Panorama by Manual Correspondences and Gradient Descent Optimization

EE475: Computer Vision

Muhammad Usman

BEE8-D | *194483*

muusman.bee16seecs

Contents

1 Introduction

Image-stitching is not a trivial task and is still an active area of research. This documentation outlines implementation of the most plain form of stitching.

2 Functional Description

There are two buttons to upload images such that they can be stitched together as in figure 1. By pressing "Select Correspondece", points are selected in each image to be used for computing projection matrix. An image representing error in the same correspondences for forward as well as backward projection is also shown such that quality of panorama may be assumed and corrections applied if necessary. Once the panorama is presented, it can be used again to stitch yet another image.

3 Results

A typical outcome can be observed in figure 2. Uploaded images along with their projection errors can be observed. Red color is that of calculated projections and green color dots imply projections after optimization has been applied. Below is the final image obtained after stitching together the two images.

4 Implementation

4.1 State Diagram

State diagram for the implementation can be observed in figure 3.

4.2 Application

The application layer on a higher layer is divided in only two tasks namely:

- Calibration
- Projection

The same can be visually observed in flowchart 4.

4.3 Calibration

Calibration is achieved by letting the user view images. User selects points and press `enter` for both images. The projection matrix is then calculated and error in projection is visually presented. Moreover, gradient descent optimization is also applied and projections resulting from it are also presented. Flowchart in figure 5 represents implementation of code snippet of appendix A.1

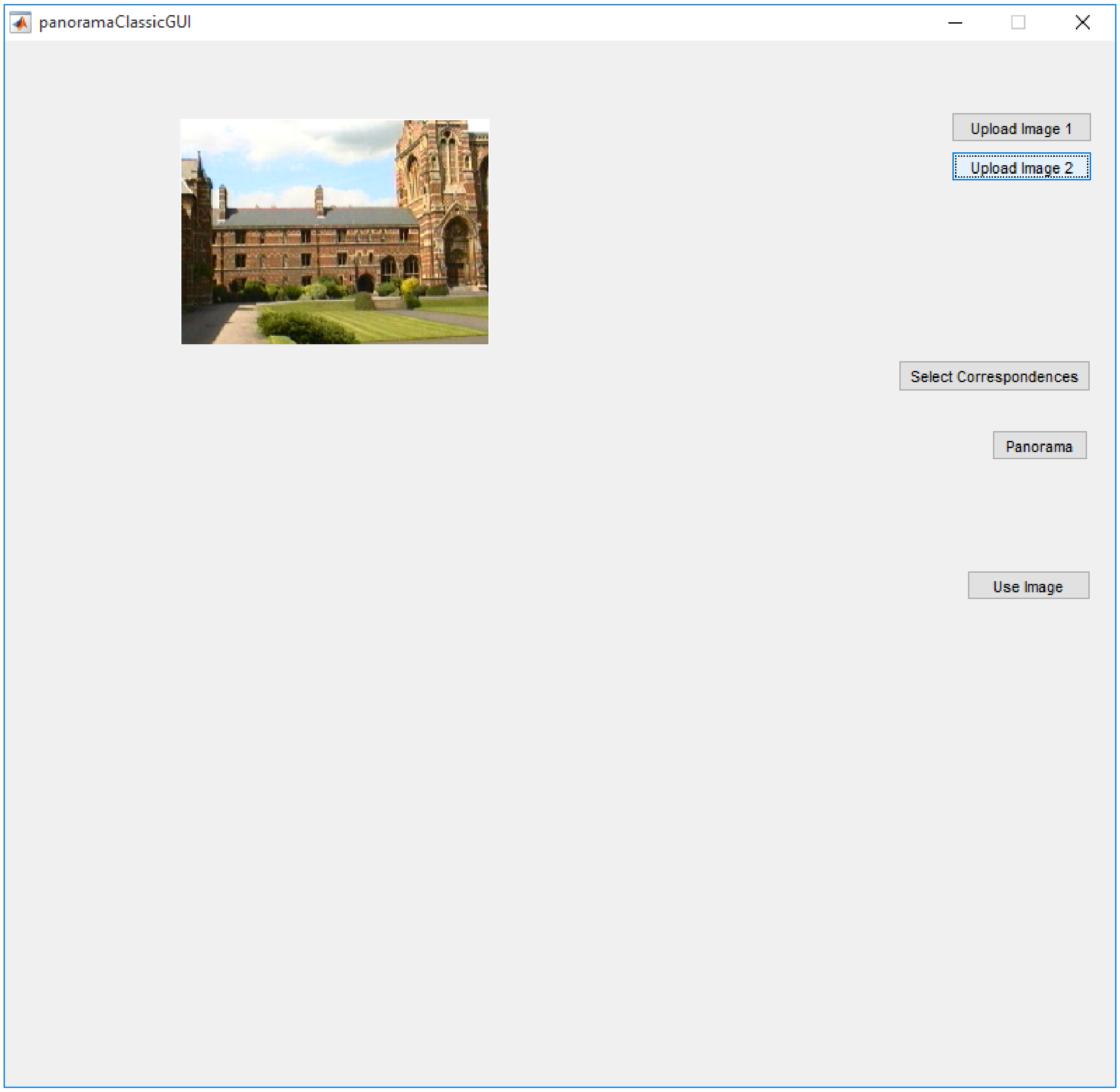

Figure 1: A GUI layout for classic panorama application.

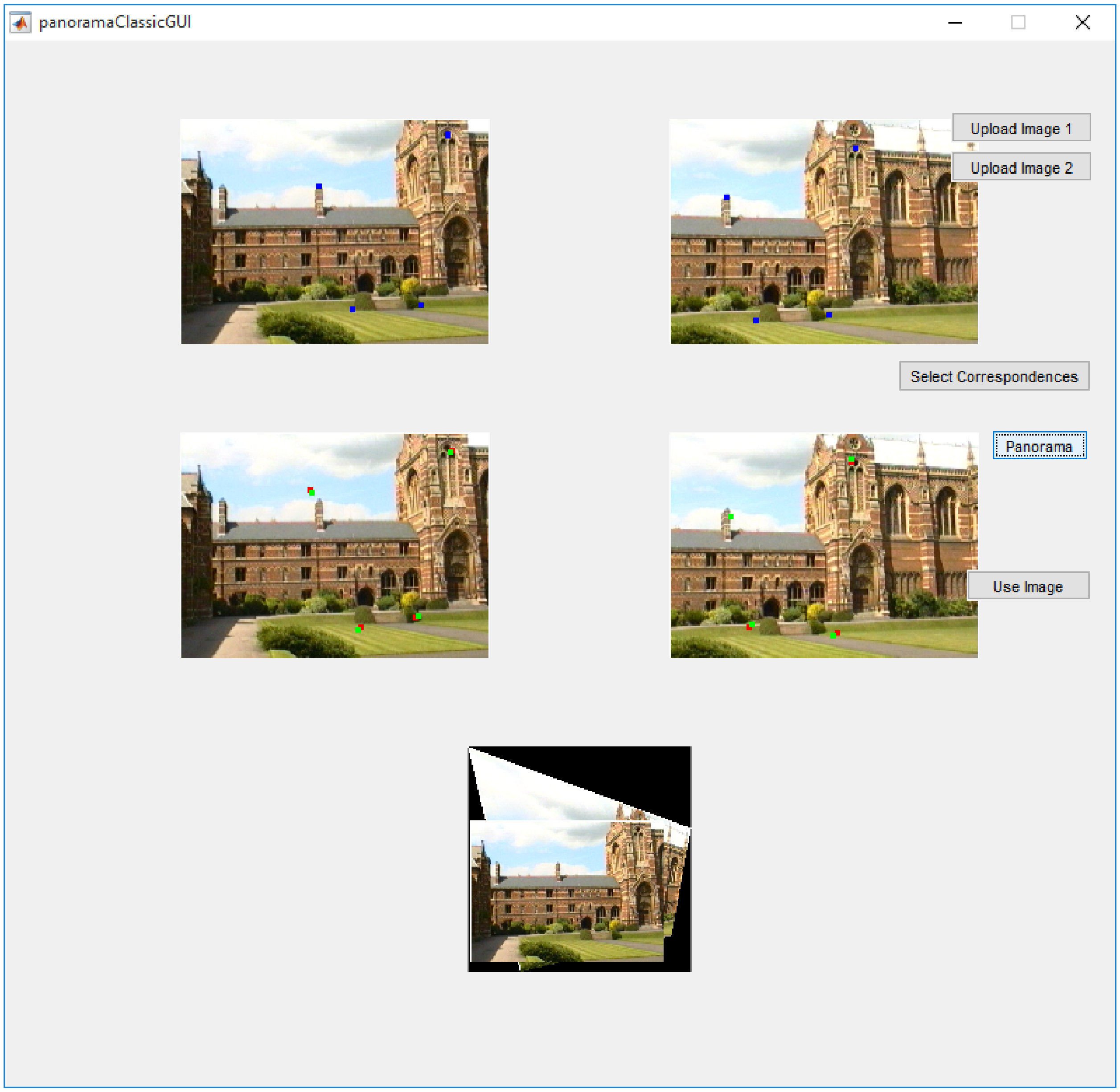

Figure 2: A typical result of classic panorama application

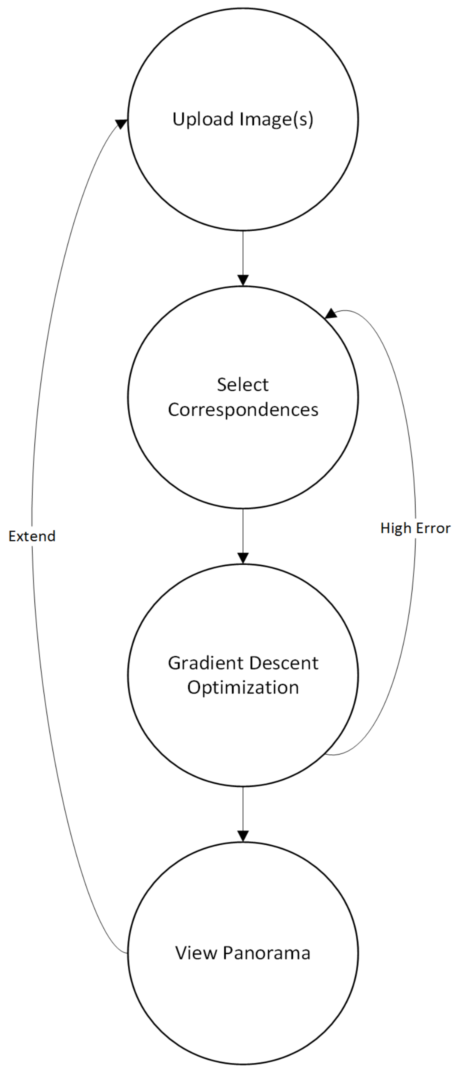

Figure 3: State diagram for classic panorama implementation.

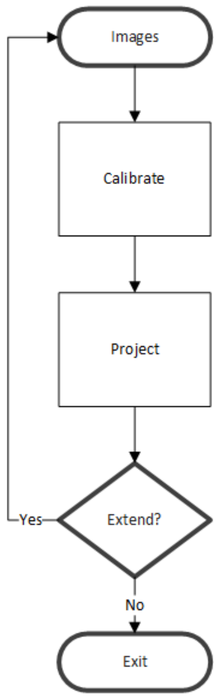

Figure 4: Application layer flowchart of classic panorama implementation.

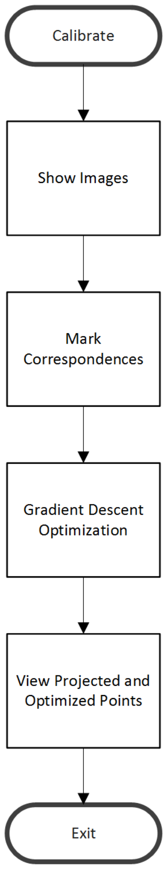
Calibrate
Show Images
Mark Correspondences
Gradient Descent Optimization
View Projected and Optimized Points
Exit

Figure 5: Flowchart for calibration of panorama in implementation.

4.3.1 Point Selection

Points for as many correspondences as user may enter are selected through `getpts` function. Refer to code snippet in appendix A.2 for implementation.

4.3.2 Projection Matrix

A matrix of form

$$\begin{bmatrix} 0 & 0 & 0 & -x_i & -y_i & -1 & y^{'}x_i & y^{'}y_i & y^{'} \\ x_i & y_i & 1 & 0 & 0 & 0 & x^{'}x_i & x^{'}y_i & x^{'} \end{bmatrix}$$

is repeated for as many correspondences available. This matrix is solved using SVD method. For implementation details refer to appendix A.3.

4.3.3 Mapping Projections

A generic function is used for projecting points in forward as well as backward direction. Refer to appendix A.4 for details.

4.3.4 Gradient Descent Optimization

Gradient descent optimization has been applied in calibration phase using following cost functions:

$$\delta = \frac{\Sigma(x_i - \widehat{x_i})^2 + (x_i' - \widehat{x_i'})^2}{N}$$

where x_i and x_i' are points of two image planes, $\widehat{x_i} and \widehat{x_i'}$ are points projected from the other plane and N is the total number of points.

Refer to appendix A.6 for implementation.

4.4 Projection

Corner points are first mapped to guess the bounding box. Once achieved, only required projections are mapped to it, the other image is pasted otherwise as flowchart in figure 6 suggests. Refer to appendix A.5 for implementation.

4.4.1 Bounding Dimensions

Dimensions bounding the new panorama are evaluated by using corner points and projection of corner points of other image. Refer to appendix A.7 for implementation.

4.4.2 Inverse Projection

The technique of inverse projection is being exploited for obtaining a consistent image. Points from new image are iterated over using projections of corner points available. These points are inverse projected to image for obtaining colors. Refer to figure 7 and appendix A.8.

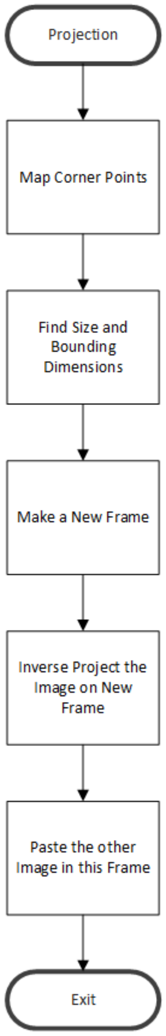
Projection
Map Corner Points
Find Size and Bounding Dimensions
Make a New Frame
Inverse Project the Image on New Frame
Paste the other Image in this Frame
Exit

Figure 6: Projection routine for implementation of classic panorama algorithm

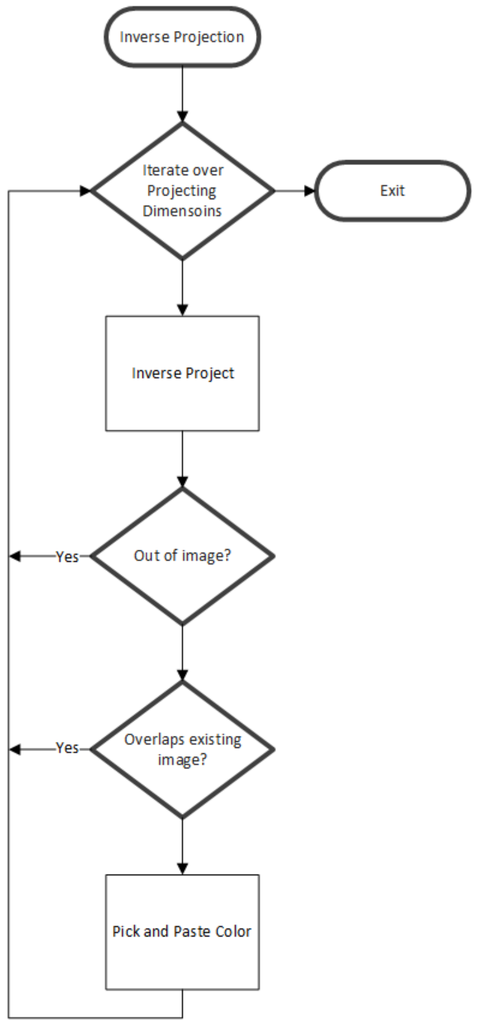

Figure 7: Flowchart for inverse projection of image on the new plane

4.4.3 Shift & Paste

Plain image is pasted over a predefined offset governed by bounding dimensions. Refer to appendix A.9 for implementation.

A Code Snippets

A.1 Calibration

```matlab
function [ H_ab, H_ba ] = calibratePanorama( im_a, im_b )
%calibratePanorama Calculate projection matrices for input images.

subplot(3,2,1)
ptsA = selectPoints();
imPtsA = markPt(im_a, ptsA, 'blue');
imshow(uint8(imPtsA));

subplot(3,2,2);
ptsB = selectPoints();
imPts_B = markPt(im_b, ptsB, 'blue');
imshow(uint8(imPts_B));

[H_ab, H_ba] = projMatrix(ptsA, ptsB);


ptsBA = mapPt(H_ba, ptsB);
ptsAB = mapPt(H_ab, ptsA);

imPtsProj_ba = markPt(im_a, ptsBA, 'red');
imPtsProj_ab = markPt(im_b, ptsAB, 'red');

options = optimset('GradObj', 'off', 'Maxiter', 400, 'Display', 'iter');
[H, cost] = fminunc(@(t)costHomography(t,ptsA,ptsB), H_ab, options);
H_ab = H;
H_ba = inv(H);


ptsBA = mapPt(H_ba, ptsB);
ptsAB = mapPt(H_ab, ptsA);

imPtsProj_ba = markPt(imPtsProj_ba, ptsBA, 'green');
imPtsProj_ab = markPt(imPtsProj_ab, ptsAB, 'green');

subplot(3,2,3);
imshow(uint8(imPtsProj_ba));
subplot(3,2,4);
imshow(imPtsProj_ab);

end
```

A.2 Select Points

```matlab
function [ pts ] = selectPoints()
%selectPoints Selects points for correspondences.

[pts_y, pts_x] = getpts();

pts = round([pts_x pts_y]);

end
```

A.3 Projection Matrix

```matlab
function [ H_ab, H_ba ] = projMatrix( pt_a, pt_b )
%projMatrix Evaluates forward and backward projection
%           Matrix for given points.


eqMat = zeros(2*length(pt_b),9);

for i = 1 : length(pt_b)
    eqMat(2*i-1,4:6) = -1.*[pt_b(i,:) 1];
    eqMat(2*i-1,7:9) = pt_a(i,2).*[pt_b(i,:) 1];
    eqMat(2*i,1:3) = [pt_b(i,:) 1];
    eqMat(2*i,7:9) = -pt_a(i,1).*[pt_b(i,:) 1];
end

[~,S,V] = svd(eqMat);

nullNum = 0;
for i = 1 : min(size(S))
   if (round(S(i,i),0)==0)
        nullNum = i;
        break;
   end
end

if ~nullNum
    fprintf('Cannot determine nullspace from SVD.\n\r');
else
    fprintf('The null space is %d column of V*\n\r', nullNum);
end

nullSpace = V(:,nullNum);

H_ba = vec2mat(nullSpace,3);
H_ab = inv(H_ba);

end
```

A.4 Mapping Projections

```
function [ ptOut ] = mapPt( H, pt )
%mapPt Maps Points to the required proejction plane.
ptOut = zeros(length(pt),2);

for i = 1:length(pt)
    tmp0 = H*[pt(i,:) 1]';
    tmp0 = tmp0./tmp0(3);
    ptOut(i,:) = round([tmp0(1) tmp0(2)]);
end
```

A.5 Projection

```
function [ im_ba, im_ab ] = classicPanorama( im_a, im_b, H_ab, H_ba)
%classicPanorama Projects one image plane to the other using
%                projection matrices.

cornerPts_a = cornerDim( im_a );
cornerPts_b = cornerDim( im_b );

mapCornerPts_ba = mapPt(H_ba, cornerPts_b);
mapCornerPts_ab = mapPt(H_ab, cornerPts_a);

[dims_ba, imProjSize_ba] = boundingDims(mapCornerPts_ba, cornerPts_b);
[dims_ab, imProjSize_ab] = boundingDims(mapCornerPts_ab, cornerPts_a);

im_ba = zeros([imProjSize_ba 3]);
im_ab = zeros([imProjSize_ab 3]);

dimsOut_a = size(im_a);
dimsOut_b = size(im_b);

im_ba = invProj(im_ba, im_b, H_ab, dims_ba, dimsOut_a);
im_ba = imShift(im_ba, im_a, [dims_ba(1,1), dims_ba(2,1)]);


im_ab = invProj(im_ab, im_a, H_ba, dims_ab, dimsOut_b);
im_ab = imShift(im_ab, im_b, [dims_ab(1,1), dims_ab(2,1)]);

end
```

A.6 Cost Function

```
function [ meanSqError ] = costHomography(H_ab,ptsA,ptsB)
%COSTHOMOGRAPHY Optimize H_ab for correspondences of
%               image A, ptsA and of image B, ptsB.

% map points to the other images
ptsAB = mapPtReal(H_ab, ptsA);
ptsBA = mapPtReal(inv(H_ab), ptsB);

% finding the squared difference.
```

```matlab
meanSqError = (ptsBA - ptsA).^2 + (ptsAB - ptsB).^2;
meanSqError = sum((sum(meanSqError)));

end
```

A.7 Bounding Dimensions

```matlab
function [ dims, imProjSize ] = boundingDims( mapCornerPts, cornerPts )
%boundingDims Evaluate corner points using corner points
%             and mapped corner points.

minHeight = min(mapCornerPts(1,1), mapCornerPts(4,1));
minHeight = min(minHeight, cornerPts(1,2));
maxHeight = max(mapCornerPts(2,1), mapCornerPts(3,1));
maxHeight = max(maxHeight, cornerPts(3,1));
minWidth = min(mapCornerPts(1,2), mapCornerPts(2,2));
minWidth = min(minWidth, cornerPts(1,1));
maxWidth =  max(mapCornerPts(3,2), mapCornerPts(4,2));
maxWidth = max(maxWidth, cornerPts(3,2));
imProjSize = [maxHeight - minHeight, maxWidth - minWidth];

dims = [minHeight, maxHeight; minWidth, maxWidth];

end
```

A.8 Inverse Projection

```matlab
function [ imOut ] = invProj( imFrame, im, H, dims, dimsOut )
%invProj Project image on other plance using projection matrix
%        according to dimensions and sizes.

[imHeight, imWidth, ~] = size(im);

for i = dims(1,1) : dims(1,2)
    for c = dims(2,1) : dims(2,2)
        projPos = H*[i; c; 1];
        projPos = round(projPos./projPos(3),0);

        if (projPos(1) <= 0 || projPos(2) <= 0 || projPos(1) >= imHeight ||
    projPos(2) >= imWidth)
            continue;
        end
        if (i <= dimsOut(1) && c <= dimsOut(2) && i >= 1 && c >= 1)
            continue;
        end
        imFrame(i-dims(1,1)+1, c-dims(2,1)+1,:) = im(projPos(1),projPos(2),:);
    end
end

imOut = imFrame;

```

```
end
```

A.9 Shift & Plase

```
function [ imOut ] = imShift( imOut, im, shift )
%imShift Shift image and paste it over defined offset
%        in the new frame.

[imHeight, imWidth, ~] = size(im);

for i = 1:imHeight
    for c = 1:imWidth
        imOut(i-shift(1)+1, c-shift(2)+1,:) = im(i,c,:);
    end
end

end
```