Detecting moving objects in freely moving camera

By: Amer Sheeny, Foad Rezek

Supervised by: Udi Barzelay, Dror Porat*,*Prof. Ehud Rivlin

Contents

Motivation:

With the significant growth of moving camera platforms like smartphones, robots and vehicles; a large portion of the video content is captured by moving cameras, and there is vital need for algorithms that isolate “interesting” objects in the captured videos. These “interesting” objects are often moving objects.

The main challenge in this domain is distinguishing the motion induced by moving the camera itself, from the motion induced by the moving objects.

Project requirements:

The project assumes that:

1- No assumption on the camera movement or moving speed.
2- The camera center **does not** remain stationary during motion.
3- The background is **not** well approximated by a plane.
4- The method operates entirely using 2D image measurements without requiring an explicit 3D reconstruction of the scene.

Literature survey:

The first part of our project was to look up for state of the art papers related to our topic, find different approaches to this problem, and decide on an algorithm that can be implemented and yields good results.

We surveyed many papers, some of the relevant ones include:

> Extracting Moving Objects from a Moving Camera Video Sequence
> **Yasuyuki SUGAYA and Kenichi KANATANI**
>
> Real-Time video-surveillance by an Active Camera
> **G.L. Foresti and C. Micheloni**

And eventually implemented the approach described in:

> Background Subtraction for Freely Moving Cameras
> **Takeo Kanade, Yaser Sheikh, Omar Javed**

Why is it good?

- This paper is relatively new (2009).
- Included full algorithm description which makes the implementation part easier and smoother (tiny gaps were missing and were filled later).
- Results looked promising – the paper presented different case with very good accuracy.

- Takeo Kanade is one of its writers.

Algorithm overview:

The main goal is find a pixel-wise labeling for the foreground and the background.

This is how it works:

1. **Split the video** to windows, each window consists of a small number of frames (30) that our algorithm runs on.

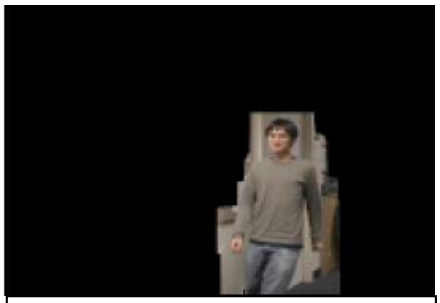
Picture 1: one of the frames in the window

2. **Detect & track features**: Use feature detection algorithm to detect interesting pixels in the middle frame. We then use a tracking algorithm to track the features through the rest of the frames in the window.
   The output of this stage is the **trajectory** vector for each one of the tracked features.

3. Use **RANSAC** algorithm to robustly select subset of three basis trajectories that parse the background space. The criterion for these trajectories selection is to maximize the number of features which lay in the space spanned by the 3 trajectories.
   After that we split the set of trajectories into two groups:
   a. Inliers (background): the three trajectories and all the trajectories lying in their space.
   b. Outliers (foreground): all the features not selected in the previous step.

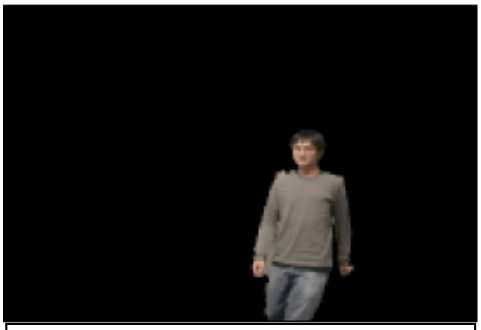
Picture 2: The tracked features in the frame. **Blue** dots are correct foreground points. **Red** points a background dots detected as foreground, **Black** dots are correctly detected background points.

4. Build a **probability model**, according to the inliers and outliers sets, match for each pixel a Boolean labeling: background/foreground.
   The main considerations are for the labeling are:
   i. The distance of the current pixel from the set of chosen trajectories.
   ii. The colors values of the current pixel compared to its neighbors.



Picture 3: The pixels labeling according to the probability mode

5. Use MRF (Markov Random Field) process to smoothen the results.



Picture 4: Applied Markov Random Field to smooth the results.

Implementation details:

We implemented this project with C++/opencv environment (VS 2010).

Our program receives as an input a path to the frames to be processed (mostly obtained from moving camera video), reads and process window after window (30 frames).

We start with **finding features points** – we used Shi Tomasi algorithm to detect features implemented with the opencv method: goodFeaturesToTrack. We tracked 500-2000 feature in the range of frames we tried.

After that we use optical flow to track the feature points in the rest of the frames. We used the Lucas Kanade algorithm implemented in opencv method: calcOpticalFlowPyrLK.

Then we run the RANSAC algorithm as follows:

- Do maximum of RANSAC_MAX_ITERATIONS (=1000) iterations:
    - Randomly choose 3 feature points w_i, w_j, w_k where w is 2x30 trajectory vector (x,y locations in each tracked frame).
    - Define $W_3 = \left[w_i^T, w_j^T, w_k^T \right]$
    - Define $P = W_3 \left(W_3^T W_3 \right)^{-1} W_3^T$
    - Use the following error function: $f\left(w_i \mid W \right) = \left\| P w_i - w_i \right\|_2$ and RANSAC_THRESHOLD (=0.1) parameter to find how many of the other features has an error less than RANSAC_THRESHOLD. Denote the total number of the features fulfilling this condition by: d.
    - If (d > 95% * TOTAL_FEATURE_POINTS)
        - Use the current w_i, w_j, w_k as background basis.
        - Add the d points along with the 3 basis to “inliers” set, and the other feature points to “outliers” set.
        - Break
- If no three trajectories fulfilled the condition, choose the three yielding maximal d value as background trajectories, create the inliers and outliers sets.

Now that we have the two sets: inliers with n points and the outliers with m points, we **build background/foreground models** based on probability model.

First of all we build 2 new sets:

$$\psi_b = \{[r,g,b,x,y] \ \ for\, each \ \ inlier \ \ feature \ point\}$$
$$\psi_f = \{[r,g,b,x,y] \ \ for\, each \ \ outlier \ \ feature \ point\}$$

Then for each pixel we define background and foreground probabilities:

1- The probability of pixel “x” belonging to the foreground:

$$P\left(x \mid \psi_b \right) = \frac{1}{n} \sum_{i=1}^{n} \varphi_H \left(x - y_i \right)$$

2- The probability of pixel “x” belonging to the background:

$$P\left(x \mid \psi_f \right) = \frac{1}{m} \sum_{i=1}^{m} \varphi_H \left(x - y_i \right)$$

Where φ_H is a density kernel function, H is the kernel matrix and the formula is:

$\varphi_H \left(z \right) = \mid H \mid^{-\frac{1}{2}} \varphi\left(H^{-\frac{1}{2}} z \right)$. φ is common kernel function like Uniform, Gaussian, Epanechnikov.

We used the Epanechnikov kernel function and given by the following formula:

$K(u) = \frac{3}{4}\left(1-u^2\right)1_{\{|u|<1\}}$. The H matrix was calculated per frame according to Silverman's rule-of-thumb (Silverman (1986)).

Now that we have for each pixel the probabilities values of whether it belongs to the background and foreground, we only need to find pixel-wise labeling for the frame. We use Markov Random Field to achieve smooth labeling

$$p(\mathcal{L}) \propto \exp\left(\sum_{i=1}^{N}\sum_{i=1}^{N}\lambda\Big(l_i l_j + (1-l_i)(1-l_j)\Big)\right)$$

Thus, we look for the labeling that maximizes the value:

$$\begin{aligned}\log p(\mathcal{L}|\mathbf{x}) \quad = \quad & \left(\sum_{i=1}^{N}\sum_{i=1}^{N}\lambda\Big(l_i l_j + (1-l_i)(1-l_j)\Big)\right) \\ & + \sum_{i=1}^{N}\log\left(\frac{p(\mathbf{x}_i|\psi_f)}{p(\mathbf{x}_i|\psi_b)}\right)l_i.\end{aligned}$$

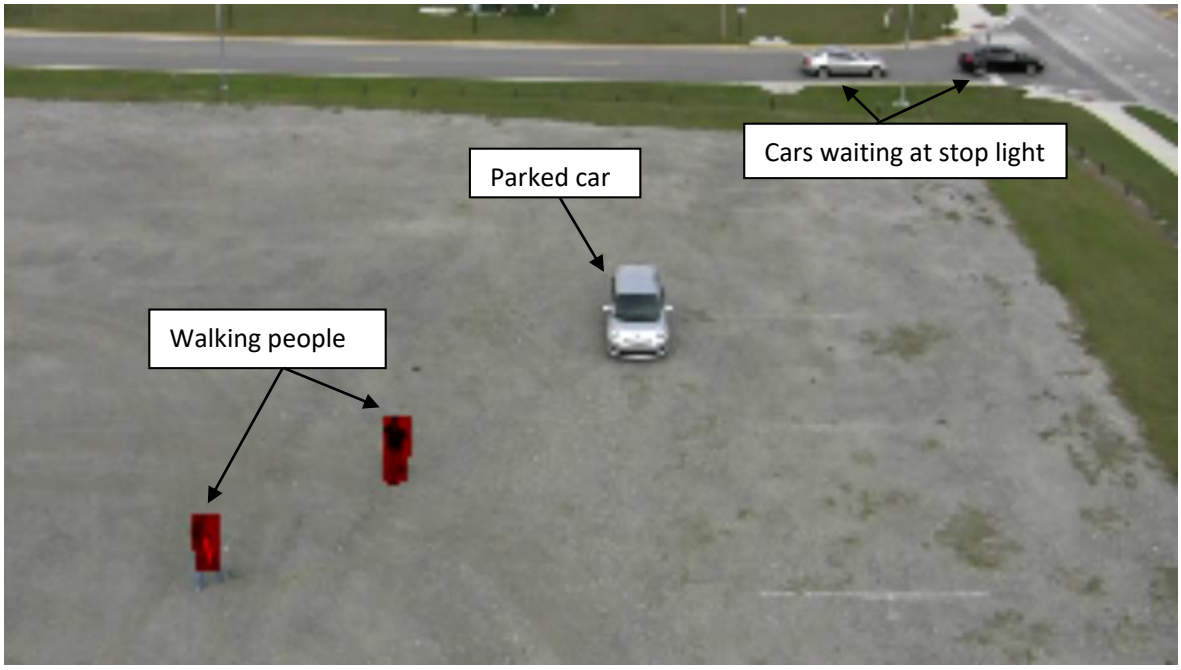
This can be solved efficiently using graph-cuts, as described in paper *"V. Kolmogorov and R. Zabih. What energy functions can be minimized via graph cuts? IEEE TPAMI, 2004."*

The idea is to build a graph that constitutes a reduction from the above problem to finding minimal-cut in a graph. A minimal cut can be found efficiently using known algorithms and problem-specific optimizations.

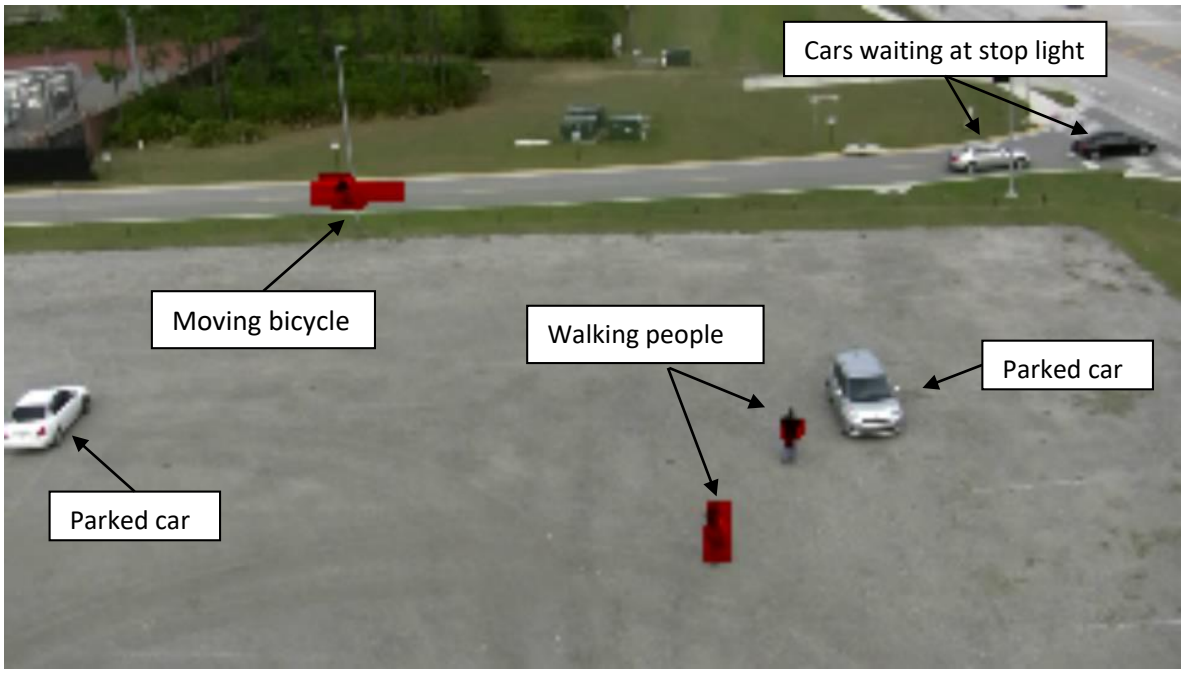
Algorithm Results:

The result quality varied slightly in video clips, depending the speed of the moving camera and the quality of the images. We have achieved an average of 0.65 recall and precision after running on three random videos, with varying degrees of quality and camera speed.

The following frame is from a video with moderate camera movement, and show high precision:



The camera then starts moving faster, and the precision is decreased.



Points for improvement - future enhancements:

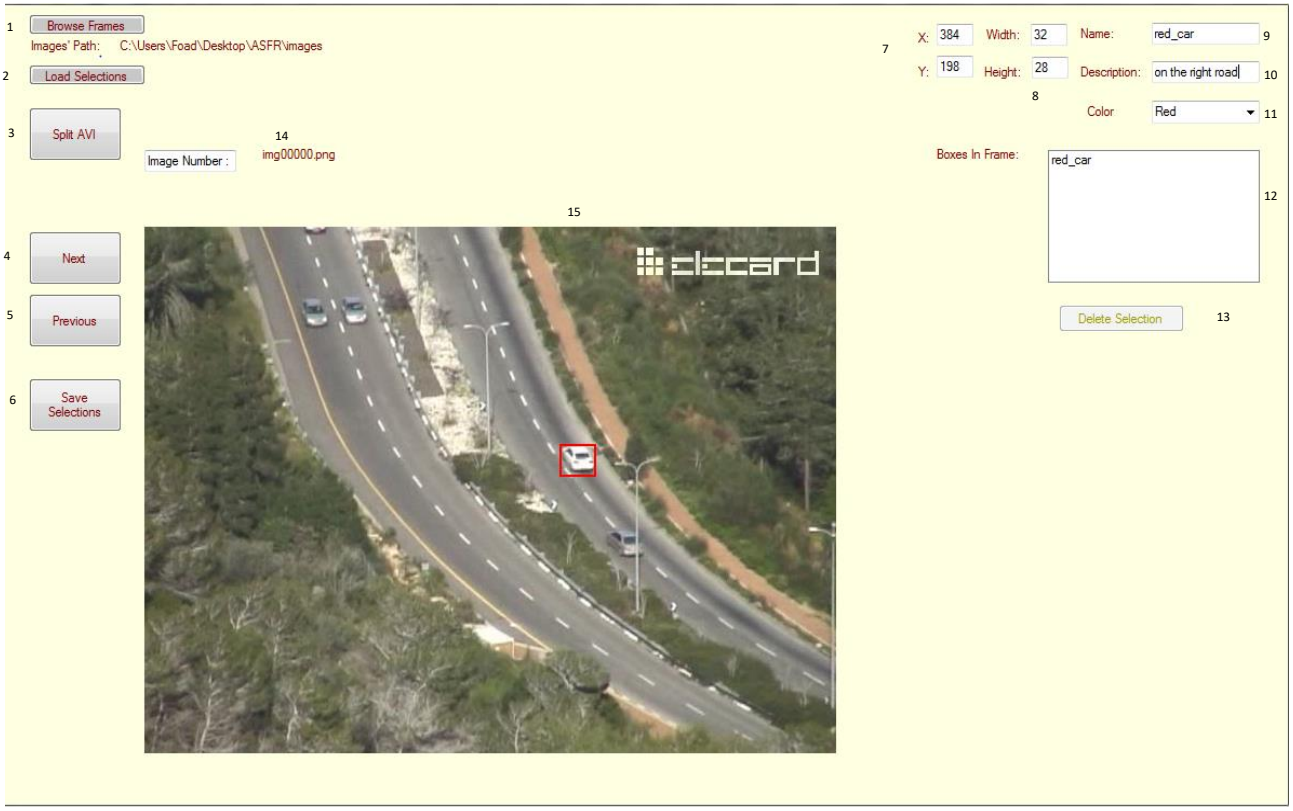
As mentioned above, the quality of the result is highly dependent on the video quality. Here are some enhancements that we think can improve the results:

1- Improve features detection and tracking: choosing the feature points is very critical part in the algorithm, a few noisy features can cause serious distractions. It is very important to detect and track quality features. This can be is very hard to accomplish if the camera movement is of high velocity.
2- Consider other approaches for choosing kernel matrix H. In our implementation, we used a matrix fixed for each frame. Other approaches suggest choosing the matrix dynamically per each pixel.
3- Study the effect of resizing on the results. Resizing the image leads to noticeable performance boost.

GUI tool for ground truth:

We implemented a C# GUI tool to easily creating ground truth for the tested frames. This tool gives the user a visual tools for creating ground truth and save them as text file. These files can be parsed later and used for measuring the algorithm accuracy and other statistics.

Here is the main window of the tool:



User guide:

1- Browse for a directory containing frames with the following format: img00000.png, img00001.png etc …
2- Load selections previously saved to a txt file.
3- Split an AVI video into frames with a suitable format.
4- Load next frame (auto saves selections for current frame, if next has no selections, initializes it with same selections as in current one).
5- Load previous frame (auto saves selections).
6- Save the selections to a txt file.
7- x,y coordinates of the top left corner of the current selection.
8- Width and height of the current selection.
9- Name of the selection.
10- Optional description for the selection.
11- Set selections' color.
12- The list of the selections in the current frame.
13- Delete the current selection box.
14- The name of the current frame.
15- The frame itself, use the mouse to draw a rectangle selection and resize it to fit the object.