Computer Vision Homework 3
Kalman Filter Tracking

By:	Andrey Leshenko	322026527
	Eli Tarnatursky	318880911
Lecturer:	Hagit Hel-Or	
HW Checker:	David Cohn	
Date:	19/06/2017	

Overview

In this project we implemented a Kalman Filter based object tracker. Given a short video of moving objects on a still background, it assigns IDs to objects in the video and attempts to track each object throughout the video.

Compilation

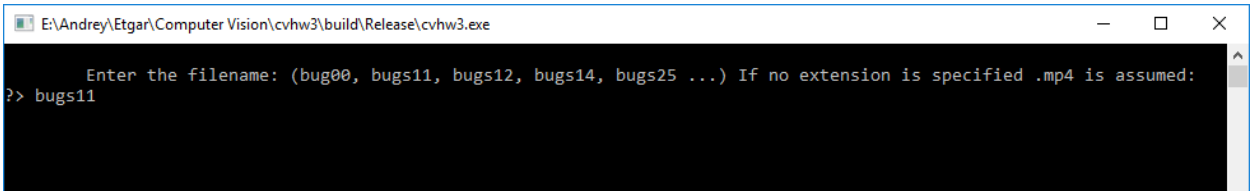
> **WARNING: Make sure to run the code in RELEASE mode. If you choose the run the code in debug mode, be aware that it will run much slower.**

If you already have a Visual Studio Solution of the project you can run the project from Visual Studio as usual. Otherwise you will have to generate the solution using CMake:

1. Install CMake from the official website.
2. If it is not yet installed, install OpenCV from the official site. The installer will ask you to where to extract the files. Remember this location. (On the Jacobs building computer it is C:/opencv3)
3. Open the CMake-GUI program.
4. Set "Where is the source code" to the correct path (for example E:/Andrey/cvhw3)
5. Set "where to build the binaries" to the same path + "/build" (for example E:/Andrey/cvhw3/build)
6. Press the Generate button below.
7. CMake will ask for a visual studio compiler version. Select your version. (Likely **V14 2015**)
8. CMake will say "**Error in configuration process**", because it can't find OpenCV. Press OK.
9. In the middle of the window there is a field "OpenCV_Dir". Set this field to the "build" directory inside your OpenCV installation. On the Jacobs computer this is "C:/opencv3/opencv/build".
10. Press Generate again. You should see "Generating done".
11. There is now a ready Visual Studio solution inside the "build" folder. Open it by pressing the Open Project button in the middle of the CMake window.
12. For video loading OpenCV depends on opencv_ffmpeg.dll. This DLL is found in the bin/ folder of the OpenCV installation. Find it, and either (1) put it into the directory of the executable, or (2) put it into some directory on the PATH, for example C:/windows/system32.
13. You can compile and debug using Visual Studio!

Usage

When started, the program prompts for a video name in the console window. The videos are taken from the $video/$ folder of the project.



Once a name is entered, a graphical window opens showing the video. In the background the program begins to compute the tracking information.

GUI controls:

- **Q**: Exit the program
- **Space key**: Stop/play video.
- **Left/right arrows**: Step through the video.
- **Up/down arrows**: Change presentation mode.
- **Home**: Jump to the beginning of the video.
- **End**: Jump the end of the video.
- **R**: Show tracking path.
- **N**: Show ID display.
- **C**: Show blob covariance.
- **Shift-C**: Show tracker covariance and velocity.
- **M**: Show markers (blob centers).
- **D**: Show blob indexes.
- **T**: Show possible transition edges.

At the bottom of the window is the Control Bar. The small rectangle shows the current position in the video. The green bar shows the progress of the tracking computation.

The title of the window displays the number of identified bugs in the current frame.

Blob Detection

Our tracking algorithm begins by building an image of the background, taking the median value of each pixel over all frames. Once we have the background we can compute the foreground of each frame as the abs diff with the background. We apply a threshold to the foreground, apply morphological operations to smooth the result and then find the connected components in the image (filtering them by size).

We represent each connected component as a Gaussian blob: mean point and covariance matrix. The blobs are then passed to the tracking algorithm.

Tracking Algorithm

The algorithm uses Kalman Filters for tracking individual objects. When multiple objects combine into one blob, it tries to consider all the possible ways the objects could go, and choose the most probable one.

The Kalman filter keeps track of position and velocity. It is defined using the following matrices:

$$state = (pos_x, pos_y, v_x, v_y), \qquad measurement = (pos_x, pos_y)$$

$$transitionMatrix = \begin{pmatrix} 1 & 0 & dt & 0 \\ 0 & 1 & 0 & dt \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \qquad measurementMatrix = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

$$processNoiseCov = \begin{pmatrix} 0.01 & 0 & 0 & 0 \\ 0 & 0.01 & 0 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 10 \end{pmatrix}$$

Additionally, $measurement$ and $measurementNoiseCov$ are set each frame according to the position and covariance of the tracked blob.

Multi-object tracking tries to assign stable IDs to each moving object. This isn't simple because sometimes multiple objects can merge into a single blob, and we have no good way to know how many objects a blob contains. For this reason, we always assume each blob contains a single object, but we also store the probabilities for other objects in this blob. If in later frames we find that one of the probabilities was in fact true, we go back, and change this object from a probability to a tracked object.

The tracking goes as follows: Take the current trackers and probabilities and use Kalman Filtering to update them for the new frame. Turn the highest probabilities into actual trackers, allowing only one tracked object per blob. For each probability chosen in this step we retroactively mark it as an actual tracker in the previous frames. Finally, we assign new ids to untracked blobs.

Code Architecture

To let the user use the program before all frames are processed, the program uses multiple threads:

- Video loading thread: loads the video, notifying a condition variable after each frame is loaded.

- Tracking thread: runs image processing on each frame and does the tracking.
- GUI thread: Periodically syncs its state with the tracking thread, and displays its state in a graphical window.

The entire tracking process state is represented using ProcessState structure:

```
struct ProcessState
{
	// Mutex for reading or writing the frame data
	mutex framesLock;
	condition_variable frameLoadedCond;
	vector<Mat> frames;
	bool finishedReadingFrames = false;
	int totalFrameCount = 0;

	// The background for background subtraction
	Mat background;

	// Mutex for reading or writing the state data
	mutex statesLock;
	vector<FrameState> states;
	// The index of the earliest state that was changed when this state was computed.
	// This is used for syncing by the GUI thread
	vector<int> firstChangedState;
};
```

For each frame a FrameState is computed:

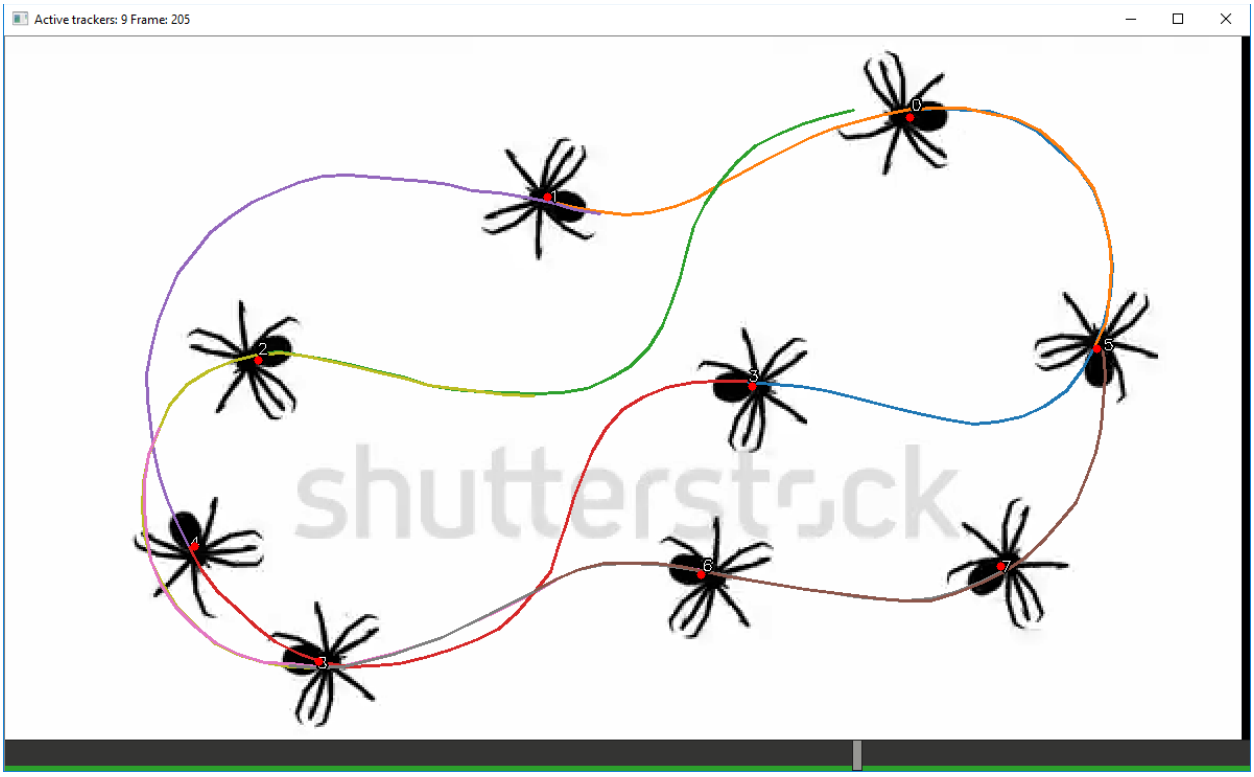
```
struct FrameState
{
	// The mean position of each blob (marker) in the image
	vector<Point2f> markers;
	// The covariance matrix of each blob (marker) in the image
	vector<Matx22f> covar;

	// For each marker i in the last frame we store (i, j)
	// for all j that he could have moved to.
	multimap<int, int> mov;
	// The inverse of mov
	multimap<int, int> imov;

	// For each blob (marker) we store tracking information
	// for different IDs.
	vector<vector<Tracker>> trackers;
	// Here we store (id, markerID) for each id located at markerID
	map<int, int> ids;

	// The first ID that was allocated at this frame state.
	// Used for visualizing the new IDs.
	int firstNewId;
};
```

We represent the different probabilities of IDs using the Tracker structure:

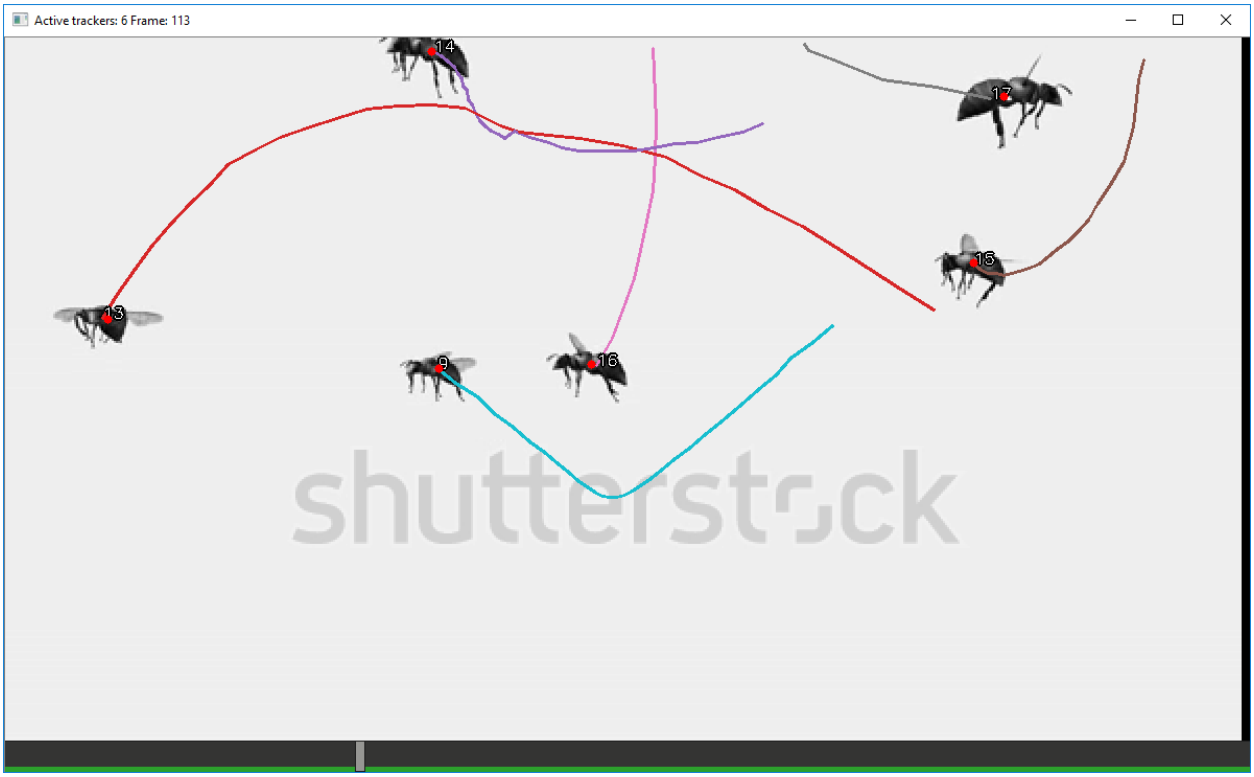
```
// Represents that some ID could be here with probability PROB,
// and movement state described by KALMAN.
struct Tracker
{
	int id;
	float prob;
	KalmanState kalman;
	// This is used when backpatching to know where
	// where this marker came from.
	int prevMarker;
	float distanceToLastSureMarker;
};

// We could use a cv::KalmanFilter everywhere, but it allocates
// a bunch of temporary data on the heap, which makes it expensive
// to have many copies of it. Here we store just the data we need
// in a fixed size struct.
struct KalmanState
{
	Vec4f mean;
	Matx44f covar;
};
```

Results

The algorithm shows very good results. In simple videos like bugs14 it easily tracks each spider throughout the entire video:

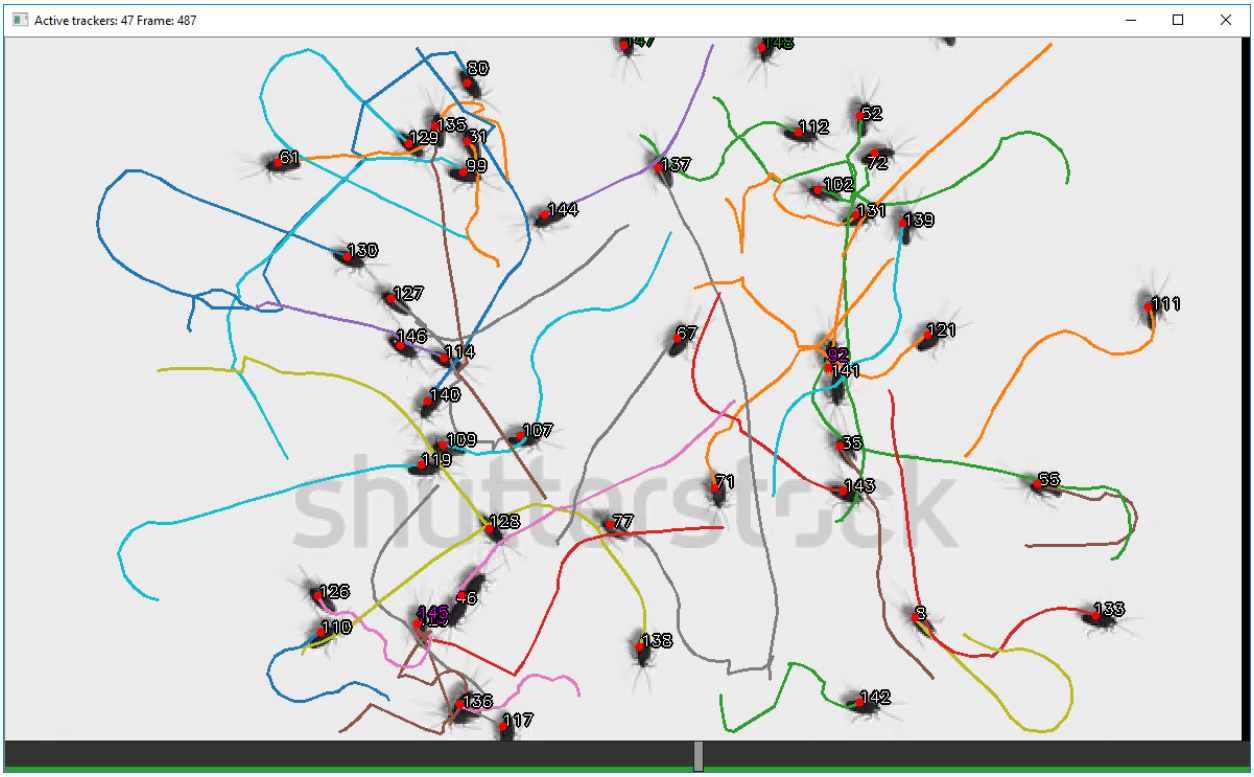


In the video with the wasps, the tracked objects are sometimes of very different sizes, which causes problems with blob detection. Despite that, most of the wasps are tracked correctly:



In the bugs11 video – the hardest video we were given – the tracking algorithms does fail in some complex cases, but it tracks well most of the time.

The mistakes happen when many bugs join into a single blob, and stay that way for a long time. In these cases the algorithm will sometimes assign incorrect ids after the bugs split apart.


Active trackers: 47 Frame: 487
80
135
129
31
61
99
144
137
112
52
72
102
131
139
130
127
146
114
140
109
107
119
67
92
141
121
111
35
71
143
53
123
77
126
46
145
110
138
3
133
136
117
142