Signal Processing Report - Lab 9 & 10

Olivier Jais-Nielsen - Group 2

January 13, 2009

Chapter 1

Tracking Procedure based on normalized cross correlation

1.1 Preliminary functions

1.1.1 Maximum of a matrix

This function returns the maximum MAX of a matrix m and its indexes i and j.

We first transform the matrix in a vector composed by the matrix's columns. The built in function *max* gives the maximum coefficient MAX of the new vector and its index I. To have the indexes i and j of MAX in the matrix, we use that if $\hat{I} = I - 1$ and $\hat{i} = i - 1$ and $\hat{j} = j - 1$, then $\hat{I} = \hat{j}.M + \hat{i}$ with M being the height of the matrix.

Algorithm 1.1 max_mat.m

```
function [MAX, i, j] = max_mat(m)
        [M, N] = size(m);
        [MAX, I] = max(reshape(m, numel(m), 1));
        i = mod(I - 1, M) + 1;
        j = floor((I - 1)/M) + 1;
```

1.1.2 Normalized cross correlation

This function returns the normalized cross correlation p between two matrices u and v with the same dimensions.

One of the matrix is transformed in a line and the other in a column so that multiplying them gives the scalar product of the two matrices.

Algorithm 1.2 cross_correl.m

```
function p=cross_correl(u,v)
        u = reshape(u,1,numel(u));
        v = reshape(v,numel(v),1);
        u = u - mean(u);
        v = v - mean(v);
        p = u*v/(norm(u)*norm(v));
```

1.1.3 Rectangle drawing

This function, given an image im_init, a point's coordinates i and j a height n and a width m, returns im the same image with a white rectangle centered on (i, j) with height $2.m + 1$ and width $2.n + 1$ or the part of the rectangle that fits in the image.

(i_min, j_min) and (i_max, j_max) are the top left hand corner and the bottom left hand corner of the rectangle. Their definition garantees that they are valid coordinates of the image.

Algorithm 1.3 rectangle.m

```
function im=rectangle(im_init, i, j, m, n)

        im = im_init;
        [M,N] = size(im);

        i_min = min([max([i - m; 0]);M]);
        i_max = max([min([i + m; M]);i_min]);
        j_min = min([max([j - n; 0]),N]);
        j_max = max([min([j + n; N]),j_min]);

        lm = i_max - i_min + 1;
        ln = j_max - j_min + 1;

        im(i_min:i_max,j_min) = 255 * ones(lm, 1);
        im(i_min:i_max,j_max) = 255 * ones(lm, 1);
        im(i_min,j_min:j_max) = 255 * ones(1, ln);
        im(i_max,j_min:j_max) = 255 * ones(1, ln);
```

1.2 Main algorithm

1.2.1 Measuring the best match

This function, given a reference patch *patch_ref* (i. e. the image of the bicycle chosen in the first image), an image *img*, a length *search_rad* (defining the size of the search area) and a point's coordinates *i0* and *j0* calculates the normalized cross correlation between *patch_ref* and each patch of *img* centered in a point of the serach area and returns the highest value *correl* along with the coordinates i and j of the corresponding point.

The search area is defined as a square centered in the reference point and with a side of $2.search\_rad + 1$. The program defines *res*, a matrix of the same size as the search area. For each point of the search area, the program puts the value of the normalized cross correlation between the reference patch and the patch centered in the current point in the matrix res. If the patch centered in the current point doesn't fit in the image, the value put in *res* is 0, meaning we exclude this point. Eventually, the maximum of *res* is returned as correl and its coordinates i and j are used to return the absolute coordinates of the corresponding point in the image.

Algorithm 1.4 measure.m

```
function [i,j,correl]=measure(patch_ref,img, search_rad, i0, j0)
        res = zeros(2*search_rad +1,2*search_rad +1);
        [M,N] = size(img);
        [m,n] = size(patch_ref);
        patch_m = floor(m/2);
        patch_n = floor(n/2);

        for i = (-search_rad:search_rad)
                for j = (-search_rad:search_rad)
                        if (i0 + i - patch_m) > 0 && (i0 + i + patch_m) <= M && (j0 + j - patch_n) > 0 && (j0 + j + patch_n) <= N
                                patch = img(i0 + i - patch_m:i0 + i + patch_m, j0 + j - patch_n:j0 + j + patch_n);
                                res(search_rad + 1 + i, search_rad + 1 + j) = cross_correl(patch_ref, patch);
                        else
                                res(search_rad + 1 + i, search_rad + 1 + j) = 0;
                        end
                end
        end
        [correl, i, j] = max_mat(res);
        i = i0 + i - search_rad -1;
        j = j0 + j - search_rad -1;
```

1.2.2 Main algorithm

The main algorithm loads the sequence *dataset.mat* in *img,* defines the initial coordinates of the bicycle we want to track *i0* and *j0*, the reference patch *patch_ref* which is the plain rectangle containing the bicycle and the width of the search area which is $2.search\_rad + 1$. The first white rectangle is then drawn according to the given coordinates. Then, for each frame of the sequence, we use the *measure* fonction with the reference patch and the coordinates of the bicycle calculated in the previous frame to calculate these coordinates in the current frame and to draw in it the corresponding white rectangle. Finally, we save these frames as a sequence in *dataset_tracked_correl.mat*.

Algorithm 1.5 cross_correl_track.m

```
function []=cross_correl_track()
        data = load('dataset.mat');
        img = data.img;
        i0 = 62;
        j0 = 146;
        patch_m = 15;
        patch_n = 10;
        search_rad = 10;
        [M,N,K] = size(img);
        patch_ref = img(i0 - patch_m:i0 + patch_m, j0-patch_n:j0+ patch_n);

        img(:,:,1) = rectangle(img(:,:,1), i0, j0, patch_m, patch_n);

        for im = (2:K)

                [i0,j0,correl] = measure(patch_ref, img(:,:,im), search_rad, i0, j0);
                img(:,:,im) = rectangle(img(:,:,im), i0, j0, patch_m, patch_n);

        end
        save('dataset_tracked_correl.mat', 'img');
```

1.3 Results

This tracking method works correctly provided the object tracked doesn't move too fast (i. e. it doesn't leave the search area between two frames) and it remains visible. Indeed, the first bicycle is tracked correctly until it gets entirely hidden by trees.

Figure 1.1: Frame 1

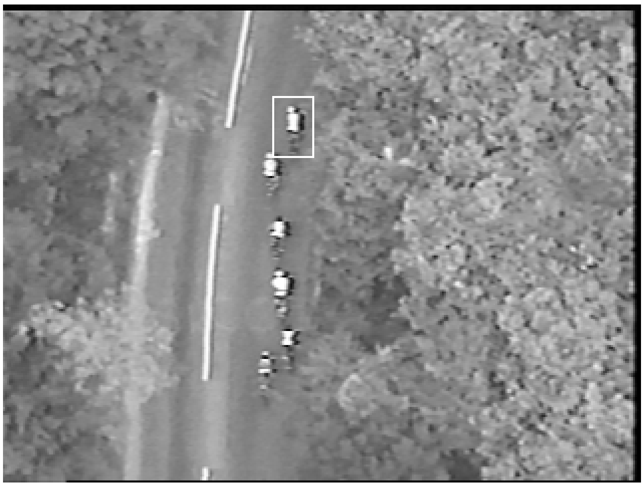

Figure 1.2: Frame 105

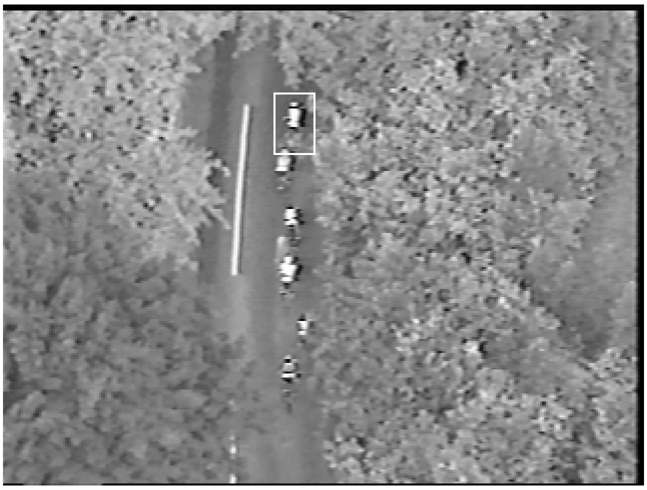

Figure 1.3: Frame 145

Chapter 2

Tracking Procedure based on the Kalman Filter

2.1 Preliminary questions

2.1.1 Mathematical background

We define the state vector as $x_t = \begin{pmatrix} px_t \\ py_t \\ u \\ v \end{pmatrix}$ where px_t and py_t are the estimated coordinates of the bicycle and u and v are the components of its speed. Thus the state transition matrix has to be $A = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$. Finally, $z_t = \begin{pmatrix} px'_t \\ py'_t \end{pmatrix}$ thus $H = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$ with px'_t and py'_t being the measured coordinates of the bicycle.

2.1.2 Measurements

We will use the measure function based on the normalized cross correlation previously written as a measurement. The quality of the measurement will be given by the value of the maximum normalized cross correlation returned along with the corresponding coordinates. If this value is superior to $1/2$, then the quality is considered good and $R = 50.I$; otherwise, the quality is considered bad and $R = 10000.I$.

2.2 Algorithm

2.2.1 The program

First, all the parameters are defined with obvious notations or as before. u and v are initialized equal to zero as the speed is unknown and relatively low. x_post represents $\hat{x}_t$ and x_pri $\hat{x}_t^-$. We initialize x_post with the known values of px and py and the values of u and v. The a posteriori error covariance is initialized as zero as the first position is known. However, if px and py are estimated coordinates, the corresponding coordinates in the matrix representation of the frame are $i0 = [py]$ and $ij0 = [px]$. As before, we draw the first rectangle. Then, for each frame, we proceed to the prediction step, then we invoke the function measure with the predicted coordinates and redefine R according to the quality of the measure. We proceed to the correction step and draw the rectangle according to the corrected coordinates.

Eventually, we save the sequence in *dataset_tracked_kalman.mat.*

Algorithm 2.1 kalman_track.m

```
function []=kalman_track()
        data = load('dataset.mat');
        img = data.img;
        [M,N,K] = size(img);
        patch_m = 15;
        patch_n = 10;
        search_rad = 10;
        A = [[1,0,1,0];[0,1,0,1];[0,0,1,0];[0,0,0,1]];
        H = [[1,0,0,0];[0,1,0,0]];
        Q = [[5,0,0,0];[0,5,0,0];[0,0,1,0];[0,0,0,1]];
        px = 146;
        py = 62;
        u = 0;
        v = 0;
        x_post = [px; py; u; v];
        z = [px; py];
        P_post = zeros(4,4);
        patch_ref = img(py - patch_m:py + patch_m, px-
            patch_n:px+ patch_n);

        i0 = round(x_post(2));
        j0 = round(x_post(1));

        img(:,:,1) = rectangle(img(:,:,1), i0, j0,
            patch_m, patch_n);

        for im = (2:K)

                x_pri = A*x_post;
                P_pri = A*P_post*A' + Q;
                [i,j,correl] = measure(patch_ref,img(:,:,
                    im), search_rad, i0, j0);
                z = [j; i];
                if (correl > 1/2)
                        R = 50*eye(2,2);
                else
                        R = 10000*eye(2,2);
                end
                K = P_pri*H'*inv(H*P_pri*H' + R);
                x_post = x_pri + K*(z - H*x_pri);
                P_post = (eye(4,4)-K*H)*P_pri;

                i0 = round(x_post(2));
                j0 = round(x_post(1));
                img(:,:,im) = rectangle(img(:,:,im), i0,
                    j0, patch_m, patch_n);

        end


        save('dataset_tracked_kalman.mat', 'img');
```

2.2.2 Results

Unlike the previous method, this filter is able to track the bicycle even after it has been briefly hidden.

Figure 2.1: Frame 1

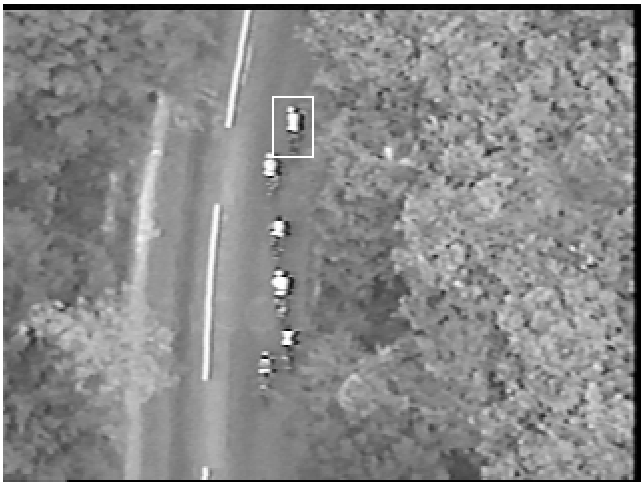

Figure 2.2: Frame 130

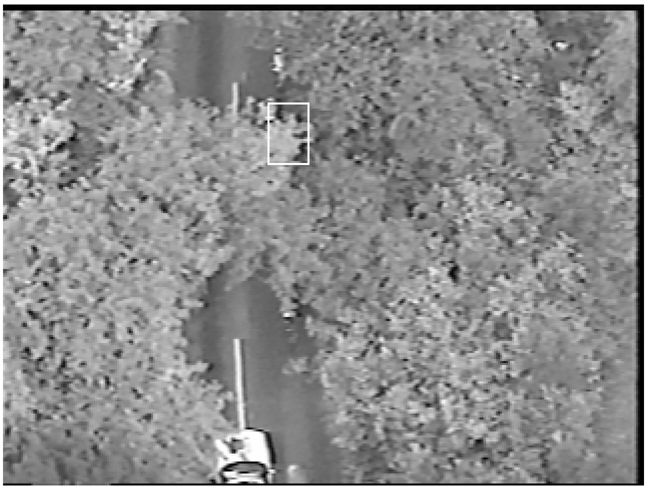

Figure 2.3: Frame 145