MATLAB toolbox for learning object models from video

Michalis K. Titsias
School of Informatics, University of Edinburgh,
Edinburgh EH1 2QL, UK
michalis_titsias@yahoo.gr

1 Introduction

This report provides information for using the toolbox for learning layered object models from a video which is publically available from http://www.anc.ed.ac.uk/code/titsias/. The theoretical foundations of the algorithm used in the software can be found in [4]; see also [6]. A complete description of the algorithm can be also found in [3].

The toolbox consists of a set of MATLAB functions. To run this software you need to have installed a recent version of MATLAB (version 6.0 or later) together with the image processing toolbox. Note that the current version assumes grayscale images, so in case of RGB images the software will automatically transform the images to grayscale image format. The background object in each video frame is assumed to follow a translational motion across all frames, while the foreground objects can undergo similarity transformations (a combination of translation, rotation and scaling).

Section 2 provides a general description of the algorithm, section 3 gives installation information together with the list of the supplied files and directories. The use of the main functions in the toolbox is described in section 4. Finally section 5 discusses in detail one of the demos.

2 General description of the algorithm

We are given as input a set of images containing views of multiple objects, and wish to learn appearance-based models of each of the objects. Over the last decade or so a layer-based approach to this problem has become popular, where each object is modelled in terms of its appearance and region of support, see e.g. [5] and [1]. In [2] a principled generative probabilistic framework is described for this task, where each image must be explained by instantiating a model for each of the objects present with the correct instantiation parameters. A major problem with this formulation is that as the number of objects increases, there is a combinatorial explosion of the number of configurations that need to be considered. If there are L possible objects, and that there are J possible values that the instantiation parameters of any one object can take on, then we will need to consider $O(J^L)$ combinations to explain any image. Jojic and Frey [2] tackled this problem by using a variational inference scheme, searching over all instantiation parameters simultaneously. In contrast, in [6] we developed

MATLAB functions	GLOMO/	greedy.m learnbackg.m learnforegobj.m occlorder.m refinement.m reconstruct.m readframes.m imaffineforw.m imshift.m
Demos	GLOMO/	demo1.m demo2.m demo3.m demo4.m
Demos data directories	GLOMO/videos/	frey_jojic/ panorama/ arms_torso/
Documents	GLOMO/docs/	licence.txt doc.pdf

Table 1: Files supplied.

a *sequential* or greedy approach to object discovery whereby each model is extracted in turn from the whole dataset using a robust statistical method.

When we have video data, we can greatly speed up the greedy algorithm in [6] by carrying out approximate localization (or tracking) of the multiple objects in the scene. In [4] we have developed a method that first localizes each object and then learns its appearance. This method is applied to raw image sequence data and extracts the objects one at a time. First, the (possibly moving) background is learned, and moving objects are found at later stages. The localization (or tracking) algorithm recursively updates an appearance model so that occlusion is taken into account, and matches this model to the frames through the sequence. The MATLAB toolbox we present here implements the method in [4].

3 Files supplied

The software consists of a compressed tar-file that can be downloaded from `http://www.dai.ed.ac.uk/homes/s0129556`. Once you decompress the tar-file, a root directory is created with the name `GLOMO/` that contains the files and subdirectories displayed in the Table 1.

Note that in order to run the demos `demo1.m`, `demo2.m` and `demo3.m` you should also download the videos-tar files `frejojic.tar.gz`, `panorama.tar.gz` and `armstorso.tar.gz`. `frejojic.tar.gz` should be decompressed inside the directory `GLOMO/videos/frey_jojic/`, the `panorama.tar.gz` inside the directory `GLOMO/videos/panorama/` and the `armstorso.tar.gz` inside the directory `GLOMO/videos/arms_torso/`.

4 Main functions

In this section we provide a tutorial for using the main features of the toolbox. Section 4.1 and 4.2 discuss `readframes.m` and `greedy.m` which are the functions you need to explicitly call in order to run the software. Section 4.3 discusses the usage of the function `reconstruct.m`.

4.1 `readframes.m`

`readframes.m` is the function that loads the frames into MATLAB from a given directory. The frames can be in any MATLAB-recognizable image format. This function is called as follows:

```
>> frames = readframes(imdir, downsamp)
```

The `imdir` argument specifies the name of the directory in your machine where the images are located, as well as the image format. For example, if `imdir = '/disk/home/frames/*.png'`, then all the `png` images stored in the directory `/disk/home/frames/` are loaded. You can also give information about the name of frames. For example,
`imdir = '/disk/home/frames/mich*.png'` means that that all `png` images with the name starting with 'mich' are of interest. The `downsamp` argument can be 0 or a positive integer and specifies the degree of down-sampling you wish to apply to the frames, while loading them into to MATLAB. For example, if `dowsamp = 0` the frames are loaded in their original size and if it is 1 then the images will be half of the original size (in both axes) etc. The `downsamp` and argument is optional, so the command

```
>> frames = readframes(imdir)
```

is equivalent to

```
>> frames = readframes(imdir, 0)
```

The output variable `frames` is a $P_x \times P_y \times N$ table where P_x is the number of rows of each image, P_y is the number of columns and N is the number of images and it stores the images in grayscale format. If the frames are in RGB the function will transform them to grayscale. Note that `frames` is a `uint8`-MATLAB type variable. You can plot the nth image using e.g. `imshow(frames(:,:,n))`[1].

[1]Notice that if you wish you can use your favourite way to load the frames into MATLAB (e.g. you might already have the frames stored in a `.mat` file). The only constraint for using the software is that you have to store the frames in a $P_x \times P_y \times N$ table and in the temporal order according to which they appear in the video.

4.2 greedy.m

greedy.m is the function that you need to call in order to learn the background and foreground objects in the video sequence. It is called as follows:

```
>> [objs, transforms, order] = greedy(frames, op, wtrs, wrots, wscales)
```

frames is the video frames having the format specified in the previous section.

op is a vector of user-defined options. op(1) is the maximum number of iterations of each EM algorithm applied for learning the background and each foreground object. op(2) is the number of foreground objects to be found. Note that if op(2)=0, then only the background is learned. op(3) and op(4) can be 0 or positive integers and allow for increasing the size of each foreground appearance and mask pair[2], so as to be larger than the frame size. Note that making the masks and appearances larger than the frame size is useful when the object overlaps with the borders of the image or it is larger than the frame size. In particular, the size of each foreground appearance and mask is increased by op(3) in the upper and lower borders of an $P_x \times P_y$ image and by op(4) in the left and right borders, so each of them will be an $(P_x + 2 * \texttt{op}(3)) \times (P_y + 2 * \texttt{op}(4))$ image. op(9) takes the value 1 if you wish to find the occlusion ordering of the foreground objects in the video sequence (see section 3.4 in [3]), the value 2 if at the end you wish to refine jointly all the object parameters (see again section 3.4 in [3]). Finally, op(10) is set to a non-zero value in order to turn on the visualization features during running (see section 5). If op(10)= 0 the visualization features are turned off.

The remaining three inputs wtrs, wrots and wscales, specify the number and the type of the discrete transformations used during tracking[3]. wtrs is a two-dimensional vector that specifies the window of vertical and horizontal shifts in units of one pixel. For example, if wtrs = [2 3], the search window will be 2 shifts up and 2 shifts down in the vertical axis and 3 shifts left and right across the horizontal axis. Thus, generally the size of the search window over translations is $(2*\texttt{wtrs}(1) + 1) \times (2*\texttt{wtrs}(2) + 1)$. Similarly, wrots is 0 or a positive integer that specifies the number of discrete rotations. Particularly, the rotations are spaced in a one-dimensional grid where two neighbouring rotations in the grid differ by a unit of runit $= 360/\max(P_x, P_y)$ degrees, where $P_x \times P_y$ is the image frame size. Thus, the window search for rotations will be the set $\{-\texttt{wrts*runit}, \ldots, -\texttt{runit}, 0, \texttt{runit}, \ldots, \texttt{wrts*runit}\}$, which in total consists of $2*\texttt{wrots}+1$ rotations. Similarly, wscales specifies the number of scalings so as the corresponding window size is $2*\texttt{wscales}+1$. Each discrete scaling resizes the image in $(1/max(P_x, P_y))$ units. Thus each scaling changes the size by a factor of $(1 + \frac{i}{max(P_x, P_y)})$ where $i = -\texttt{wscales}, \ldots, -1, 0, 1, \ldots, \texttt{wscales}$. Note that the total search window has a size of $(2 * \texttt{wtrs}(1) + 1) \times (2 * \texttt{wtrs}(2) + 1) \times (2 * \texttt{wrots} + 1) \times (2 * \texttt{wscales} + 1)$ transformations.

To speed up the search over the discrete set of transformations during tracking, and also allow for large object motions we have implemented a coarse-to-fine strategy. Particularly, we form of Gaussian pyramid of all images involved in the matching criterion given in equations (10) and (12) in [4]. Then, starting from the coarsest level, we find the most probable transformation by searching over $(2*\texttt{wtrs}(1) + 1) \times (2*\texttt{wtrs}(2) + 1) \times (2*\texttt{wrots}+1) \times (2*\texttt{wscales}+1)$

[2]The foreground appearance and mask is denoted by ($\boldsymbol{\pi}_\ell$ and $\mathbf{f}_\ell$) in the referred papers.

[3]When we estimate the transformation j^{t+1} at frame at time $t+1$ given that we have already approximated the transformation j^t at the previous frames; see section 3 in [4] or section 4.2 in [3].

objs.B	$\longleftrightarrow$	$\mathbf{b}$
objs.Bmask	$\longleftrightarrow$	$\mathbf{m}$
objs.F$\{\ell\}$	$\longleftrightarrow$	$\mathbf{f}_\ell$
objs.Mask$\{\ell\}$	$\longleftrightarrow$	$\boldsymbol{\pi}_\ell$
objs.b_var	$\longleftrightarrow$	σ_b^2
objs.f_var(ℓ)	$\longleftrightarrow$	σ_ℓ^2

Table 2: Correspondence between the fields in the struct `objs` and the parameters names used in the published papers.

transformations where all the discretization units (see previous paragraph) are re-defined for the image size in this level. Then, we move to next finer level and we repeat the search over a window centred at the most probable transformation (properly interpolated to account for the current size of the images) of the previous level. This process is continued until we arrive the finest level of the pyramid, where the images have the original size. The final transformation is the one selected from the last search in the finest level.

We should point that for the background we only consider translational motion (as mentioned in the introduction), thus only the `wtrs` is used, while each the foreground objects all `wtrs`, `wrots`, `wscales` are relevant which allow the objects to undergo similarity transformations. Also passing the arguments `wrots` and `wscales` is optional and if they are omitted, only translational motion for the foreground objects will be considered. Thus, the command

```
>> [objs, transforms, order] = greedy(frames, op, wtrs)
```

is equivalent to

```
>> [objs, transforms, order] = greedy(frames, op, wtrs, 0)
```

and

```
>> [objs, transforms, order] = greedy(frames, op, wtrs, 0, 0)
```

The output variable `objs` is a struct that stores the parameters for the background and the foreground objects that are identified by the algorithm. The correspondence between the fields in the struct `objs` and the parameters they represent as they have been named in the published papers is illustrated in Table 2.

`transforms` is an $L+1$ dimensional struct that stores the transformations of the foreground objects and the background in each video frame. `transforms`$\{\ell\}$, $\ell = 1, \ldots, L$ corresponds to the transformations of the foreground objects and `transforms`$\{L+1\}$ to the background. The subfields of the `transforms` variable are explained in Table 3.

`order` represents the computed occlusion ordering of the foreground objects. It is a $1 \times L$ vector that represents a permutation of the integers from 1 to L, so that the first element in the vector points the object closest to the camera, the second element the second object closest to the camera and so on. .

Variable	Size	Description
transforms$\{\ell\}$.transls $\ell = 1, \ldots, L$	$N \times 2$	The nth line stores the vertical and horizontal shifts of the transformation of the nth frame
transforms$\{\ell\}$.matrices $\ell = 1, \ldots, L$	$2 \times 2 \times N$	Every nth element is a 2×2 matrix that stores the transformation synthesized by the best rotation θ and scaling s found for the nth frame according to $\begin{bmatrix} s * \cos(\theta) & -s * \sin(\theta) \\ s * \sin(\theta) & s * \cos(\theta) \end{bmatrix}$
transforms$\{L+1\}$.btransls	$N \times 2$	The nth line stores the translation of the background for the nth frame. Each translation is represented by the location of the upper-left corner in $\mathbf{b}$ so that a $P_x \times P_y$ block can be selected

Table 3: The subfields of the `transforms` variable.

4.3 `reconstruct.m`

`reconstruct.m` is a special function that can be used once you have learned the parameters of objects using `greedy.m`. This function has the following uses: it can reconstruct the original frames from the learned object parameters and the inferred transformations, it can create imaginary videos where some of the objects have been removed or the occlusion ordering has been changed, it can output the segmentation of the original video frames and others. The `reconstruct.m` is called as follows:

```
>> rframes = reconstruct(objs, transforms, order, Pxy, inout);
```

where the input arguments `objs`, `transforms` and `order` represent the objects parameters, the transformations in the training frames, and the occlusion ordering, respectively (see section 4.2). `Pxy` is a 2-dimensional vector that gives the size of training images, i.e. `Pxy` $= [P_x \ P_y]$. `inout` is a $L+1$-dimensional binary vector where the first L elements correspond to the L foreground objects (in the order that have been stored in `objs.F`) and the last element refers to the background. If `inout(i)=1`, the corresponding object will be included in the reconstruction, while if `inout(i)=0` the object will be ignored. Thus, `inout` can be used to remove certain objects. Note that if we choose to ignore the background, then a neutral background with 0 grayscale value will be used instead. The output variable `rframes` stores the created video frames in a $P_x \times P_y \times N$ table.

See `demo4.m` to get some ideas about how this function can be used to reconstruct the original frames, to obtain segmentation labels for the training frames and to create imaginary videos.

```
%%%%%%%%%%%%%%  demo1.m  %%%%%%%%%%%%%%
% Runs the algorithm for the Frey-Jojic sequence

% read the frames
imdir = 'videos/frey-jojic/*.png';
frames = readframes(imdir);

% user-defined inputs
op(1)=35; % number of iterations
op(2)=2; % number of foreground objects

op(3)=5; % enlargement of lower and upper borders of the foreground appearances
op(4)=15; % enlargement of left and right borders of the foreground appearances

op(9)=2; % occlusion ordering/refinement
op(10)=1; % visualization

% specify the window of translations
wtrs = [2 3];

[objs, transforms, order] = greedy(frames, op, wtrs);
```

Table 4: `demo1.m` file.

5 Demos

In this section we go through one of the demos, and particularly the `demo1.m`. We also explain the visualization features that optionally can be used during running. The MATLAB code of `demo1.m` is displayed in Table 4. This demo runs the algorithm in the Frey-jojic sequence (see webpage). The used-defined options (variable `op`) in the order they appear in the `demo1.m` (from top to bottom) have the following meanings: the number of EM iterations when we learn each object is 35, the number of foreground objects is 2, the masks and the foreground appearances are enlarged compared to the frame size by 5 pixels in the lower and upper image borders and 15 pixels in left and right borders. Also the computation of the occlusion order together with joint refinement of all the parameters will take place, and visualization features will be displayed.

Once the user-defined options are specified the next command in the `demo1.m` specifies the window of translations (variable `wtrs`) that is used during tracking the background and each foreground object. Particularly, 2 shifts up ans 2 down, and 3 left and right will be considered (in a coarse-to-fine way as explained in section 4.2). Note that we consider only translations for the foreground objects, so we don't need to specify the arguments related to rotation and scaling.

The next command in `demo1.m` runs the `greedy.m` function which carries out the learning of the objects. As explained above, when you run the `greedy.m` function we have the option to turn on the visualization features by setting `op(9)` to a non-zero value. This has the effect that several plots become visible during different stages of the program. Next we explain this features. When the function `greedy.m` is run we get the following messages:

```
>> [objs, transforms, order] = greedy(frames, op, wtrs, 0, 0)
***** Tracking the background *****
.........................................
***** Learning the background using EM *****
.....................................
***** Tracking foreground object #1 *****
.......................................
***** Learning foreground object #1 using EM *****
.....................................
***** Tracking foreground object #2 *****
.......................................
***** Learning foreground object #2 using EM *****
.....................................
***** Compute the occlusion order of the foreground layers *****
***** Joint refinement of the parameters *****
```

Each message surrounded by "****" indicates a separate stage of the algorithm. In the first stage, during tracking the background, a plot will open showing three images. The upper-left plot will show the current frame, the lower-left plot the background[4] and the lower-right

[4]As it updated in each frame according to equation (11) in [4].

plot will display the background mask (see section 3 in [4]) that indicates the area of the background that has been explored[5].

In the next stage the background is learned and a single plot is displayed showing the value of the background as EM iterates.

In the next stage the first foreground object is tracked. A panel with four plots will be visible in your screen where the upper-left plot is the current frame, the lower-left plot is the element-wise product of the foreground appearance and mask transformed by the best transformation for the current frame (i.e. what is displayed is the image $(T_{j^{t+1_1}}\mathbf{f}_1) * (T_{j^{t+1_1}}\boldsymbol{\pi}_1)$), the lower-right plot is the transformation mask, (i.e. the image $(T_{j^{t+1_1}}\boldsymbol{\pi}_1)$). Note that $\mathbf{f}_1$ and $\boldsymbol{\pi}_1$ are updated as we process the frames according to section 3.2 in [4]. Finally the upper-right plot displays all the pixels that have been removed from consideration in the current frame[6].

In the next stage the first foreground object's parameters are refined by EM (see section 3.2 in [4]) and the mask $\boldsymbol{\pi}_1$ as well the the element-wise product $\boldsymbol{\pi}_1 * \mathbf{f}_1$ are displayed.

The visualization features for the second object and any other object are the same with the first object. When the occlusion ordering of the foreground objects is computed there is no visualization, while during the final refinement of the parameters all the appearance parameters of the foreground objects and the background are plotted.

References

[1] M. Irani, B. Rousso, and S. Peleg. Computing Occluding and Transparent Motions. *International Journal of Computer Vision*, 12(1):5–16, 1994.

[2] N. Jojic and B. J. Frey. Learning Flexible Sprites in Video Layers. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition 2001*. IEEE Computer Society Press, 2001. Kauai, Hawaii.

[3] M. K. Titsias. Unsupervised Learning of Multiple Objects in Images. PhD thesis, School of Informatics, University of Edinburgh, 2005.

[4] M. K. Titsias and C. K. I. Williams. Fast unsupervised greedy learning of multiple objects and parts from video. In *Proc. Generative-Model Based Vision Workshop*, 2004.

[5] J. Y. A. Wang and E. H. Adelson. Representing Moving Images with Layers. *IEEE Transactions on Image Processing*, 3(5):625–638, 1994.

[6] C. K. I. Williams and M. K. Titsias. Greedy Learning of Multiple Objects in Images using Robust Statistics and Factorial Learning. *Neural Computation*, 16(5):1039–1062, 2004.

[5]Note that if the background remains static though the frames, as it the case for the video in `demo1.m`, the visualization of $\mathbf{m}$ is not that interesting. Try running `demo2.m` to see a more interesting visualization for $\mathbf{m}$ when we have a moving background.

[6]What is plotted is the vector $\mathbf{w}_1^t$ the semantics of which are explained the section 3.2 in [4].