1 Appendix: Code Listings

1.1 File: main.m

```
% Main

% Initialize data locations
imagesDir = 'data/bindermat';
backgroundDir = 'data/field.jpg';
videoDir = 'data/video';

% manually selected corners of the overlaying background image
backgroundCorners=[[40,182]',[39,429]',[474,453]',[474,155]']';

% Loading data
images = readData(imagesDir);
backgroundImage = double(imread(backgroundDir));

[width,height,nChannels,nImages] = size(images);

% Transforming images so that they are displayed correctly
images = transformData(images);
earlyFrame = images(:,:,:,2);

writeImgs(images,1:nImages, 'original');

% Find a big patch for estimating the plane based on the four corners
sizePatch= 100;
middlePoint = round(mean(backgroundCorners));
xRange = [middlePoint(1)-sizePatch, middlePoint(1)+sizePatch];
yRange = [middlePoint(2)-sizePatch, middlePoint(2)+sizePatch];
xSize = xRange(2) - xRange(1) + 1;
ySize = yRange(2) - yRange(1) + 1;

% Compute Background overlaying plane
initialPoints = earlyFrame(xRange(1):xRange(2),yRange(1):yRange(2),1:3);
initialPoints = reshape(initialPoints(:,:,1:3), xSize*ySize, 3);

[plane, fit] = fitplane(initialPoints);


% Iterate over the images
for i=1: nImages

    frame = i

    % In case of a very improbable event of not finding the best fitting
    % plane, while there is one, run the algorithm again
    for j=1:1
        attempt = j
        [ quadPoints, suitcasePlane ] = planeExtraction(images(:,:,:,i));
        if isSuitcase(quadPoints)
            break;
        end
    end

    if isSuitcase(quadPoints)
        % find ordered corners of the suitcase
        suitcaseCorners = getCorners(quadPoints);
        pixelVals = getPixelVals(suitcaseCorners, images(:,:,:,i));
        orderedCorners = orderCorners(pixelVals);

        % Load corresponding video frame
        listing = dir( videoDir );
        imagePath = strcat(videoDir, '/', listing( i+3 ).name);
        videoFrame = double(imread(imagePath));

        % Remap video on the suitcase
        images(:,:,:,i) = remap(videoFrame, images(:,:,:,i), suitcasePlane, ↩
            orderedCorners, 0.02);
    end

```

```
    writeImgs(images,i, 'quad');

    % Make pixels without any value into background pixels
    images(:,:,:,i) = fillMissingVals(images(:,:,:,i), earlyFrame(:,:,4:6));
    writeImgs(images,i, 'fill');

    % Remap field image as background image
    UV=backgroundCorners;    % target points
    images(:,:,:,i) = remap(backgroundImage, images(:,:,:,i), plane, UV, 0.05);

    imshow(images(:,:,4:6,i));
end

% create the AV.avi video of all the frames
generateVideo(images);

% write all the final images to the output folder
writeImgs(images,1:nImages, 'final');
```

1.2 File: planeExtraction.m

```
function [ oldlist, plane ] = planeExtraction( image )
% Finds a plane with the lowest residual errror using a RANSAC method.
% First finds a candidate patch, which it grows until it satisfies the
% planar property, or there are no more points to add.


im3d = image(:,:,1:3);
imRGB= image(:,:,4:6);

% Two thresholds, strict for initial points and relaxed for all the points
initialPtsBinary = hsvThresh(imRGB, 0.8) & hasRangeData(im3d);
inclusiveBagPtsBinary = hsvThresh(imRGB, 1.6) & hasRangeData(im3d);

initialPts = im3d(repmat(initialPtsBinary,[1 1 3]));
inclusiveBagPts = im3d(repmat(inclusiveBagPtsBinary,[1 1 3]));

R = reshape(im3d, 640*480,3); % 3d points
initialPts = reshape(initialPts, length(initialPts)/3,3);
inclusiveBagPts = reshape(inclusiveBagPts, length(inclusiveBagPts)/3,3);

[NPts, ~] = size(R);

remaining = inclusiveBagPts;

% select a random small surface patch
[oldlist,plane] = select_patch(initialPts);

% grow patch
stillgrowing = 1;

DISTTOL  = 0.01;            % grow slowly to avoid jumping to another plane
PLANETOL = 0.01;

pointsAdded = size( oldlist, 1 );

while stillgrowing

% find neighbouring points that lie in plane
  stillgrowing = 0;
  [newlist,remaining] = getallpoints(plane,oldlist((end-pointsAdded + 1):end,:), ↩
      remaining,NPts,DISTTOL,PLANETOL);
  newlist = [ oldlist ; newlist ];

  [NewL,W] = size(newlist);
  [OldL,W] = size(oldlist);

  pointsAdded = NewL - OldL;


  if (pointsAdded > 10) % TODO: look at this param
```

```
    % refit plane
    [newplane,fit] = fitplane(newlist);
    if fit > 0.04*NewL       % bad fit - stop growing
      break
    end
    stillgrowing = 1;
    oldlist = newlist;
    plane = newplane;
  end
end

end
```

1.3 File: select_patch.m

```
function [fitlist,plane] = select_patch(points)
% Finds a candidate planar patch.
% Iteratively increases residual error tolerance until a good fit is found.

[L,D] = size(points);
tmpnew = zeros(L,3);

initialErrTol = 0.01;
maximumErrTol = 0.04;
nSteps        = 500;

residErrTol   = initialErrTol;
stepSize      = (maximumErrTol - initialErrTol) / nSteps;

% pick a random point until a successful plane is found or no plane can
% be found (allow for 500 random point selections)
for nStep = 1:nSteps
    idx = floor((L-1)*rand)+1;   %BUGFIX: avoiding access of unexisting elements
    pnt = points(idx,:);


    % find points in the neighborhood of the given point
    DISTTOL = 0.08;
    fitcount = 0;
    for i = 1 : L
      dist = norm(points(i,:) - pnt);
      if dist < DISTTOL
        fitcount = fitcount + 1;
        tmpnew(fitcount,:) = points(i,:);
      end
    end

    if fitcount > 1500
      % fit a plane
      [plane,resid] = fitplane(tmpnew(1:fitcount,:));

      if resid < residErrTol
        fitlist = tmpnew(1:fitcount,:);
        return
      end
    end

    %increase allowed error tolerance
    residErrTol = residErrTol + stepSize;
end

% no plane found - return points which will fail.
fitlist=points(1:5,:);
plane=[1; 1; 1; 1];
```

1.4 File: getallpoints.m

```
function [newlist,remaining] = getallpoints(plane,oldlist,P,NP, DISTTOL,PLANETOL)
% selects all points in pointlist P that fit the plane and are within
% TOL of points added newly to the list of points in the plane.

  pnt = ones(4,1);
  [N,W] = size(P);
  [Nold,W] = size(oldlist);

  tmpnewlist = zeros(NP,3);
  tmpremaining = zeros(NP,3);           % initialize unfit list
  countnew = 0; %Nold;
  countrem = 0;

  for i = 1 : N
    pnt(1:3) = P(i,:);
    notused = 1;


    if abs(pnt'*plane) < PLANETOL
        % speed optimisation of vectorising the for loop was suggested
        % by Cristian Cobzarenco
        if any( sum((oldlist - repmat( P(i,:), Nold,1)) .^ 2, 2 ) < DISTTOL^2 )
            countnew = countnew + 1;
            tmpnewlist(countnew,:) = P(i,:);
            notused = 0;
        end
    end

    if notused
      countrem = countrem + 1;
      tmpremaining(countrem,:) = P(i,:);
    end
  end

  newlist = tmpnewlist(1:countnew,:);
  remaining = tmpremaining(1:countrem,:);
```

1.5 File: getCorners.m

```
function [ corners3d ] = getCorners( quadPts )
% This function gets the corners of a quad given a list of points. It finds
% the point which is further away from the centroid, and then removes all
% the points which are closer to that corner than to the centroid. Then it
% repeats this three more times.

corners3d = zeros(4,3);
nPts = length(quadPts);
centroid = mean(quadPts, 1);

dists3 = abs(quadPts - repmat(centroid, [nPts 1]));
dists = get3dLen( dists3 );


for i= 1: 4
    [~, idx] = max(dists);
    corners3d(i,:) = quadPts(idx, :);

    distsCor3 = abs(quadPts - repmat(corners3d(i,:), [length(quadPts) 1]));
    distsCor = get3dLen( distsCor3 );

    L = repmat(dists < distsCor, [1 3]);

    quadPts = quadPts( L );
    quadPts = reshape(quadPts,length(quadPts)/3, 3);

    dists3 = abs(quadPts - repmat(centroid, [length(quadPts) 1]));
    dists = get3dLen( dists3 );
end

end
```

1.6 File: orderCorners.m

```
function [ outputCorners ] = orderCorners( inputCorners )
% This function gets a list 4 of corners and using their angle from the
% centre of the quad, sorts them from top left to bottom right (clockwise).

corners(:,1) = inputCorners(:,2);
corners(:,2) = inputCorners(:,1);

Cx = sum(corners(:,1)) / 4;
Cy = sum(corners(:,2)) / 4;

centre = [Cx, Cy];
centre = repmat(centre, 4, 1);

tmp = corners - centre;

angles = atan2(tmp(:,1),tmp(:,2));
[sortAngles, idx] = sort(angles,'descend');

almostOrderedCorners = corners(idx,:);
orderedCorners(1,:) = almostOrderedCorners(4,:);
orderedCorners(2:4,:) = almostOrderedCorners(1:3,:);

outputCorners(:,1) = orderedCorners(:,2);
outputCorners(:,2) = orderedCorners(:,1);

end
```

1.7 File: hsvThresh.m

```
function vThresh = hsvThresh(inImage, val)

% Performs Value Tresholding on a given image relative to the average
% brightness.

% Convert the image to the HSV space
hsvImage = rgb2hsv(inImage);

% Get the value plane and threshold
vPlane = hsvImage(:,:,3);
valThresh = val * mean(mean(vPlane))^2;

% Find indeces that will be removed based on thresholding
vThresh = (vPlane < valThresh ) & (vPlane>0);

end
```

1.8 File: remap.m

```
function [ finalImage ] = remap( inimage,image, plane, UV, DISTTOL )
% Remaps inimage onto the quad in image. UV specifies the four corners  of
% the quad in the clockwise order. DISTTOL specifies the distance tolerance
% from the plane of the quadrilateral.

outimage = image(:,:,4:6);


% get sizes of the input and output images
[IR,IC,D]=size(inimage);
[OR,OC,D]=size(outimage);

% corners of the input image
XY=[[1,1]',[1,IC]',[IR,IC]',[IR,1]']';    % source points

```

```
P=esthomog(UV,XY,4);    % estimate homography mapping UV to XY

pnt = zeros(4,1);
pnt(4) = 1;

% loop over all pixels in the destination image, finding
% corresponding pixel in source image
for r = 1 : OR
    for c = 1 : OC
      v=P*[r,c,1]';        % project destination pixel into source
      y=round(v(1)/v(3));  % undo projective scaling and round to nearest integer
      x=round(v(2)/v(3));

      pnt(1:3) = image(r,c,1:3);

      if (x >= 1) && (x <= IC) && (y >= 1) && (y <= IR) && ...
              (abs(pnt'*plane) < DISTTOL || pnt(3)==0 )
        outimage(r,c,:)=inimage(y,x,:)/255;   % transfer colour
      end
    end
end

finalImage(:,:,1:3) = image(:,:,1:3);
finalImage(:,:,4:6) = outimage(:,:,1:3);

end
```

1.9 File: fillMissingVals.m

```
function [ image6d ] = fillMissingVals( image6d, bcg )
% Replace the missing colour values (range == 0) with the background values.
% Background pixel values are estimated using an early frame normalised for the
% mean brightness (HSV Value).

img = image6d(:,:,4:6);
    I =repmat(image6d(:,:,3)==0, [1 1 3]);    % distance (z-value) == 0

    img(I) = bcg(I) * mean(mean(mean(img))) / mean(mean(mean(bcg)));
    image6d(:,:,4:6) = img;

end
```

1.10 File: getRange.m

```
function [ points3d ] = getRange( img, points )
% Given a list of pixel indeces returns a corresponding list of 3d points.

[nPoints, nDims] = size(points);
points3d = zeros(nPoints,nDims+1);

for i=1:nPoints
    points3d(i,:) = img(points(i,1), points(i,2), 1:3);
end

end
```

1.11 File: hasRangeData.m

```
function [ binary ] = hasRangeData( pts3d )
% Returns a logical of all the points which have some range data.

ranges = pts3d(:,:,3);
binary = ranges ~= 0;
```

```
6
7 end
```

1.12 File: normRGB.m

```
1 function normImage = normRGB(image)
2 % gets the RGB values in the [0,1] range.
3
4 normImage(:,:,1:3) = image(:,:,1:3);
5 normImage(:,:,4:6) = image(:,:,4:6) / 255;
6 end
```

1.13 File: get3dLen.m

```
1 function [ dists ] = get3dLen( dists3 )
2 % Finds the 3d length for all the points
3
4 dists = dists3(:,1).^2 + dists3(:,2).^2 + dists3(:,3).^2;
5
6 end
```

1.14 File: getPixelVals.m

```
1 function [ corners ] = getPixelVals( corners3d, image )
2 % Given the corners of the quad in 3D space it finds corresponding 2d pixel
3 % values of each corner.
4
5 [r1,c1] = find((image(:,:,1) == corners3d(1,1)) & (image(:,:,2) == corners3d(1,2)) & (↩
      image(:,:,3) == corners3d(1,3)));
6 [r2,c2] = find((image(:,:,1) == corners3d(2,1)) & (image(:,:,2) == corners3d(2,2)) & (↩
      image(:,:,3) == corners3d(2,3)));
7 [r3,c3] = find((image(:,:,1) == corners3d(3,1)) & (image(:,:,2) == corners3d(3,2)) & (↩
      image(:,:,3) == corners3d(3,3)));
8 [r4,c4] = find((image(:,:,1) == corners3d(4,1)) & (image(:,:,2) == corners3d(4,2)) & (↩
      image(:,:,3) == corners3d(4,3)));
9
10 corners(1,1:2) = [r1,c1];
11 corners(2,1:2) = [r2,c2];
12 corners(3,1:2) = [r3,c3];
13 corners(4,1:2) = [r4,c4];
14
15 end
```

1.15 File: isSuitcase.m

```
1 function [ isSuitcase ] = isSuitcase( pts3d )
2     % This function simply checks if a set of points in 3D space is large
3     % enough to be assumed to be the correct quad (suitcase of the man).
4
5     nPoints = length(pts3d);
6     nPointsTest = nPoints > 10000;
7     % residual error of the plane is already checked
8
9     isSuitcase = nPointsTest;
10
11 end
```

1.16 File: readData.m

```
function [ images ] = readData( path )
% Loads a sequence of images from a given 'path' (a string specifying a
% path to a folder).

listing = dir( path );

nImages    = size( listing, 1 ) - 3;
images     = zeros( 640, 480, 6, nImages);


for i = 1 : nImages
    var=importdata( [ path '/' listing( i+3 ).name ] );
    images( :, :, :, i ) = reshape(var, 640,480,6);
end

end
```

1.17 File: transformData.m

```
function [ images ] = transformData( imgs )
% Normalises the images (so that RGB is in double as well) and transposes
% the data.

[width, height, channels, nImages] = size(imgs);
images = zeros(height, width, channels, nImages);

for i=1: nImages

    im = normRGB(imgs(:,:,:,i));
    for j=1: channels
        images(:,:,j,i) = im(:,:,j)';
    end

end

end
```

1.18 File: generateVideo.m

```
function generateVideo(images)

    % Generates a video from the individual frames

    vw = VideoWriter('AV_movie.avi');
    vw.FrameRate = 6;
    vw.open();

    for i = 1:36
        image = images(:,:,4:6,i);
        imshow(image);
        writeVideo(vw,getframe(gcf));
    end

    close(vw);
end
```

1.19 File: writeImgs.m

```
function [ ] = writeImgs( images, range, fname )
%Writes the images in the output folder
```

```
3
4  for i=range
5      imwrite(images(:,:,4:6,i), ['output/',int2str(i),fname,'.jpg'] );
6  end
7
8
9  end
```