UNIVERSITY OF BURGUNDY
VISUAL TRACKING , MSCV 2018

The Mean Shift Algorithm

Ioannis Tsagkatakis

November 11, 2018

1 The Mean Shift Algorithm

In this report, an implement visual of visual tracking using mean-shift algorithm is given. For the representation of the tracked object a metric based on the color histogram-based model is used. For the distance norm the Bhattacharyya distance is used. The algorithm is implemented in `Matlab`. The code this report as well as the results can be found in github at https://github.com/jtsagata/meanshift_tracking.

1.1 Initialization

The tracker can be initialized with various methods. For example a background subtraction methods can be used. Hera we initialize the position manually and save it with the video as a `.mat` file. This method is choosed to speed up the algorithm development.

The matlab files that do the choosing and saving are:

`demo_cars_prepare.m`	For the cars sequence.
`demo_head_prepare.m`	For the head sequence.
`demo_toy_prepare.m`	For the toy sequence.

1.2 Mean-Shift tracking

Mean-Shift considers considers that the feature space can be modeled as a probability density function (pdf). If dense regions (or clusters) are present in the feature space, then they correspond to the local maxima of the pdf. Thus the color histogram can be used as an estimator of the pdf.

The mean shift algorithm can be summarized as:

1. Fix a window around the traget point
2. Compute the mean data with-in the window
3. Shift the window to the mean
4. Repeat until converge

1.3 Object model

The object is represent by a color distribution as showend in figure 2

The histogram is given by the formula

$$h(u) = C \sum_{i=1}^{n} k\left(\|x_i\|^2\right) \delta\left(b(x_i) - u\right)$$

where $\|x_i\|^2$ is the normalized distace from pixel x_i to the region center and $\delta(\cdot)$ is the function

$$\delta(\alpha) = \begin{cases} 1 & when\, \alpha = 0 \\ 0 & otherwise \end{cases}$$

and C a normalizer. In order to take account the spatial nature of the pixels around the center location a kernel is used. A common used is the Epanechnikov kernel given by the equation

$$K_E(x) = \begin{cases} \frac{1}{2} C_d^{-1}(d+2)(1-x) & when\, x < 1 \\ 0 & otherwise \end{cases}$$

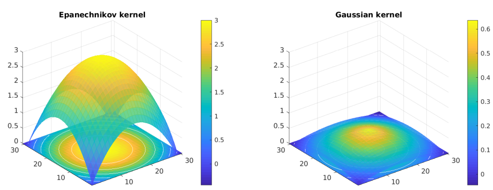

Figure 1: Common kernels

where C_d is the volume of the unit d-dimensional sphere. In our case, $d = 2$.

A graphical representation of an Epanechnikov and a gaussian kernel is given in figure 1.

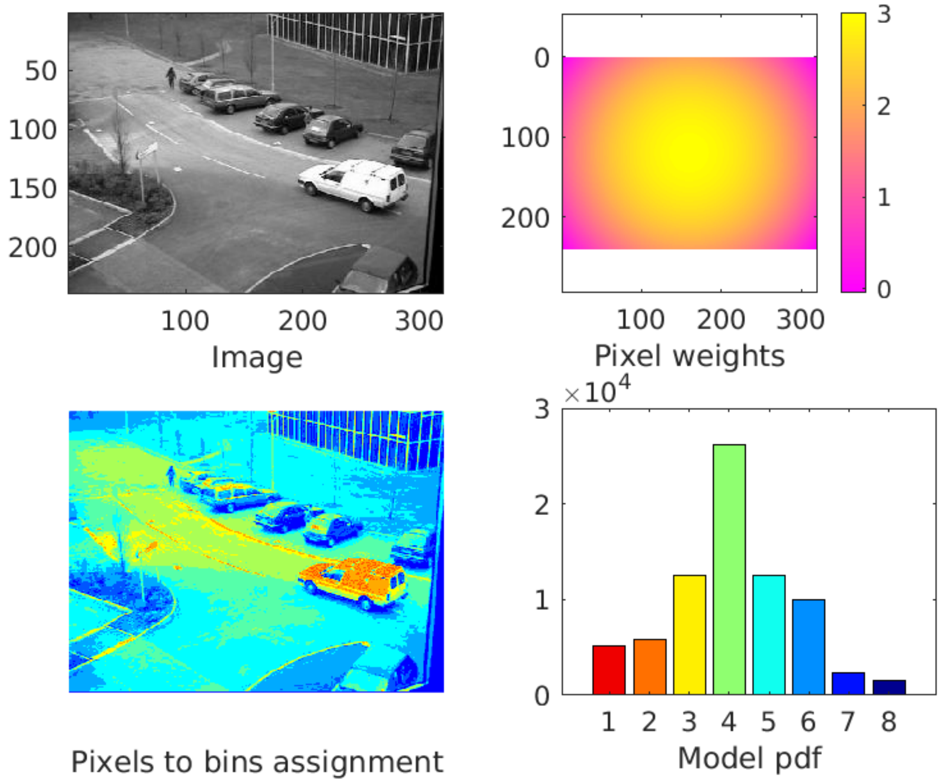

Figure 2: Color historgram representation

1.4 Color historgram distance metric

To estimate the similarity between a canditate location and the previous location the Bhattacharyya metric is used, given by the formula

$$\rho[p, q] = \sum_{u=1}^{m} \sqrt{q(u)q(u)}$$

Figure 2 show the bin assignment of a frame, the kernel and the final color distribution model from a single channel image.

1.4.1 Color distribution demos

A video demonstration with a sliding window over an image and the color distribution is provided.

`demo_algo_meanshift_car.m`	For the cars sequence.
`demo_algo_meanshift_toy.m`	For the toy sequence.

The resulting videos `demo_algo_meanshift_car.avi` and `demo_algo_meanshift_toy.avi` is on the github repository. A frame is given in figure 3.

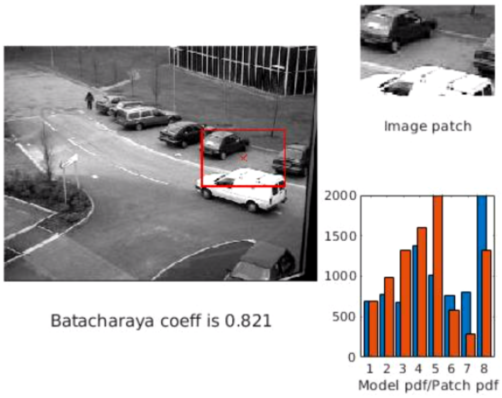

Figure 3: Mean shift historgram demo

1.4.2 Matlab codes

The file `kernelMatrix.m` calculates a kernel of size $n \times m$.

```
function d_matrix = kernelMatrix(m,n,varargin)
    %distancematrix(m,n) Calculate the  epanechnikovMatrix
       of size MXN
    %    Return an NXM kernel matrix of coefficients
    % or distancematrix(m,n, 'kernel','kernelname')
    %    where kernelname is 'gaussian' or 'epanechnikov' (
       default)

    % Handling of parameters
    p = inputParser;
    defaultKernel = 'epanechnikov';
    expectedKernels = {'gaussian','epanechnikov'};
    validScalarPosNum = @(x) isnumeric(x) && isscalar(x) &&
        (x > 0);
    addRequired(p,'m', validScalarPosNum);
```

```
    addRequired(p,'n', validScalarPosNum);
    addParameter(p,'kernel',defaultKernel,...
                 @(x) any(validatestring(x,expectedKernels)
                    ));
    parse(p,m,n,varargin{:});
    kernel=p.Results.kernel;

    cm = m/2;
    cn = n/2;
    norm_dist = sqrt( (cm-1)^2 + (cn-1)^2);

    [xx,yy]=meshgrid(1:m,1:n);
    X = sqrt((xx - cm).^2 + (yy - cn).^2) ./norm_dist;

    if strcmp(kernel, 'epanechnikov')
        d_matrix = 3/1 .* ( ones(n,m) - X.^2 );
    else % Gaussian
        d_matrix = 2/pi .* ( ones(n,m) - X );
    end

    % if x>=1
    d_matrix(d_matrix<0)=0;

    d_matrix = transpose(d_matrix);

end
```

The file `bhattacharyya_coeff.m` calculates the Bhattacharyya metric. The file `histDistMat.m` distributes pixels to histogram bins. The file `color_distribution.m` calculates the color distribution of a monochrome or RGB image. Note than an `eps` is added to every value, in order to avoid zero elements.

```
function patch_model = color_distribution(imPatch, Nbins,
   weights)

    patch_model=[];
    od = repmat({':'},1,ndims(imPatch)-1);
    imageDims=size(imPatch,3);

    % Histogram is the sum of histograms in each size
    for c=1:imageDims
        chanel_model = zeros(1,Nbins);
        if imageDims == 3
```

```
            chanel_image = imPatch(od{:},c);
        else
            chanel_image = imPatch;
        end

        whereToPut = histDistMat(chanel_image,Nbins);
        for indx=1:Nbins
            chanel_model(indx) = sum(sum( weights .* (
               whereToPut == indx)));
        end
        patch_model=[patch_model chanel_model];
    end

    % No zeroelements
    patch_model = patch_model + eps;

end
```

1.5 The tracking algorithm

The code that implements the mean shift tracking is given bellow. In order the algorithm not enter into infinite or large loops *loop_count* and *rho_loop_count* variables limits the number of iterations.

```
function [patch_roi,rho1]=meanshift_algorithm(frame,
   prev_center,target_roi,target_model,NBins)
%UNTITLED6 Summary of this function goes here
%   Detailed explanation goes here

    stable_center=false;
    loop_count = 0;
    while ( not(stable_center) & (loop_count < 10))

        patch_roi = xRoi(prev_center, target_roi.width,
           target_roi.height);
        patch_image = patch_roi.getRoiImage(frame);
        patch_model = xRoi(patch_image).color_model(
           patch_image, 'nbins', NBins);

        rho0 = region_rho(frame, patch_roi, target_model,
           NBins);
```

```
        % Derive the weights and compute the mean-shift
           vector
        W = meanshift_weights(patch_image, patch_model,
           target_model, NBins);
        assert(all(W(:)>=0));

        new_center = meanshift_vector(patch_image,  W);

        % Re-evaluate at new center
        patch_roi = xRoi(new_center, target_roi.width,
           target_roi.height);
        rho1 = region_rho(frame, patch_roi, target_model,
           NBins);

        % Converge to the new center
        rho_loop_count = 0;
        while ( (rho1 < rho0) & (rho_loop_count<25) )
            new_center = ceil((prev_center + new_center) /
               2);
            patch_roi = xRoi(new_center, target_roi.width,
               target_roi.height);
            rho1 = region_rho(frame, patch_roi,
               target_model, NBins);
            rho_loop_count = rho_loop_count +1;
        end

        if norm(new_center-prev_center, 1) < 1
            stable_center = true;
        else
            loop_count = loop_count+1;
        end
    end % while

end
```

The code that calculates the mean shift vector is given bellow. Note that the code works for for both grayscale and RGB images. The code is hightly optimized and vectorized.

```
function Z = meanshift_vector(imPatch, weights)

    imageDims=size(imPatch,3);
    od = repmat({':'},1,ndims(imPatch)-1);
    Z=zeros(imageDims,2);
```

```
    for c=1:imageDims
        if imageDims == 3
            chanel_image = imPatch(od{:},c);
            weights_image = weights(od{:},c);
        else
            chanel_image = imPatch;
            weights_image = weights;
        end
        Z(c,:)=meanshift_vector_2D(chanel_image,
           weights_image);
    end

    if imageDims == 3
        Z = mean(Z);
    end
end


function Z = meanshift_vector_2D(imPatch, weights)
    % All X coords
    ix=1:size(imPatch,1);
    SXW = sum(sum(ix * weights ));

    % All Y coords
    iy=1:size(imPatch,2);
    SYW = sum(sum(weights * transpose(iy) ));

    % Mean Location
    SW = sum(weights(:));

    % TODO: Check this
    Z = ceil([SXW/SW, SYW/SW]);
end
```

The code to calculate the meanshift weights is also presented here. The values of the weight matrix there always be bigger than zero by starting with a small `eps` value.

```
function weights = meanshift_weights(imPatch, TargetModel,
   ColorModel, Nbins)

    % Avoid zero elements
    weights = ones(size(imPatch))*eps;
```

```
    whereToPut = histDistMat(imPatch,Nbins);

    for i=1:Nbins
        multiplier = sqrt(TargetModel(i)/ColorModel(i));
        weights = weights + ( whereToPut == i) .*
           multiplier;
    end

    assert(all(weights(:) >=0));
end
```

1.6 Support code

Please look at the github repository[1] for the rest of the support code. A lot of operations have been put in a custom matlab object called xROI that represent a region of interest of an image. A region of interest knows his top left, center and bottom right coordinates and the center location. It can scale and get subimages and color distributions among other operations. Look at the files `xRoi.m` for the implementation and at `xRoi_test.m` for some basic usage and testing.

2 Mean Shift applications and demos

In the repository theres is the code for the following demos

`demo_cars.m`	Basic car demo
`demo_cars_adaptive.m`	Recalculates the histogram at each run (failed).
`demo_cars_varsize.m`	Adapt frame size (good results)
`demo_head.m`	Heads demo
`demo_head_varsize.m`	Adaptive heads demo
`demo_toy.m`	Toy sequence demo (bad results)
`demo_toy_varsize.m`	Adaptive toy sequence (bad results)

each demo loads the video and the tracking roi from a `.mat` file, runs the tracking algorithm and save the video result. Also its saves a picture with selective frames from the video and a text file containing timing information. The resulting files are in the github repository.

[1] https://github.com/jtsagata/meanshift_tracking

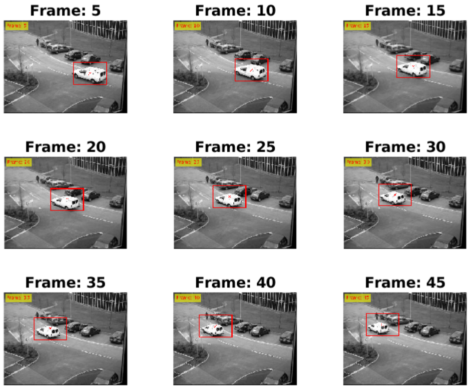

Figure 4: Car sequence simple demo

Some frames for the car sequence is given in figure 4 and in figure 5 when we adapt the frame window. The tracking result is quite good and computationally applicable.
In my laptop the execution times are

`demo_cars.m`	42.9381 secs
`demo_cars_varsize.m`	31.1478 (smaller)

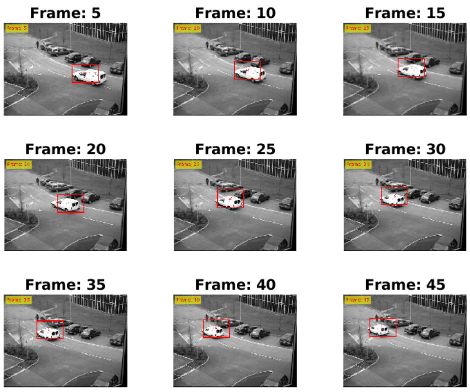

Figure 5: Car sequence variable roi size demo