near each other. The limit for the distance is determined in empirical way. Second, the a parameters should be close. Third, their base must overlap. The base of a parabola is defined as the range of x values under the arc of the parabola. This criterion is easy to met when an object is thrown: there are many parallel parabolas, so it is very unlikely that they are all disconnected in such way that no bases overlap. Yet, it helps a lot to prevent false alarms when the line of the fence is taken in account. The reason for this is that the starting and ending points are more certain.

The algorithm does the merging process in two passes, and uses heuristics for faster and easier implementation. After an old enough parabola is found, the search begins for parabolas similar to it. The average of their maximum and their a parameter is calculated. The second pass uses these values to find a final set of parabolas. This eliminates the error that could arise when a parabola whose placement is misdetected is the oldest one.

When similar parabolas are found, they are averaged, and the final trajectory is found. The set of similar parabolas are removed from the list.

After the final trajectory is found, additional filtering step is applied. The starting and ending point of the parabola is connected with a straight line, and the absolute value of the angle of the line is checked. If the angle is to big the parabola is very asymmetric and is discarded.

Considering the line of the fence If we know the line of the fence, the performance of the algorithm can be improved. In our case, we tuned the parameters of the algorithm so it gives more false positive matches than false negatives. There is normally no motion that crosses the line of fences. So if the detected parabola that crosses the line of the fence, we can be certain that an object is thrown.

In steady camera surveillance systems the line of the fence is well known. The algorithm checks for line crossing by checking are the starting and ending points of the parabolas on the opposite side of the line assigned to the fence. If it is, the trajectory is classified as a trajectory of a thrown object and an event is generated.

Figure 6 shows the trajectory of the backpack. Bold red parabolas are the final matches.

3 EXPERIMENTS

We were able to successfully detect thrown objects on both outdoor and indoor scenes. For the experiments we made the following assumption: the object has a minimum diameter of 30 pixels and the stream has a minimal frame rate of 25fps.

The Precision is near to 100% because the filtering constraints of considering the line of the fence and the parabolic path and the effective path merging algorithms do not allow failing.

Recall rates vary from scenes to scenes. We have 4 different test sets. They are shown in Table 1. The object used for deformable tests was a shirt. A backpack was used as rigid object.

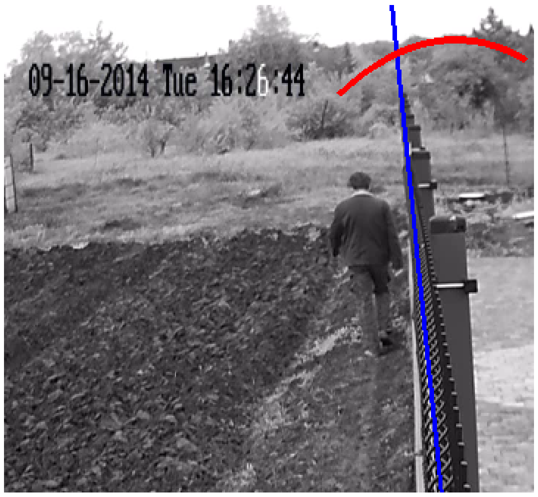

Fig. 6. The detected trajectory of the thrown object

Table 1. Test results.

Scene	No. of samples	Recall
B&W, Rigid	10	90%
Outdoor 1, Rigid	55	83%
Outdoor 1, Deformable	38	73%
Outdoor 2, Rigid	101	48%

Throwing deformable objects resulted in much higher accuracy than expected. In contrast, scene Outdoor 2 performed poorly. The reason for this is that on many samples, the thrown object is very small. Also, there is an artifact in the background whose color is very similar to the object.

4 CONCLUSIONS

The proposed algorithm performs well in outdoor scenarios, performing better than other methods in indoor environment [1]. There are, however some background patterns that can interfere with the Lucas-Kanade based motion tracker. This could be possibly improved by Kalman filtering, which needs further investigation. We were also able to achieve excellent results with deformable objects. The fusion of different methods, including tracking-based and optical flow based solutions, should be investigated to better cover all the cases. Also, pedestrian detection [7] and thrown activity may be merged in a common framework.

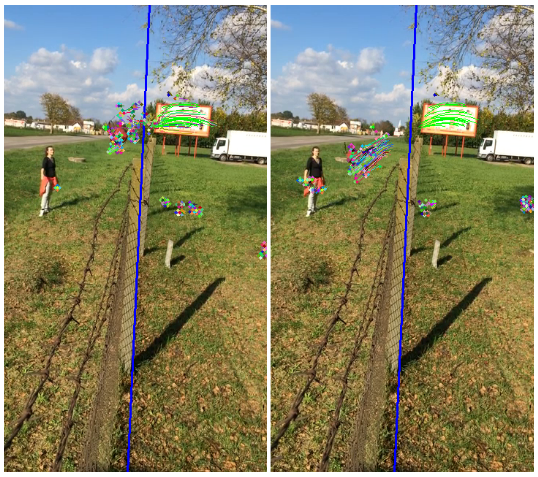

Fig. 7. An example of failure of tracking the object's path

ACKNOWLEDGMENTS

This work has been supported by the EU FP7 Programme (FP7-SEC-2011-1) No. 285320 (PROACTIVE project). The research was also partially supported by the Hungarian Scientific Research Fund (No. OTKA 106374).

References

1. E. Ribnick, S. Atev, N. Papanikolopoulos, O. Masoud, and R. Voyles, "Detection of thrown objects in indoor and outdoor scenes," in *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, Oct 2007, pp. 979–984.
2. P. Venetianer, A. Lipton, A. Chosak, M. Frazier, N. Haering, G. Myers, W. Yin, and Z. Zhang, "Video surveillance system," July 28 2005, US Patent App. 11/057,154.
3. S. Denman, C. Fookes, and S. Sridharan, "Improved simultaneous computation of motion detection and optical flow for object tracking," in *Digital Image Computing: Techniques and Applications, 2009. DICTA '09.*, Dec 2009, pp. 175–182.
4. Chris Stauffer and W.E.L. Grimson, "Adaptive background mixture models for real-time tracking," in *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, 1999, vol. 2, pp. –252 Vol. 2.
5. Jean-Yves Bouguet, "Pyramidal implementation of the lucas kanade feature tracker," *Intel Corporation, Microprocessor Research Labs*, 2000.
6. João Carlos Alves Barata and Mahir Saleh Hussein, "The moore–penrose pseudoinverse: A tutorial review of the theory," *Brazilian Journal of Physics*, vol. 42, no. 1-2, pp. 146–165, 2012.
7. L. Havasi, Z. Szlavik, and T. Sziranyi, "Detection of gait characteristics for scene registration in video surveillance system," *IEEE Trans Image Processing*, vol. 16, no. 2, pp. 503–510, Feb 2007.