Translational Shift Estimation Using a Projection-Based Algorithm

by Mahdi Zerara

Introduction:

In this project, we will use projection-based algorithms that were proposed by Cain et *al* in [1] and Ratliff in [2], to reduce the undesired horizontal and vertical translational pixel and sub-pixel shifts in shaky videos. Cain et *al* algorithm relies on reducing 2D images into 1D horizontal and vertical vectors. Then using the 1D cross-correlation function, the algorithm estimates the undesired shifts between each pair of successive frames in a video sequence. Ratliff's algorithm is based heavily on the linear interpolation concept. The key strengths of both methods reside in their inexpensive computation costs, implementation simplicity and high shift estimation accuracy. In this project, we provided our Matlab code for the above-mentioned approaches, followed by a brief explanation of the algorithms' procedures. We assume here that the reader is familiar with Matlab coding, and has a relatively strong mathematical background. Also, going through the cited papers is key in understanding the steps of this proposed work.

Translational Shift Estimation Using a Projection-Based Algorithm:

Prior to applying the projection-based algorithms, we load the uncorrected video data sequence that was stored as a cell of matrices. Notice that in our code, the cell variable is named "VideoData" (to ensure that the code runs fine with your data, make sure that you name the data stored in the cell accordingly).

```
close all
clear all
warning off
clc

%% Read the video data
load VideoData1.mat
numberOfFrames = length(VideoData);
```

The first step in our code is to use the projection-based algorithm [1] to reduce each 2D frame in a video sequence into vertical and horizontal 1D vectors. This reduces the computation cost significantly and it is performed as follows:

$$f_i^H(y) = \sum_{x=1}^{N} f_i(x, y)$$

and

$$f_i^V(x) = \sum_{y=1}^{N} f_i(x, y)$$

Between the row vectors and column vectors of each pair of successive frames, we use the 1D cross-correlation operator to obtain:

$$P_y(z) = \sum_{x=1}^{N} f_i^y(x) f_{i+1}^y(x + z)$$

and

$$P_x(w) = \sum_{y=1}^{N} f_i^x(y) f_{i+1}^x(y + w)$$

Using the vectors of the cross-correlation functions, we compute:

$$\widehat{Hs} = argmax(P_y(z))$$

and

$$\widehat{Vs} = argmax(P_x(w))$$

Where $\widehat{Hs}$ and $\widehat{Vs}$ are respectively the horizontal and vertical pixel shift estimations obtained with the projection-based method.

```
%% Estimate the vertical and horizontal pixel shifts with the projection-based method


%Reduce the 2D video frames into 1D normalized vectors
for frame = 1:numberOfFrames
rowsProjectionVector_shift{frame} = sum(im2bw(VideoData{frame}),2);%This results in a global
column vector
rowsProjectionVector_shift{frame} =
mat2gray(rowsProjectionVector_shift{frame},[min(rowsProjectionVector_shift{frame})
max(rowsProjectionVector_shift{frame})]);

columnsProjectionVector_shift{frame} = sum(im2bw(VideoData{frame}),1);%This results in a global
row vector
columnsProjectionVector_shift{frame} =
mat2gray(columnsProjectionVector_shift{frame},[min(columnsProjectionVector_shift{frame})
max(columnsProjectionVector_shift{frame})]);
end
clear frame

%Estimate the translational pixel shift using the 1D cross-correlation
for frame = 2:numberOfFrames
%Estimate the vertical pixel shift
rowsCrossCorrelationVector_shift{frame} = xcorr(rowsProjectionVector_shift{frame-
1},rowsProjectionVector_shift{frame});
[argValue_row argMax_row] = max(rowsCrossCorrelationVector_shift{frame});
verticalShift(frame) = length(VideoData{frame}(:,1))-argMax_row;

%Estimate the horizontal pixel shift
columnsCrossCorrelationVector_shift{frame} = xcorr(columnsProjectionVector_shift{frame-
1},columnsProjectionVector_shift{frame});
[argValue_column argMax_column] = max(columnsCrossCorrelationVector_shift{frame});
horizontalShift(frame) = length(VideoData{frame}(1,:))-argMax_column;
end
clear frame
```

Similarly to any video stabilization algorithm, using the estimated translational pixel shifts we compensate for the undesired displacements between the frames to achieve video stabilization.

```
%% Compensate for the vertical and horizontal pixel shifts

[height width] = size(VideoData{1});

for frame = 1:numberOfFrames
verticalShiftCompensation(frame) = -sum(verticalShift(1:frame));
horizontalShiftCompensation(frame) =  -sum(horizontalShift(1:frame));
end
clear frame

for frame = 1:numberOfFrames
% Correct the measured vertical and horizontal pixel shifts
correctedVideo_shift{frame} = imtranslate(VideoData{frame},[horizontalShiftCompensation(frame)
verticalShiftCompensation(frame)]);
% Crop the corrected video
croppedVideo_shift{frame} =
correctedVideo_shift{frame}(max(abs(verticalShiftCompensation))+1:height-
max(abs(verticalShiftCompensation)),...
max(abs(horizontalShiftCompensation))+1:width-max(abs(horizontalShiftCompensation)));
end
clear frame
```

After we have corrected for the pixel shifts using the projection-based method, we use the linear interpolation projection-based estimator to eliminate the unwanted sub-pixel shifts. We provide below the Matlab snippet for the estimation and compensation of the sub-pixel shifts. Notice that in order to understand the theory behind our work in this part, the reader will need to refer to methodology in Ratliff's dissertation [2].

```
%% Estimate the vertical and horizontal sub-pixel shifts with the linear interpolation projection-based method


%Reduce the 2D cropped video frames into 1D normalized vectors
for frame = 1:numberOfFrames
rowsProjectionVector_subShift{frame} = sum(croppedVideo_shift{frame},2);%This results in a global column vector
rowsProjectionVector_subShift{frame} = mat2gray(rowsProjectionVector_subShift{frame},[min(rowsProjectionVector_subShift{frame}) max(rowsProjectionVector_subShift{frame})]);

columnsProjectionVector_subShift{frame} = sum(croppedVideo_shift{frame},1);%This results in a global row vector
columnsProjectionVector_subShift{frame} = mat2gray(columnsProjectionVector_subShift{frame},[min(columnsProjectionVector_subShift{frame}) max(columnsProjectionVector_subShift{frame})]);
end
clear frame

%Compute the vertical sub-pixel shift
for frame = 2:numberOfFrames
for i=1:length(rowsProjectionVector_subShift{frame})-1

verticalNumerator(i) = ((rowsProjectionVector_subShift{frame}(i))*((rowsProjectionVector_subShift{frame-1}(i))-(rowsProjectionVector_subShift{frame-1}(i+1))))+...
                    (rowsProjectionVector_subShift{frame-1}(i))*((rowsProjectionVector_subShift{frame-1}(i+1))-(rowsProjectionVector_subShift{frame-1}(i)));

verticalDenominator(i) =(((rowsProjectionVector_subShift{frame-1}(i))^2)-2*((rowsProjectionVector_subShift{frame-1}(i))*(rowsProjectionVector_subShift{frame-1}(i+1)))+...
                    ((rowsProjectionVector_subShift{frame-1}(i+1))^2));

end

estimatedVerticalDeltaAlpha(frame) = -(sum(verticalNumerator))/(sum(verticalDenominator));
if ((estimatedVerticalDeltaAlpha(frame)<-1) || (estimatedVerticalDeltaAlpha(frame)>1))
  estimatedVerticalDeltaAlpha(frame) = 0;
else
estimatedVerticalDeltaAlpha(frame) = round(10*estimatedVerticalDeltaAlpha(frame))/10;

end
end
clear frame i
verticalSubShift = estimatedVerticalDeltaAlpha;
```

```
%Compute the horizontal sub-pixel shift
for frame = 2:numberOfFrames
for i=1:length(columnsProjectionVector_subShift{frame})-1

horizontalNumerator(i) =
((columnsProjectionVector_subShift{frame}(i))*((columnsProjectionVector_subShift{frame-1}(i))-
(columnsProjectionVector_subShift{frame-1}(i+1))))+...
                    (columnsProjectionVector_subShift{frame-
1}(i))*((columnsProjectionVector_subShift{frame-1}(i+1))-
(columnsProjectionVector_subShift{frame-1}(i)));

horizontalDenominator(i) =(((columnsProjectionVector_subShift{frame-1}(i))^2)-
2*((columnsProjectionVector_subShift{frame-1}(i))*(columnsProjectionVector_subShift{frame-
1}(i+1)))+...
                    ((columnsProjectionVector_subShift{frame-1}(i+1))^2));

end

estimatedHorizontalDeltaAlpha(frame) = -(sum(horizontalNumerator))/(sum(horizontalDenominator));
if ((estimatedHorizontalDeltaAlpha(frame)<-1) || (estimatedHorizontalDeltaAlpha(frame)>1))
   estimatedHorizontalDeltaAlpha(frame) = 0;
else
estimatedHorizontalDeltaAlpha(frame) = round(10*estimatedHorizontalDeltaAlpha(frame))/10;
end
end
clear frame i
horizontalSubShift = estimatedHorizontalDeltaAlpha;
```

```
%% Compensate for the vertical and horizontal sub-pixel shifts

[height width] = size(croppedVideo_shift{1});

for frame = 1:numberOfFrames
% Correct the measured vertical and horizontal sub-pixel shifts
correctedVideo_subShift{frame} = imtranslate(croppedVideo_shift{frame},[- orizontalSubShift(frame)
-verticalSubShift(frame)]);
% Crop the corrected video
croppedVideo_subShift{frame} =
correctedVideo_subShift{frame}(round(max(abs(verticalSubShift)))+1:height-
round(max(abs(verticalSubShift)))-1,round(max(abs(horizontalSubShift)))+1:width-
round(max(abs(horizontalSubShift))));
 end
clear frame
```

The last part in our code is dedicated to the obtained results. The first snippet displays the original video as well as its corrected version. The second snippet plots the graphs of the pixel and sub-pixel shifts that we measured using the projection-based and linear interpolation projection-based estimators. The reader can watch the results on Youtube, by clicking on the videos below.

Uncorrected Video

Corrected Video

Cropped Corrected Video

```
%% display the input and output videos

figure('units','normalized','outerposition',[0 0 1 1])
subplot(2,2,1)
pause(3)
% Original video
for frame = 1:numberOfFrames
    imshow(VideoData{frame})
    str = ['Input video - Frame ',num2str(frame)];
    title(str)
    pause(0.0)
end
clear frame str

% Corrected video
subplot(2,2,2)
for frame = 1:numberOfFrames
    imshow(correctedVideo_shift{frame})
    str = ['Corrected video - Frame ',num2str(frame)];
    title(str)
    pause(0.0)

end
clear frame str

% Cropped corrected video
subplot(2,2,3)
for frame = 1:numberOfFrames
    imshow(croppedVideo_shift{frame})
    str = ['Output video - Frame ',num2str(frame)];
    title(str)
    pause(0.0)

end
clear frame str

% Cropped corrected video after correcting for the sub-pixel shifts
subplot(2,2,4)
for frame = 1:numberOfFrames
    imshow(croppedVideo_subShift{frame})
    str = ['Output video (sub-pixel correction) - Frame ',num2str(frame)];
    title(str)
    pause(0.0)

end
clear frame str
```

```
%% display the graphs of the estimated values

figure('units','normalized','outerposition',[0 0 1 1])

%Estimated pixel shifts
subplot(1,2,1)
plot(verticalShift,'r','LineWidth',2)
hold on
plot(horizontalShift,'b','LineWidth',2)
grid on
xlim([0 numberOfFrames+20])
vec = [min(verticalShift) min(horizontalShift) max(verticalShift) max(horizontalShift)];
ylim([-5+min(vec) 5+max(vec)])
set(gca,'fontsize',16);
xlabel('Number of Frames','FontWeight','bold')
ylabel('Measured shift (Pixel)','FontWeight','bold')
title('Pixel shift estimation between each pair of adjacent frames')
legend('Vertical pixel shift','Horizontal pixel shift','Location','northwest')

%Estimated sub-pixel shifts
subplot(1,2,2)
plot(verticalSubShift,'r','LineWidth',2)
hold on
plot(horizontalSubShift,'b','LineWidth',2)
grid on
xlim([0 numberOfFrames+20])
ylim([-1.1 1.1])
set(gca,'fontsize',16);
xlabel('Number of Frames','FontWeight','bold')
ylabel('Measured shift (Sub-Pixel)','FontWeight','bold')
title('Sub-pixel shift estimation between each pair of adjacent frames')
legend('Vertical sub-pixel shift','Horizontal sub-pixel shift','Location','northwest')
```

Conclusion:

In this work, we have used projection-based algorithms to estimate and correct for translational pixel and sub-pixel shifts. The used algorithms reduce the computation time significantly and provide high estimation accuracy. These algorithms do not account for rotational shifts, scaling and warping. Future work must be carried to correct for all sorts of undesired motions to ensure a more effective video stabilization process.

References:

[1] Cain, Stephen C., Majeed M. Hayat, and Ernest E. Armstrong. "Projection-based image registration in the presence of fixed-pattern noise." IEEE transactions on

[2] Ratliff, Bradley Michael. "A generalized algebraic scene-based nonuniformity correction algorithm for infrared focal plane arrays." University of New Mexico, Albuquerque, NM (2004).