MTRX5700

Experimental Robotics

Assignment 2

Author(s):
Kausthub Krishnamurthy
James Ferris
Sachith Gunawardhana

SID:
312086040
311220045
440623630

Due: March 25, 2015

1 Question 1

1.a Perpendicular Distance Function

```matlab
%% PERPDIST - Perpendicular Distance Calculator
%
% Usage:  perpDist = perpdist(p_x, p_y, grad, y_int, x_int, vertFlag)
%
% Arguments:
%             p_x     - x coordinate of point
%             p_y     - y coordinate of point
%             grad    - gradient of line
%             y_int   - y intercept of the line
%             x_int   - x intercept of the line
%             vertFlag - 1 if the line is vertical; 0 if line is not
%
% Returns:
%             perpDist - perpendicular distance between the point (p_x, p_y)
%                        and the line y = grad*x + y_int
%
%
% Author:
% Kausthub Krishnamurthy
% The University of Sydney
% kkri3182@uni.sydney.edu.au
% April 2015

function perpDist = perpdist(p_x, p_y, grad, y_int, x_int, vertFlag)
    if vertFlag == 1
        %x-distance between mean of xvalues and point's x coordinate
        perpDist = abs(p_x-x_int);
    else
        %if not vertical use the y=mx+b form of the equation:
        % pdist = |y - mx - b|/sqrt(m^2 + 1)
        num = abs(p_y-(grad*p_x)-y_int);
        den = sqrt(grad*grad + 1);
        perpDist = num/den;
    end

end
```

1.b Least Square Minimization Algorithm

```
%% LEASTSQMIN -  Least Squares Minimisation Algorithm
%
% Usage:  lineDetails = leastsqmin(xpoint,ypoint, iStart, iEnd)
%
% Arguments:
%            xpoint - set of x coordinates
%            ypoint - set of y coordinates
%            iStart - starting index
%            iEnd   - ending index
%
% Returns:
%            lineDetails - array of 4 values
%               (1) gradient of estimated line
%               (2) y-intercept of estimated line
%               (3) x-intercept of estimated line
%               (4) flag value 1 or 0 (1 if line is vertical; 0 if not)
%
%
% Author:
% Kausthub Krishnamurthy
% The University of Sydney
% kkri3182@uni.sydney.edu.au
% April 2015

function lineDetails = leastsqmin(xpoint,ypoint, iStart, iEnd)

    tempX = xpoint(iStart:iEnd);
    tempY = ypoint(iStart:iEnd);

    Mx=mean(tempX);

    My=mean(tempY);

    xy=times(tempX,tempY);
    Mxy=mean(xy);

    xx=times(tempX,tempX);
    Mxx=mean(xx);

    %assebles our matrices A and Vector x
    A_mat = [Mxx , Mx; Mx, 1];
    x_vec = [Mxy; My];

    %finds the determinant of A (if this is zero ignore all the ab values)
    d = det(A_mat);

    ab = pinv(A_mat)*x_vec;
    ab = ab';

    if d == 0
        x_int = Mx;
        vertFlag = 1;
    else
        x_int = (-1*ab(2))/ab(1);
        vertFlag = 0;
    end


    lineDetails = cat(2, ab, x_int);
    lineDetails  = cat(2, lineDetails, vertFlag);

    %plotLSM(xpoint, ypoint, iStart, iEnd, lineDetails);
end

%% plotLSM -  plot based on the output of Least Squared Minimisation
%                algorithm
%
```

```
% Usage:  plotLSM(xpoint, ypoint, iStart, iEnd, lineDetails);
%
% Arguments:
%            xpoint - set of x coordinates
%            ypoint - set of y coordinates
%            iStart - starting index
%            iEnd   - ending index
%            lineDetails - array of 4 values
%               (1) gradient of estimated line
%               (2) y-intercept of estimated line
%               (3) x-intercept of estimated line
%               (4) flag value 1 or 0 (1 if line is vertical; 0 if not)
%
%
% Author:
% Kausthub Krishnamurthy
% The University of Sydney
% kkri3182@uni.sydney.edu.au
% April 2015
function plotLSM(xpoint, ypoint, iStart, iEnd, lineDetails)
    hold on
    if lineDetails(4) == 1
        %if vertical plot points at very small increments (0.01) at x-int
        y = [ypoint(iStart):0.01:ypoint(iEnd)];
        plot(lineDetails(3), y);
    else
        %plot based on y = mx+b
        x = [xpoint(iStart):0.1:xpoint(iEnd)];
        plot(x, lineDetails(1)*x+lineDetails(2));
    end
end
```

1.c Line Segmentation Algorithm

```matlab
%% LINESEG -  Line Segmentation Algorithm Handler
% Purpose: Calls the Recursive Line Segmentation Algorithm and compiles
%          & formats the output
%          Also handles plotting of lines based on corner points
%
% Usage:  vertices = lineseg(xPoints, yPoints, linThres)
%
% Arguments:
%             xpoint - set of x coordinates
%             ypoint - set of y coordinates
%             linThres - linearity threshold value (sets the variance limit)
%
% Returns:
%             2D Array coordinates of each corner in the form:
%                    x1 y1
%                    x2 y2
%                    x3 y3
%
%
% Author:
% Kausthub Krishnamurthy
% The University of Sydney
% kkri3182@uni.sydney.edu.au
% April 2015
function vertices = lineseg(xPoints, yPoints, linThres)
    %set up indices
    iStart = 1;
    iEnd = length(xPoints);

    vertexInds = [1];%first corner will always be the first data point

    %call to recursive function
    newInds = lineseg_recurs(iStart, iEnd, xPoints, yPoints, linThres);

    %appends generated corner values to list
    vertexInds = [vertexInds, newInds];

    %sorts list of corner value (not strictly necessary as it should
        %naturally concatenate from left to right
    vertexInds = sort(vertexInds);

    %reformat from index values to x & values
    numVertices = length(vertexInds);
    vertices = zeros(numVertices, 2);
    for i = 1:numVertices
            vertices(i, 1) = xPoints(vertexInds(i));
            vertices(i, 2) = yPoints(vertexInds(i));
    end

    %call to plotter based on corner values
   %close all
   plotLSEG(vertices, numVertices);
end

%% LINESEG_RECURS -  Recursive Line Segmentation Algorithm
%
% Purpose: To recursively compute and return each corner point individually
% Usage:  tempToConcat = lineseg_recurs(iStart, iEnd, xPoints, yPoints, linThres)
%
% Arguments:
%             iStart - starting index
%             iEnd   - ending index
%             xpoint - set of x coordinates
%             ypoint - set of y coordinates
%             linThres - linearity threshold value (sets the variance limit)
%
% Returns:
```

```
%           1D array of index values of corner points from existing data
%           points
%
% Author:
% Kausthub Krishnamurthy
% The University of Sydney
% kkri3182@uni.sydney.edu.au
% April 2015
function tempToConcat = lineseg_recurs(iStart, iEnd, xPoints, yPoints, linThres)
    maxPDist = 0;
    maxPDInd = 0;
    %generate first line estimation
    tempLine = leastsqmin(xPoints, yPoints, iStart, iEnd);

    %find max distance from line estimation
    for n = iStart:iEnd
        temPDist = perpdist(xPoints(n), yPoints(n), tempLine(1), tempLine(2), tempLine(3), tempLine
            ↪ (4));
        if temPDist > maxPDist
            if(n ~= iStart)
                if(n ~= iEnd)
                    maxPDist = temPDist;
                    maxPDInd = n;
                end
            end
        end
    end

    if maxPDist <= linThres
        %base exit case
        %disp 'linear... hell yeah' %debug line
        tempToConcat = [iEnd];
    else
        %recursively repeat this function to split left and right segments
            %left segment is from the starting index to the First (from the
            %left) max distance point
            %right segment is from the First max distance point to the
            %end index point
        %disp 'not linear'  %debug line
        left_ttc = lineseg_recurs(iStart, maxPDInd, xPoints, yPoints, linThres);
        right_ttc = lineseg_recurs(maxPDInd, iEnd, xPoints, yPoints, linThres);
        %concatenate corner index values returned from above (works similar
        %to a recursive merge
        tempToConcat = [left_ttc, right_ttc];
    end
end


%% PLOTLSEG -  Plot Segmented Lines
%
% Purpose: To plot the detected lines from the data after detecting valid
%           corner points
% Usage:  plotLSEG(cnr, n)
%
% Arguments:
%            vrts   -   2D Array of corners
%                       x1 y1
%                       x2 y2
%                       x3 y3
%
% Author:
% Kausthub Krishnamurthy
% The University of Sydney
% kkri3182@uni.sydney.edu.au
% April 2015
function plotLSEG(vrts, n)
    for i = 1: n-1

        xEnds = [vrts(i,1) vrts(i+1,1)];
        yEnds = [vrts(i,2) vrts(i+1,2)];
        line(xEnds, yEnds);
        end
    end
```

1.c.i Test Fitting

The following images are saved plots of sample data plots of increasing complexity where the original data is plotted with a red 'x' and the blue line through it is the approximated line segments. The relevant test scripts can be found in Appendix A [1]. It's interesting to note that the sideways v test faces some inaccuracy in the approximation due the restriction on not allowing the index end points to be considered "corners". This is one of the shortcomings of using such a simple method of resolving the "broken data set" situation. A method that minimises that error as well would be much more difficult to program and much more intricate to specific case dealing.

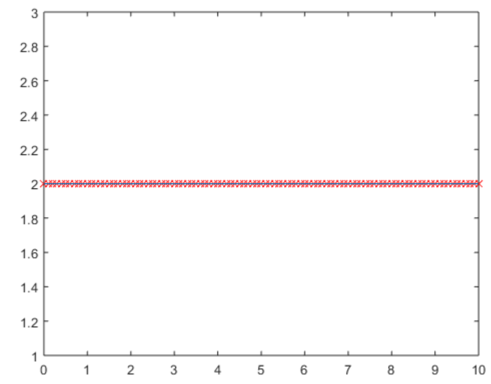


Figure 1: Horizontal Line

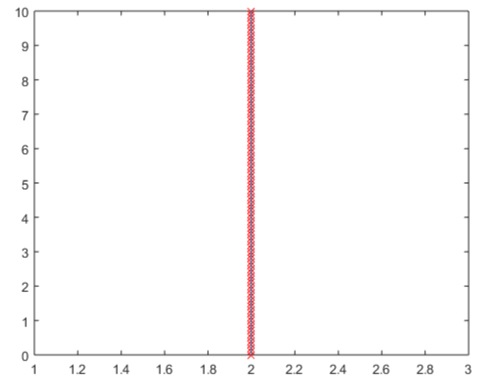


Figure 2: Vertical Line

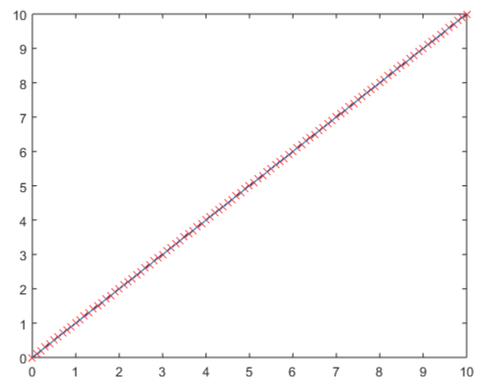


Figure 3: Slope (Through Origin)

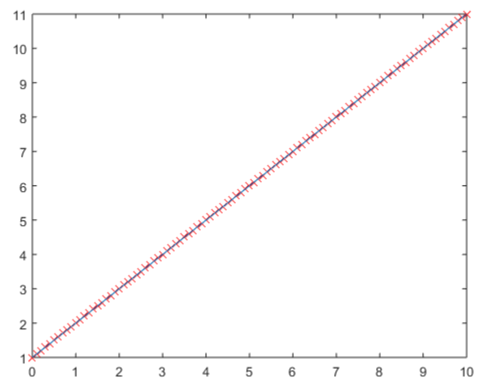


Figure 4: Offset Slope

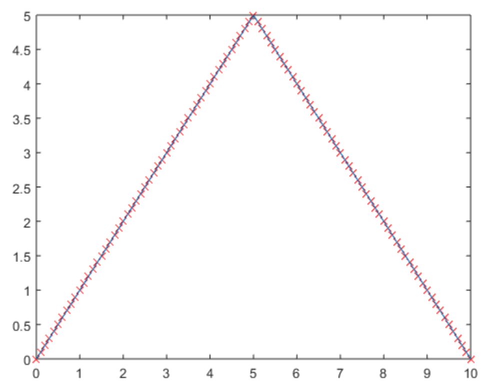


Figure 5: Inverted V

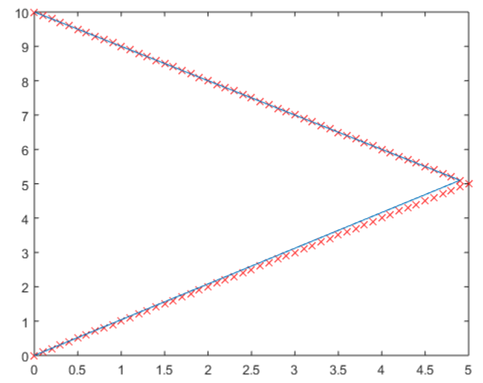


Figure 6: Sideways V

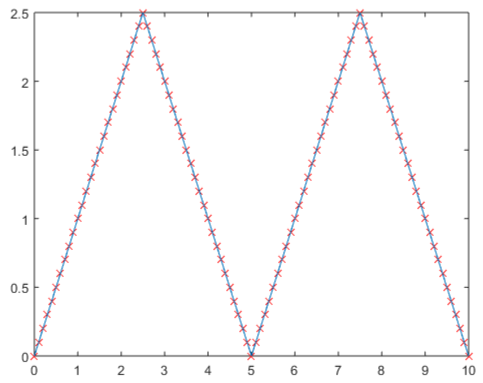


Figure 7: Perpendicular ZigZag)

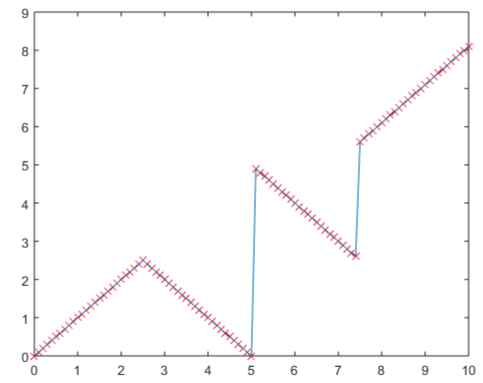


Figure 8: Broken Zig Zag

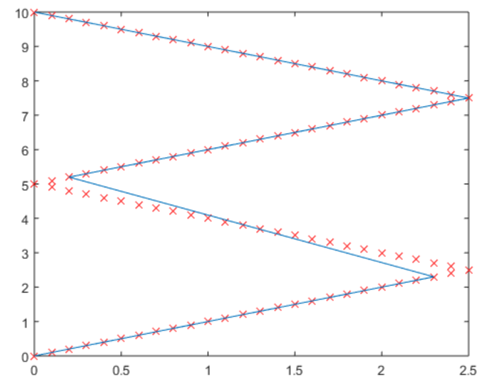


Figure 9: Perpendicular Zig Zag (Sideways)

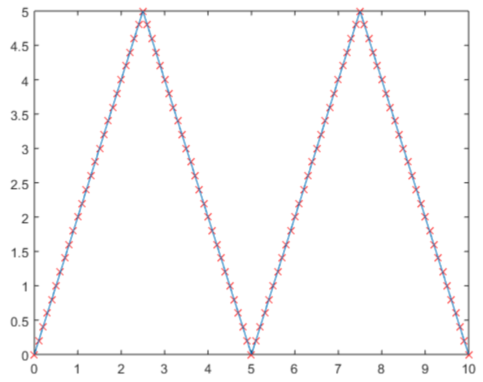


Figure 10: Non-Perpendicular ZigZag

1.c.ii Test 06 Step by Step

The following images take a step by step plotted walk through on each estimation made on a simple case such as the Zig Zag pattern shown in test 06. The important thing to note is the recursive method utilised in this code which splits this into a logical problem solving technique (similar to mergesort or quicksort algorithms) by selecting pivot-points

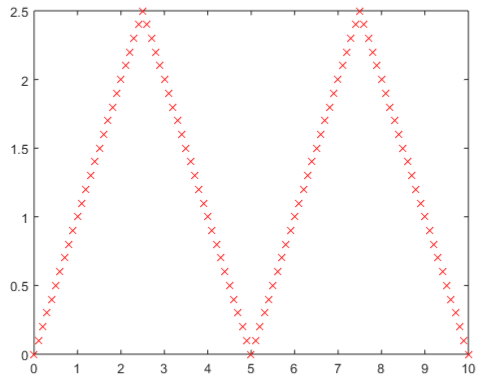


Figure 11: Test 06 Stepped Through

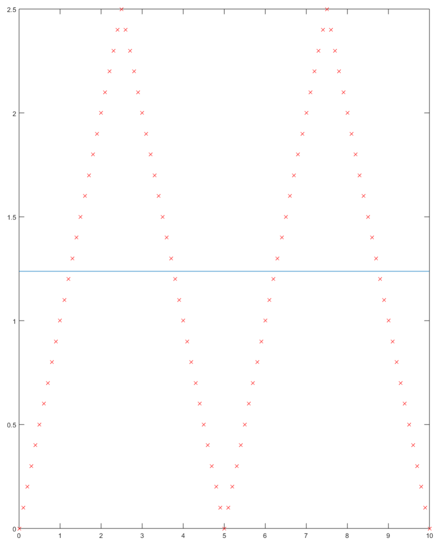


Figure 12: Test 06 Stepped Through

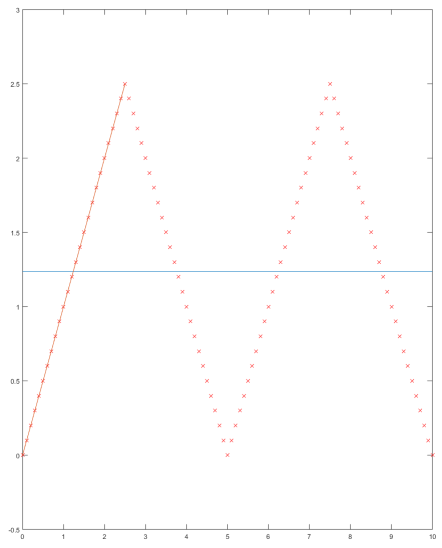


Figure 13: Test 06 Stepped Through

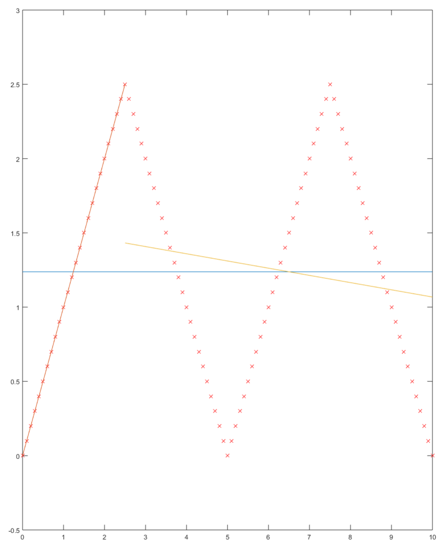


Figure 14: Test 06 Stepped Through

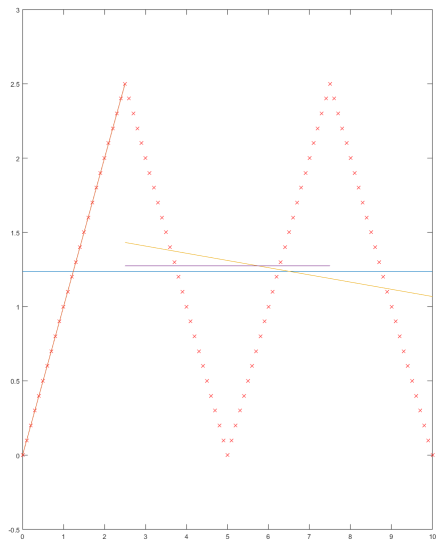


Figure 15: Test 06 Stepped Through

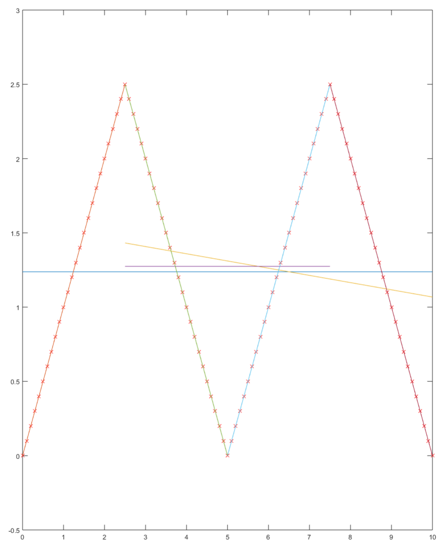


Figure 16: Test 06 Stepped Through

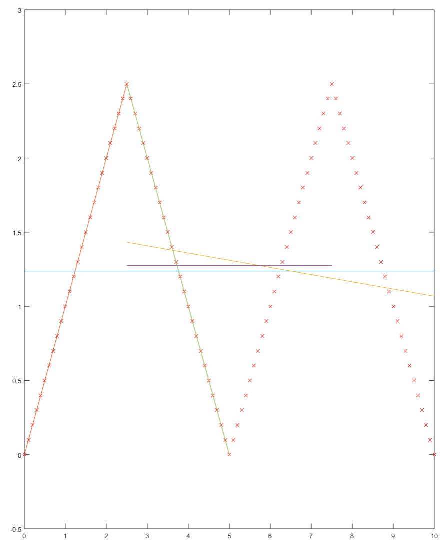


Figure 17: T

est 06 Stepped Through

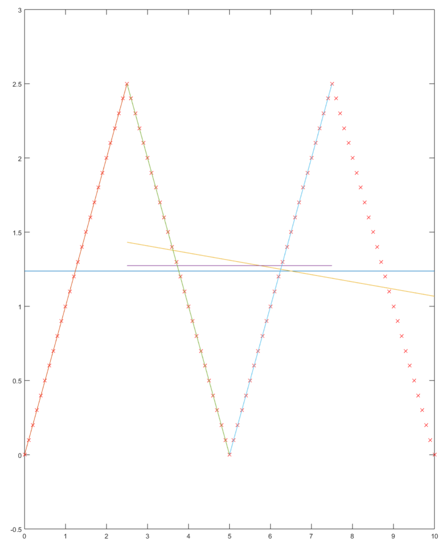


Figure 18: Test 06 Stepped Through

1.d Laser Show ACFR Line Fitting

```
%
% Author: Stefan Williams (stefanw@acfr.usyd.edu.au)
%
%

clear
close all

% load laser files
laser_scans=load('..\datasets\captureScanshornet.txt');

t0 = laser_scans(1,1);

figure
for i = 1
     tlaser = laser_scans(i,1) - t0;
     xpoint = zeros(1);
     ypoint = zeros(1);
     for j = 2:size(laser_scans,2)
         range = laser_scans(i,j) / 1000;
         bearing = ((j-1)/2 - 90)*pi/180;
         if (range < 75)
             xpoint = [xpoint range*cos(bearing)];
             ypoint = [ypoint range*sin(bearing)];
         end
     end
     plot(xpoint(:), ypoint(:), '.');
     axis equal;
     axis([0 10 -5 5]);
     xlabel('X (meter)')
     ylabel('Y (meter)')
     title(sprintf('ACFR indoor SICK data: scan %d',i))
     drawnow
end

corners = findcorners(xpoint, ypoint, 0.1)
hold on
plot(xpoint(:), ypoint(:), 'rx');
```

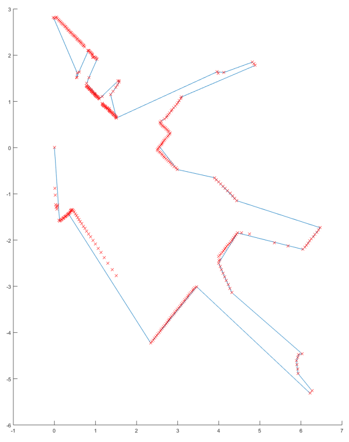


Figure 19: LaserShowACFR approximate

1.e Corner Detection

```matlab
function corners = findcorners(xvals, yvals, linThresh)
    vertices = lineseg(xvals, yvals, linThresh);
    nVert = length(vertices);
    nVec = nVert-1;
    vectors = zeros(nVec, 2);

    vertices

    for i = 1:nVec
            vectors(i, 1) = vertices(i+1, 1)-vertices(i, 1);
            vectors(i, 2) = vertices(i+1, 2)-vertices(i, 2);
    end

    vectors

    cornerCount = 0;
    for i = 1:(nVec-1)
        v1 = [vectors(i, 1), vectors(i, 2)];
        v2 = [vectors(i+1, 1), vectors(i+1, 2)];
        dp = dot(v1, v2);
        if(dp < 0.00018 && dp > -0.00018)
            cornerCount = cornerCount + 1;
            corners(cornerCount, 1) = vertices(i+1, 1);
            corners(cornerCount, 2) = vertices(i+1, 2);
        end
    end

    if cornerCount == 0
        disp 'no corners'
        corners = 'N/A';
    end

    corners
    cornerCount
end
```

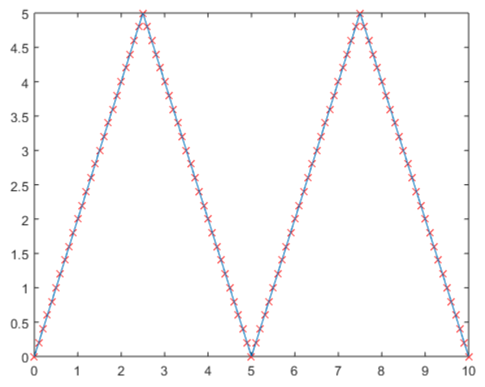


Figure 20: Non-Perpendicular Zig Zag Corner Test

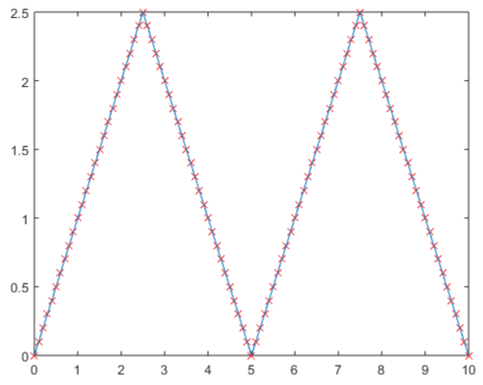


Figure 21: Perpendicular Zig Zag Corner Test

```
Q1 Test Result Corner Value outputs from LineSeg
test00: horizontal line test (no error)
test01: vertical line test (no error)
test02: simple slope line test (no error)
test03: slope line test (no error)
test04: vertical inverse V LSM test (no error)
test05: vertical inverse V LSM test (no error)
{Maximum recursion limit of 500 reached. Use set(0,'RecursionLimit',N) to change the limit. Be aware
    ↪  that exceeding your available stack space can crash
MATLAB and/or your computer.

Error in <a href="matlab:matlab.internal.language.introspective.errorDocCallback('mean')" style="
    ↪ font-weight:bold">mean</a>


}
test06
test06: horizontal zig zag test (no error)
test07: broken horizontal zig zag test (no error)
{Maximum recursion limit of 500 reached. Use set(0,'RecursionLimit',N) to change the limit. Be aware
    ↪  that exceeding your available stack space can crash
MATLAB and/or your computer.

Error in <a href="matlab:matlab.internal.language.introspective.errorDocCallback('mean')" style="
    ↪ font-weight:bold">mean</a>


}
test08
test08: vertical zig zag test (no error)
{Maximum recursion limit of 500 reached. Use set(0,'RecursionLimit',N) to change the limit. Be aware
    ↪  that exceeding your available stack space can crash
MATLAB and/or your computer.

Error in <a href="matlab:matlab.internal.language.introspective.errorDocCallback('mean')" style="
    ↪ font-weight:bold">mean</a>


}
test09
test09: non-perpendicular zigzag tst (no error)
test10: corner finder test on non-perpendicular zigzag test (no error)

vertices =

         0         0
    2.5000    5.0000
    5.0000         0
    7.5000    5.0000
```

```
   10.0000         0


vectors =

    2.5000    5.0000
    2.5000   -5.0000
    2.5000    5.0000
    2.5000   -5.0000

no corners

corners =

N/A

test11: corner finder test on perpendicular zigzag test (no error)

vertices =

         0         0
    2.5000    2.5000
    5.0000         0
    7.5000    2.5000
   10.0000         0


vectors =

    2.5000    2.5000
    2.5000   -2.5000
    2.5000    2.5000
    2.5000   -2.5000


corners =

    2.5000    2.5000
    5.0000         0
    7.5000    2.5000
```

As you can see the corners listed match perfectly with those graphed on Fig 21 (Test 11).

```

cornerCount =

     8


corners =

    0.2509   -1.4992
    0.2639   -1.4969
    0.2866   -1.4744
    0.2993   -1.4709
    0.3210   -1.4478
    0.3338   -1.4460
    2.5569    0.0223
    1.5748    1.4431
```

The above corner values relate to the plot of the line estimates of the Laser Show ACFR scan in figure 19.

Test Code Listing

See Appendix A [9.1]

2 Question 2

2.a Validity Check

2.a.i R_1

2.a.ii R_2

2.a.iii R_3

2.a.iv R_4

2.b Roll/Pitch/Yaw Angles

2.c Angle Estimation

Code Listing

See Appendix A [9.2]

3 Question 3

The Iterated Closed Loop Algorithm

By modifying the given code pieces, we will implement an Iterated Closed Loop algorithm to estimate the position of a vehicle as it moves through its surroundings.
All code pieces, original and modified, can be found in Appendix A [3].

3..i Part A - Implementing the ICP

By modifying the given showICP.m file, we will exam the resultant ICP features generated for a single data set. The set in question is frame 500, and we will use frame 520 as our initial 'guess'.
Firstly, using the default variables of a grid size of 0.005, and a maximum iterative loop of 40, we can generate the following graph:

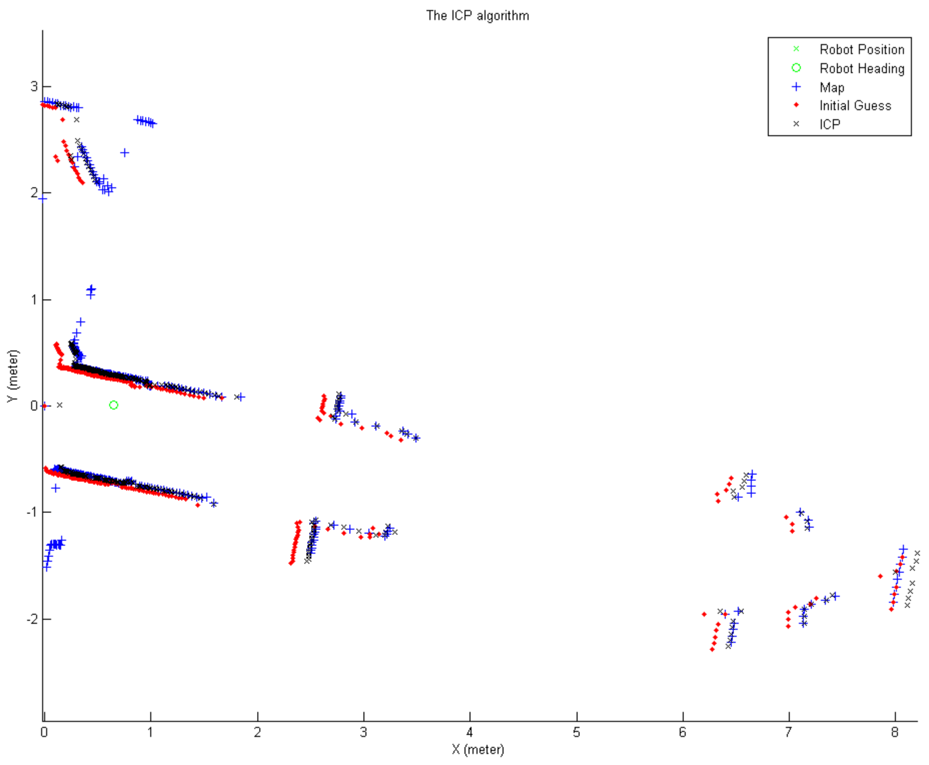


Figure 22: ICP estimate for maximum iterations of 40 and grid size of 0.005

We will now examine the effect of modifying some of the variables of the ICPv4.m algorithm.
Firstly we will look at changing the grid size. For a smaller grid size of 0.001 we obtain the following graph:

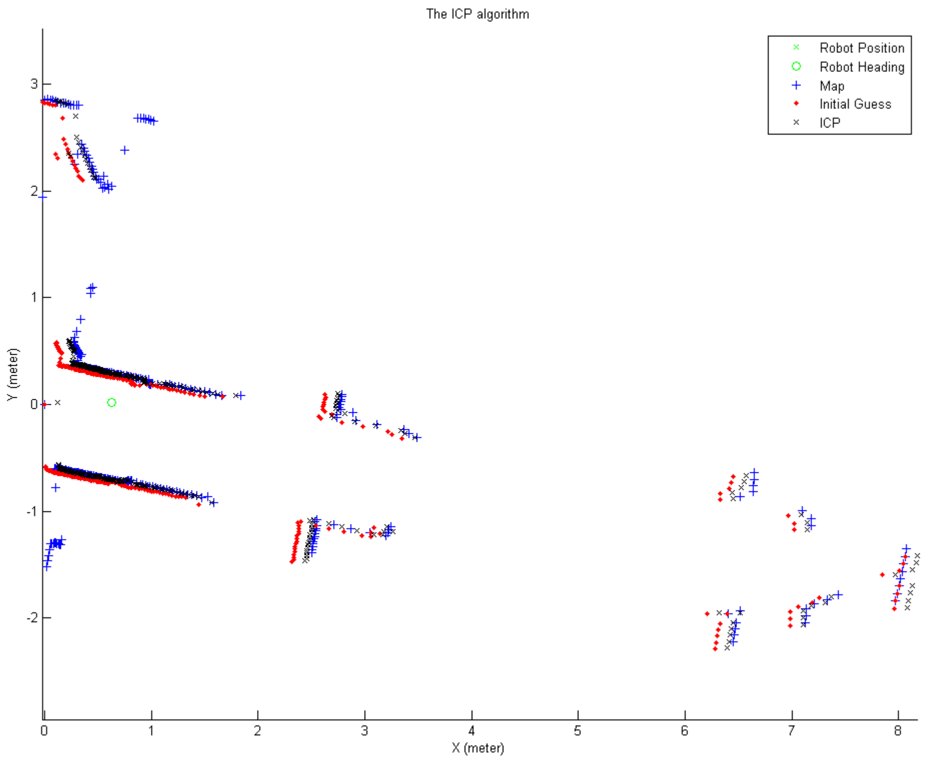


Figure 23: ICP estimate for maximum iterations of 40 and grid size of 0.001

Next, for a grid size of 0.1:

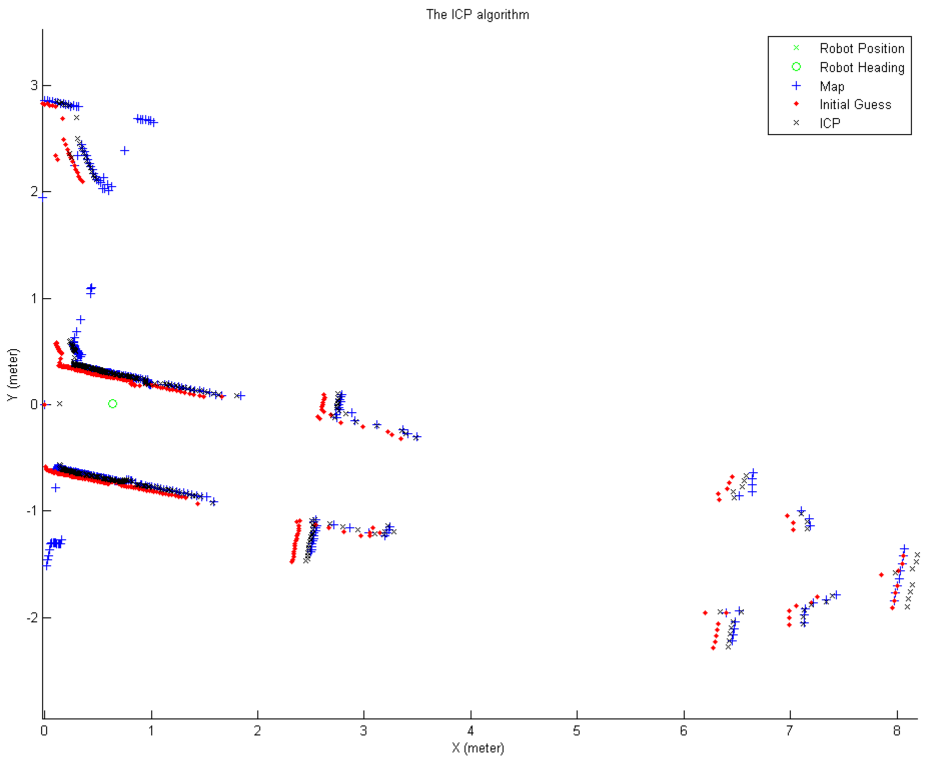


Figure 24: ICP estimate for maximum iterations of 40 and grid size of 0.005

As can be seen, changing the grid size has little effect on the overall ICP .
It does, however, have an affect on the number of collision points detected. For a grid size of 0.005, 188 points collide. For 0.001, 32 points collide, and for 0.01, 252 points collide. This is expected - an increase in in the grid size means a large sample section, with a greater likelihood of multiple points landing in a grid.

Checking for matching pairs reveals an interesting point - for all tested values for grid size, the number of matching points is the same - 362. Also of worthy note, despite a maximum number of iterations of 40, no more than 9 iterations are used. Changing the maximum number of iterations to 10 resulted in no changes to any of the previous tests. As such, the maximum iterative size does not need to be nearly so large.

Looking at the generated deltaPose$_bar, we can get an idea of the estimated heading of the vehicle, and by looking at deltaPose_bar_cov$
$The pose was as follows:$

$deltaPose_{bar} = [0.1360, 0.0116, 0.0004]$ where the pattern is $[x, y, \theta]$

This Is very close to the zero position, which is to be expected seeing as this ICP algorithm has only taken a single frame. Relative movement should be little at this point.
The Pose covariance for this is as follows:

$$deltaPose_bar_cov = 1.0^-5 * \begin{pmatrix} 0.2924 & 0.0130 & -0.0099 \\ 0.0130 & 0.4039 & -0.0863 \\ -0.0099 & -0.0863 & 0.0659 \end{pmatrix}$$

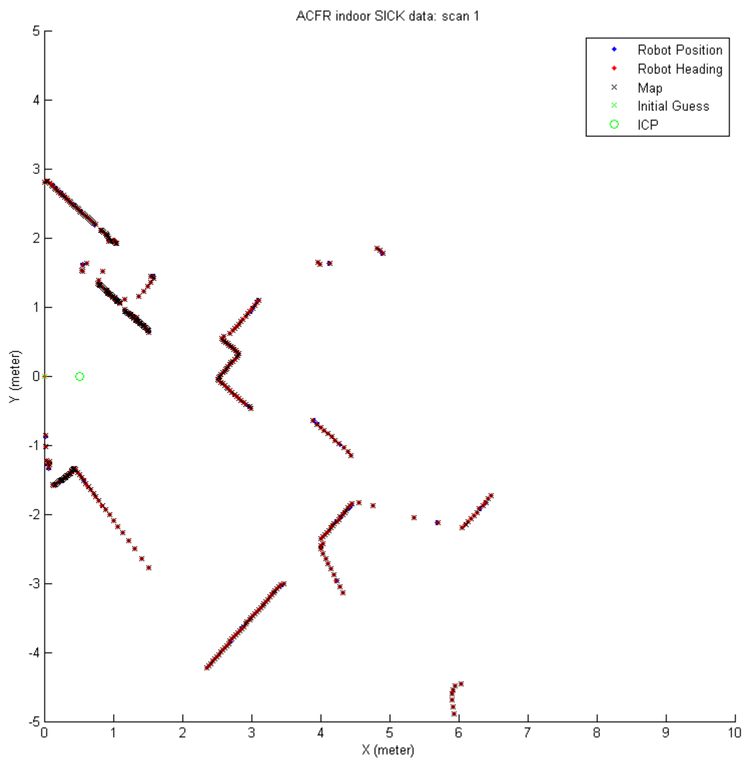
As can be seen, the covariance matrix is extremely close to zero. This is indicative of the factors involved being completely independent, though it does not confirm this. Again, seeing as this is run from a single frame, it is not an indicator for the overall relationship - we have used far too few data points to be able to rule anything out.

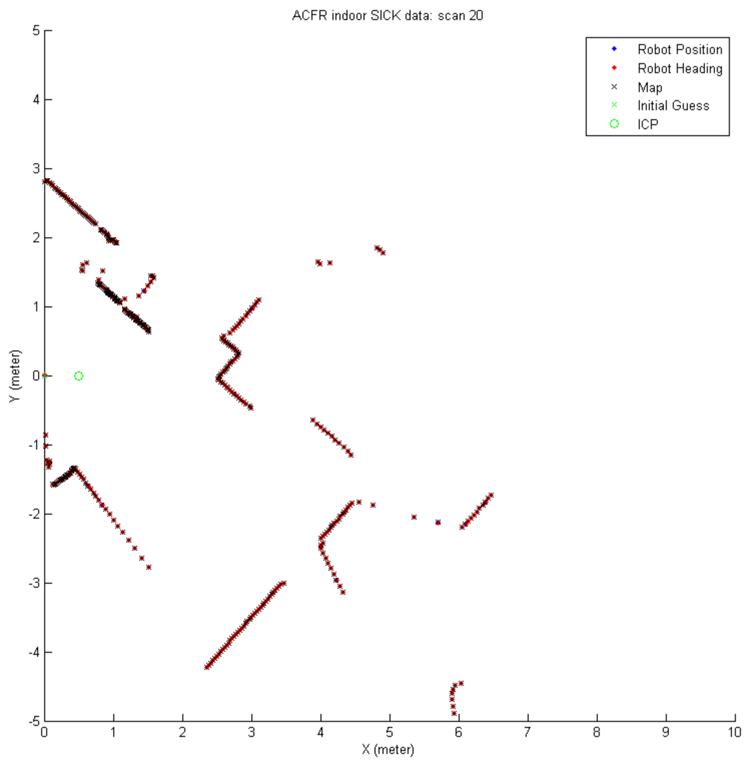
Problems with the Algorithm

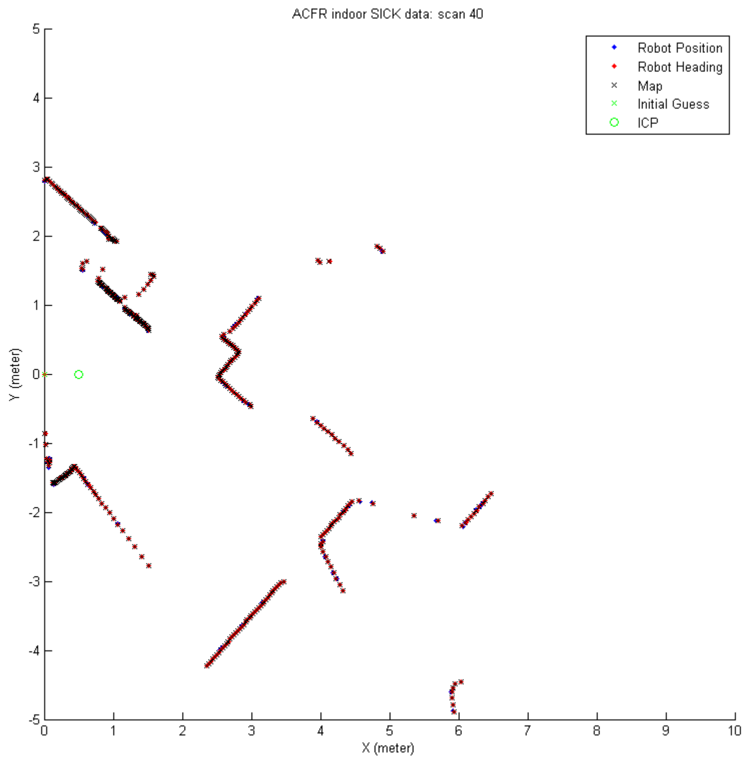
ICPv4.m takes little into account to do with angles/rotations, mostly using x and y values. This would mean that any rotational movement in the map would be taken poorly into account, as will be evident further on.

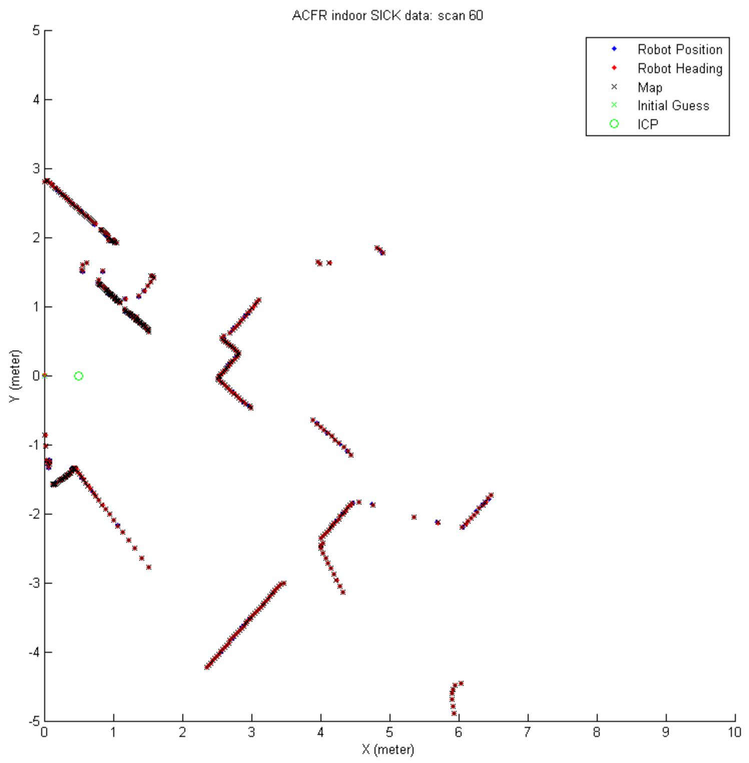
3..ii Part B

Incorporating the ICPv4.m algorithm into the laserShowACFR.m program enables us to build a real time picture of the movement of the vehicle and its perceived surroundings, relative to the actual mapped data. Observe:

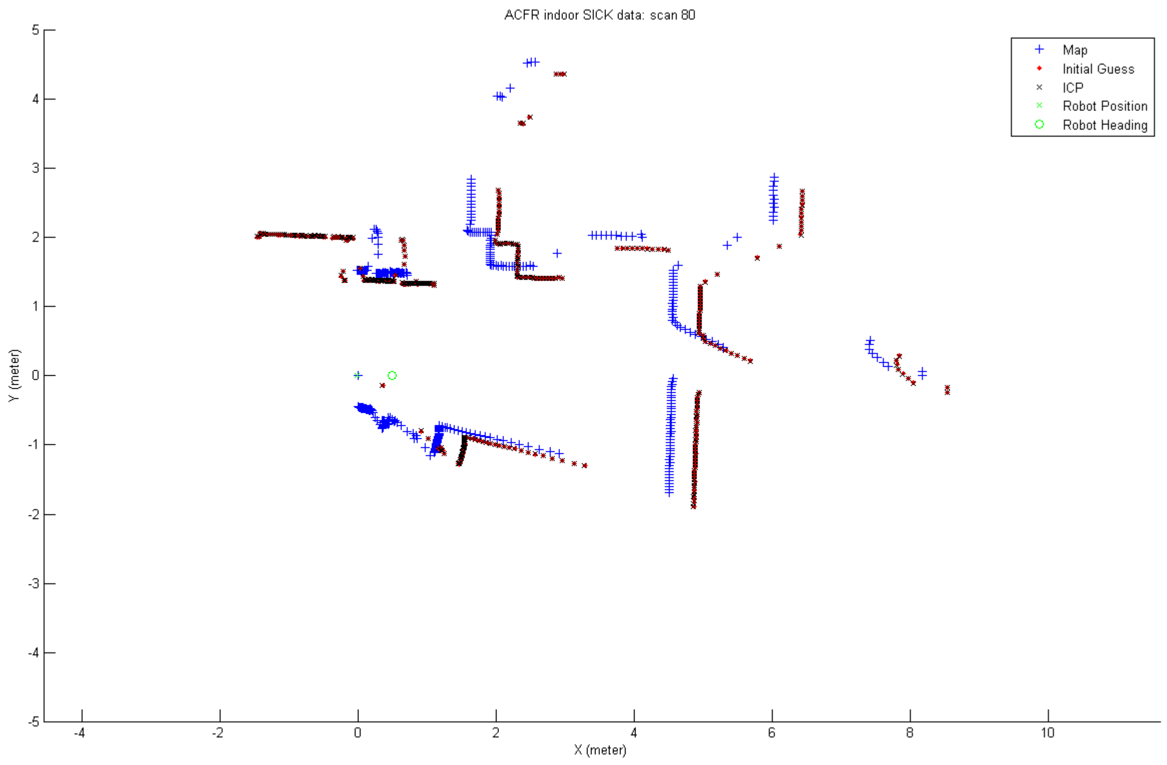



ACFR indoor SICK data: scan 20
Robot Position
Robot Heading
Map
Initial Guess
ICP
Y (meter)
X (meter)

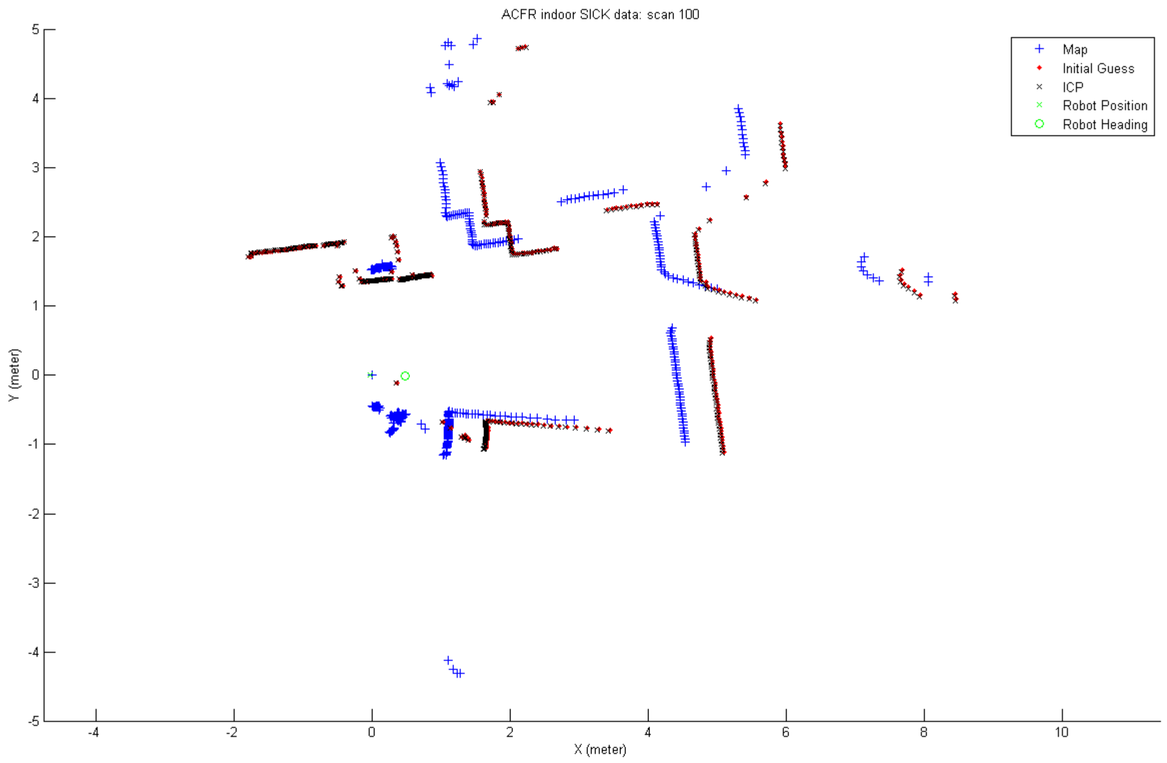

ACFR indoor SICK data: scan 40
Robot Position
Robot Heading
Map
Initial Guess
ICP
Y (meter)
X (meter)

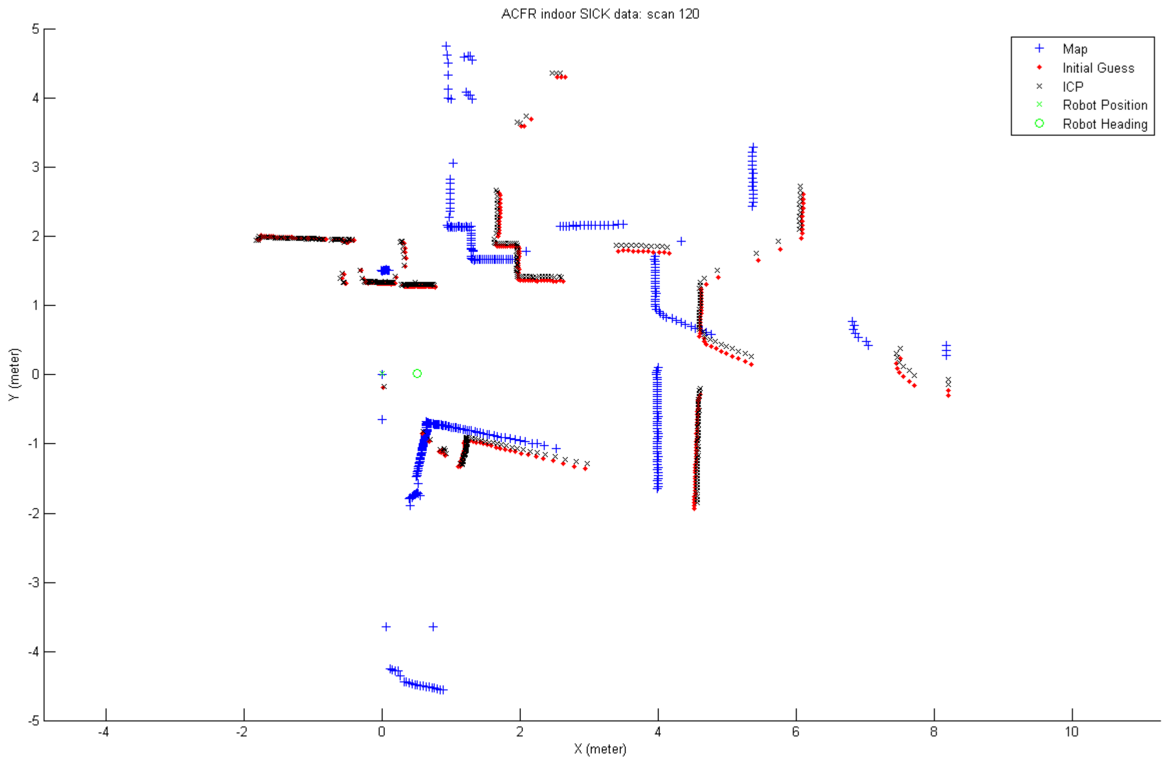


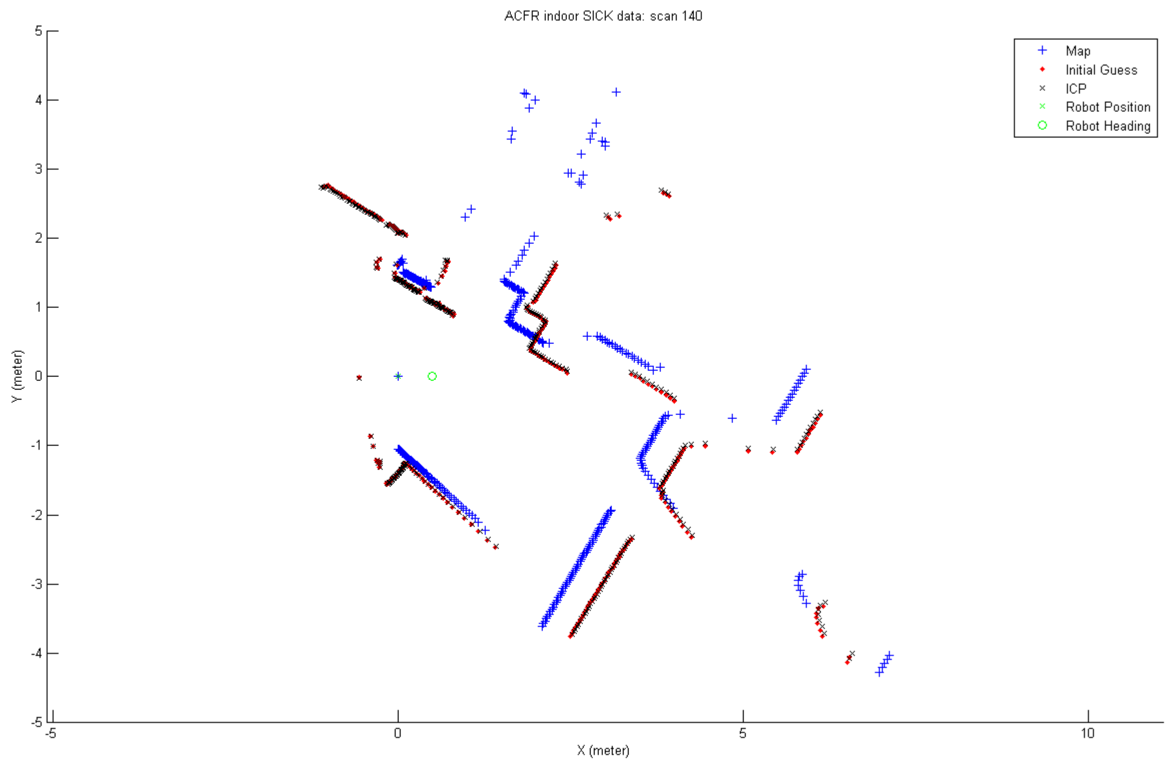
ACFR indoor SICK data: scan 60
Robot Position
Robot Heading
Map
Initial Guess
ICP
Y (meter)
X (meter)

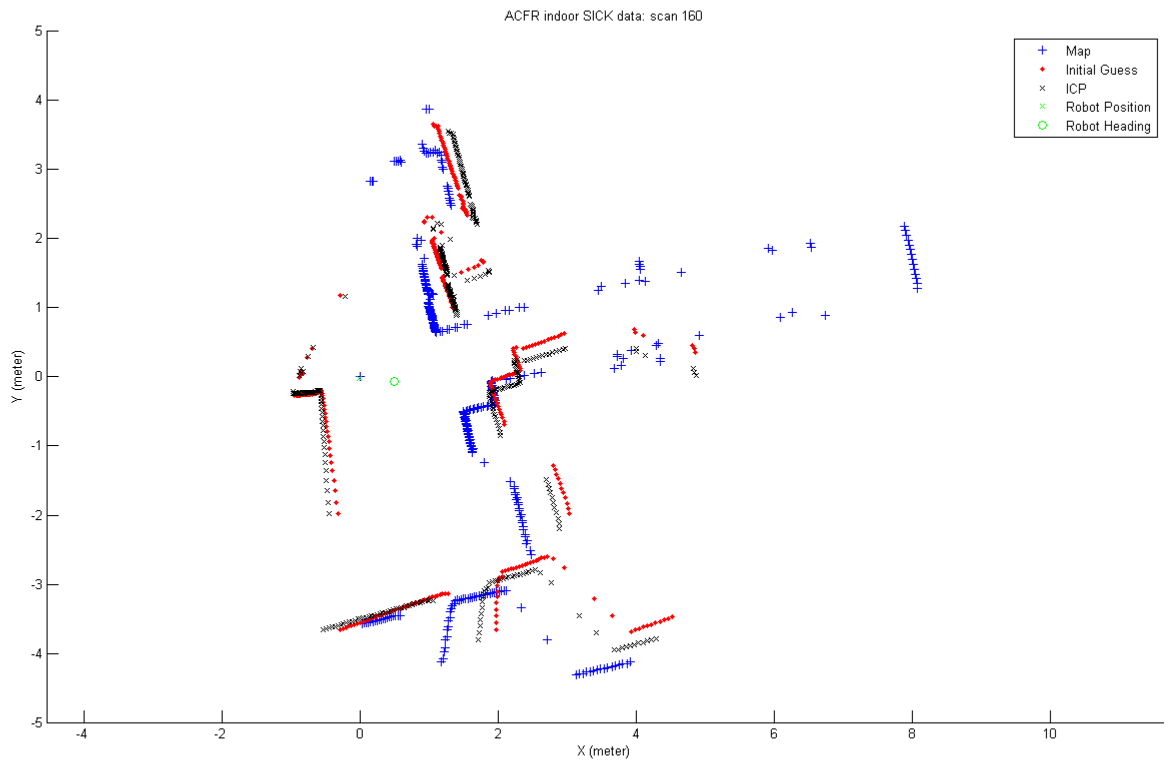


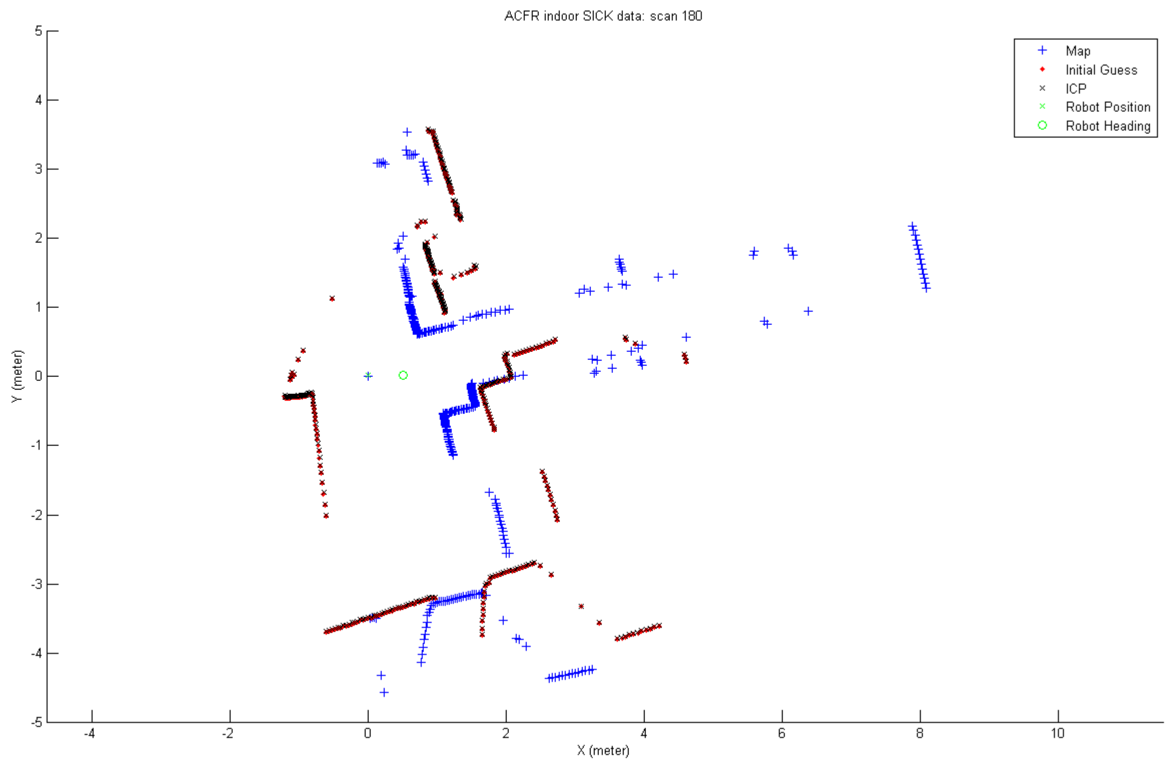
ACFR indoor SICK data: scan 80
Map
Initial Guess
ICP
Robot Position
Robot Heading
Y (meter)
X (meter)


ACFR indoor SICK data: scan 100
Map
Initial Guess
ICP
Robot Position
Robot Heading
Y (meter)
X (meter)

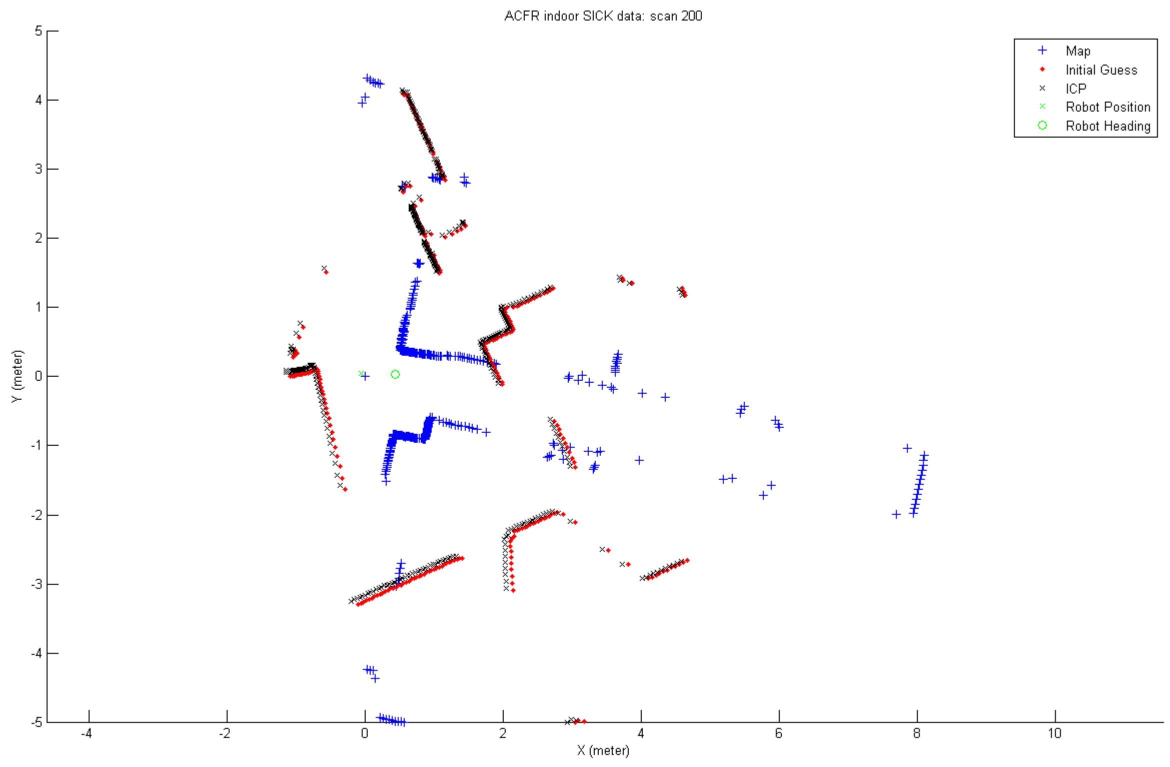

ACFR indoor SICK data: scan 120
Map
Initial Guess
ICP
Robot Position
Robot Heading
Y (meter)
X (meter)


ACFR indoor SICK data: scan 140
Map
Initial Guess
ICP
Robot Position
Robot Heading
X (meter)
Y (meter)

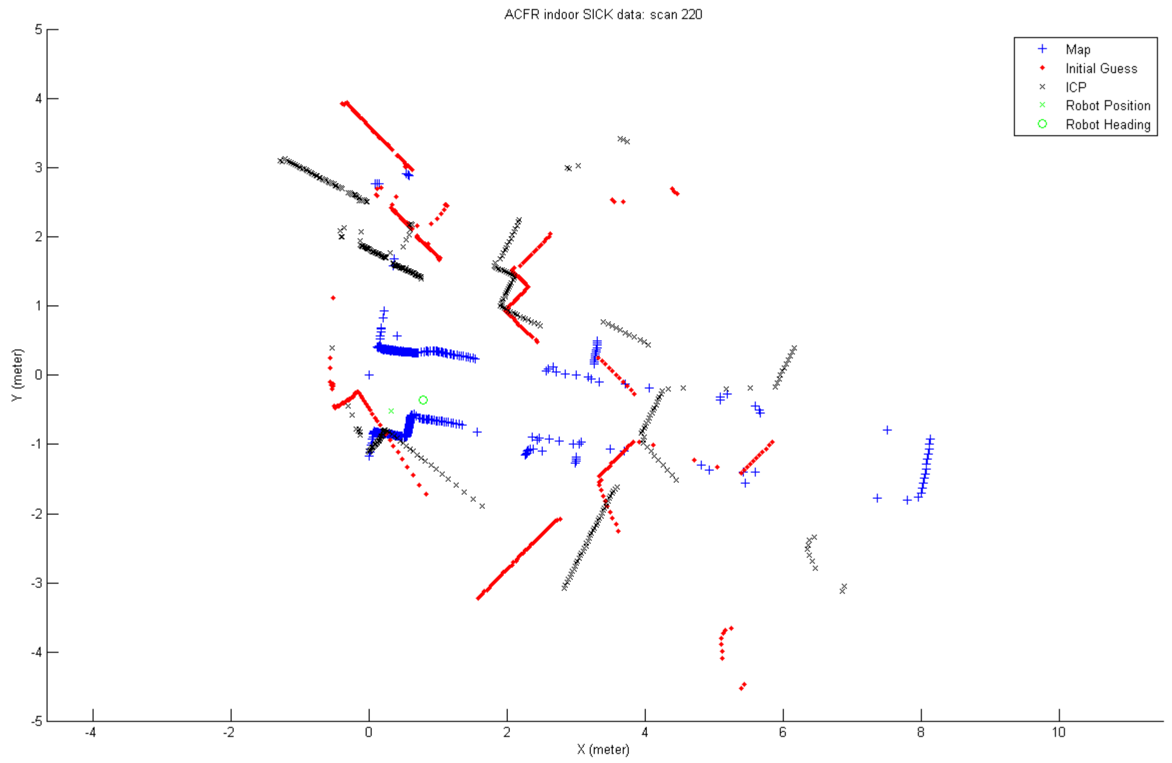

ACFR indoor SICK data: scan 160
Map
Initial Guess
ICP
Robot Position
Robot Heading
X (meter)
Y (meter)

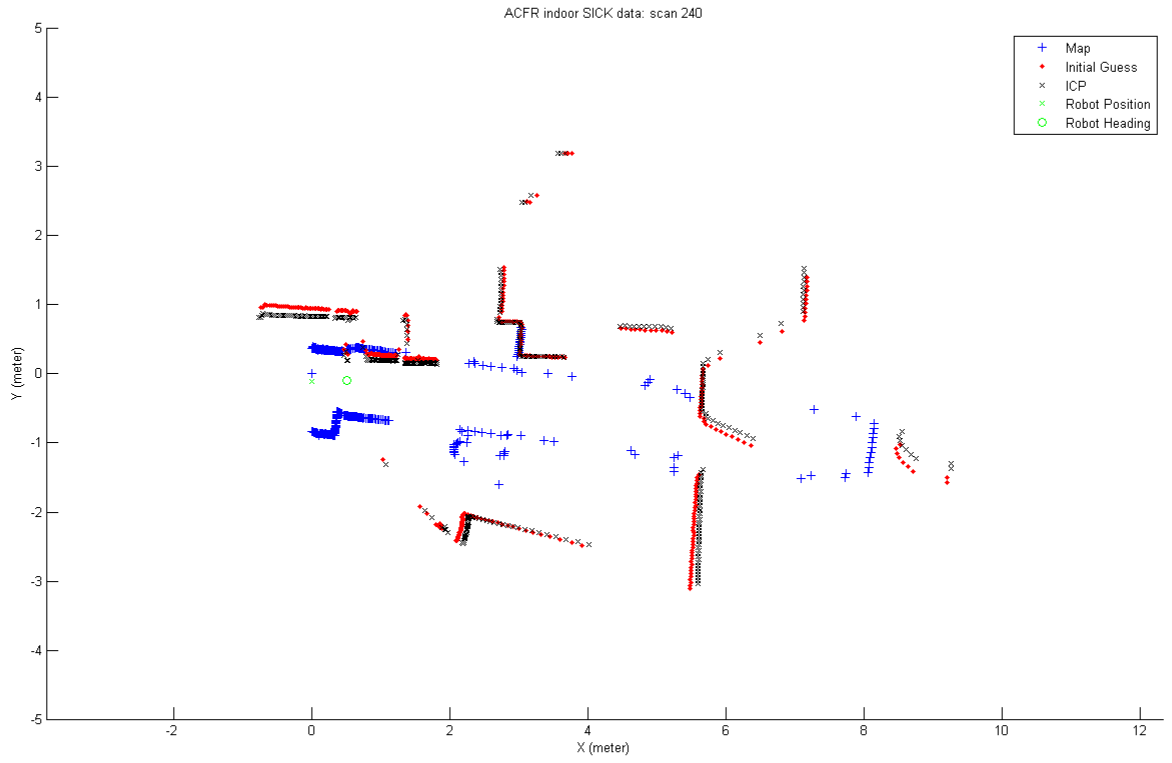


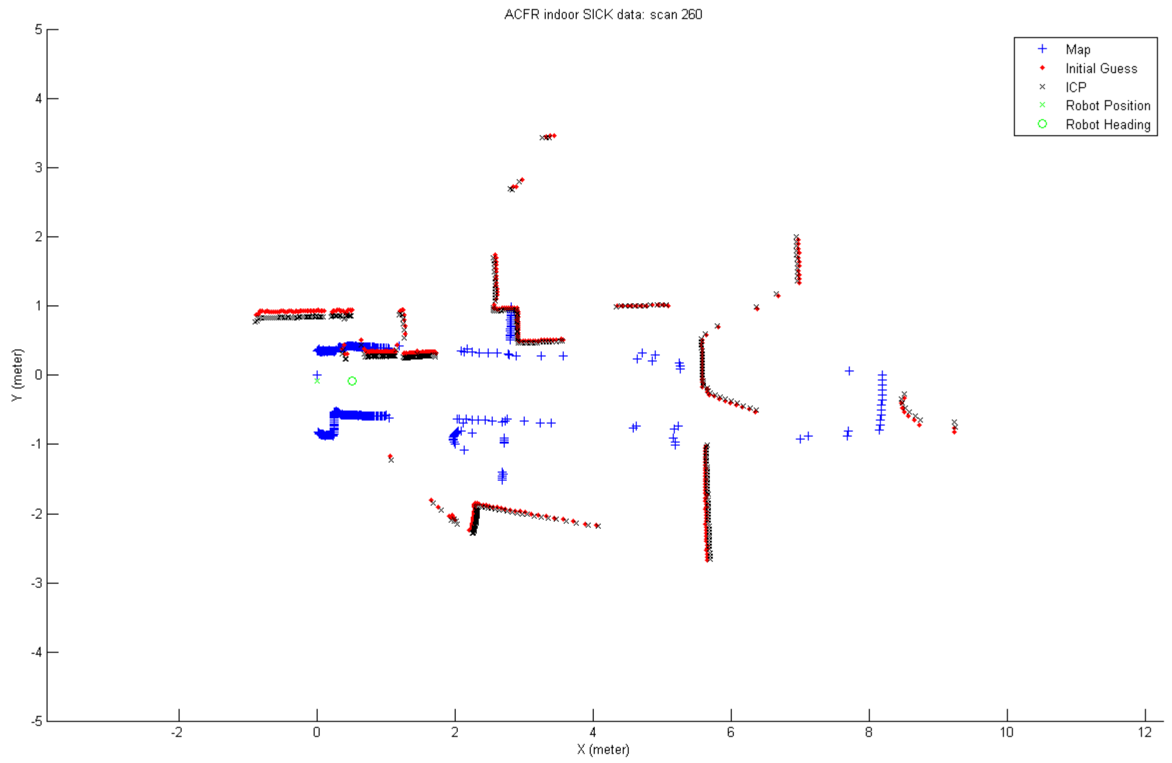
ACFR indoor SICK data: scan 180
Map
Initial Guess
ICP
Robot Position
Robot Heading
Y (meter)
X (meter)



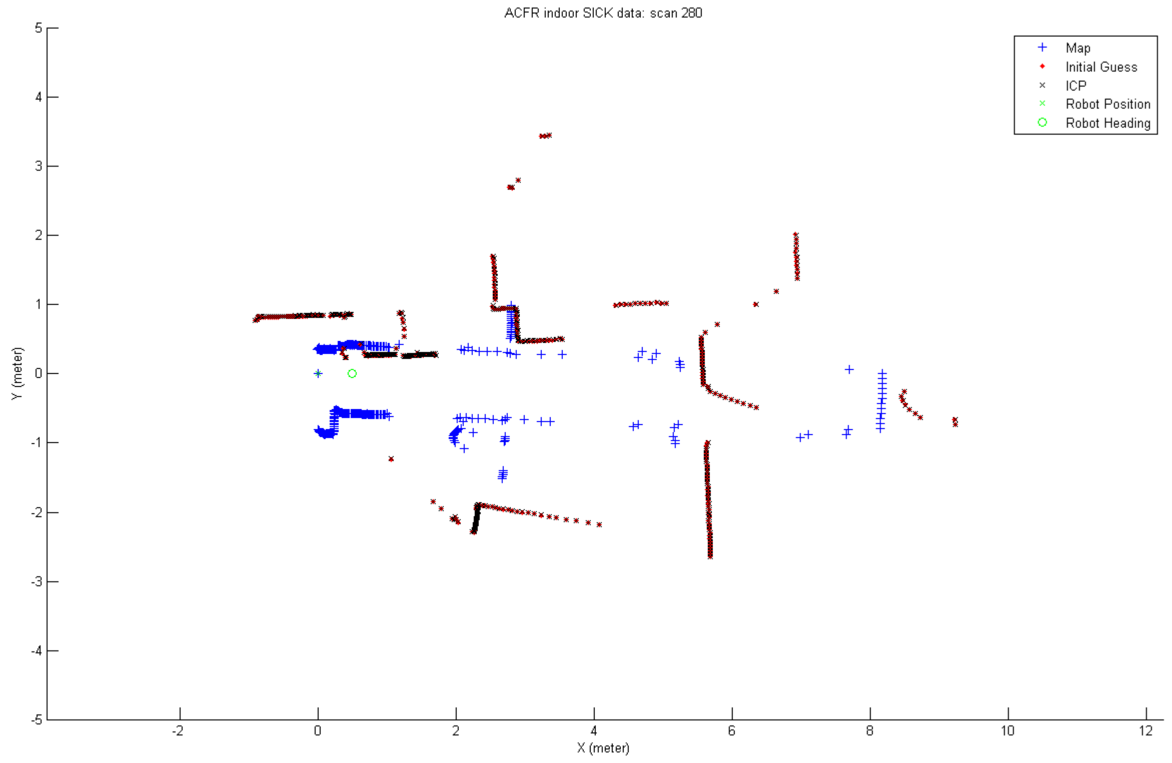
ACFR indoor SICK data: scan 200
Map
Initial Guess
ICP
Robot Position
Robot Heading
Y (meter)
X (meter)


ACFR indoor SICK data: scan 220
Map
Initial Guess
ICP
Robot Position
Robot Heading
Y (meter)
X (meter)
5
4
3
2
1
0
-1
-2
-3
-4
-5
-4
-2
0
2
4
6
8
10

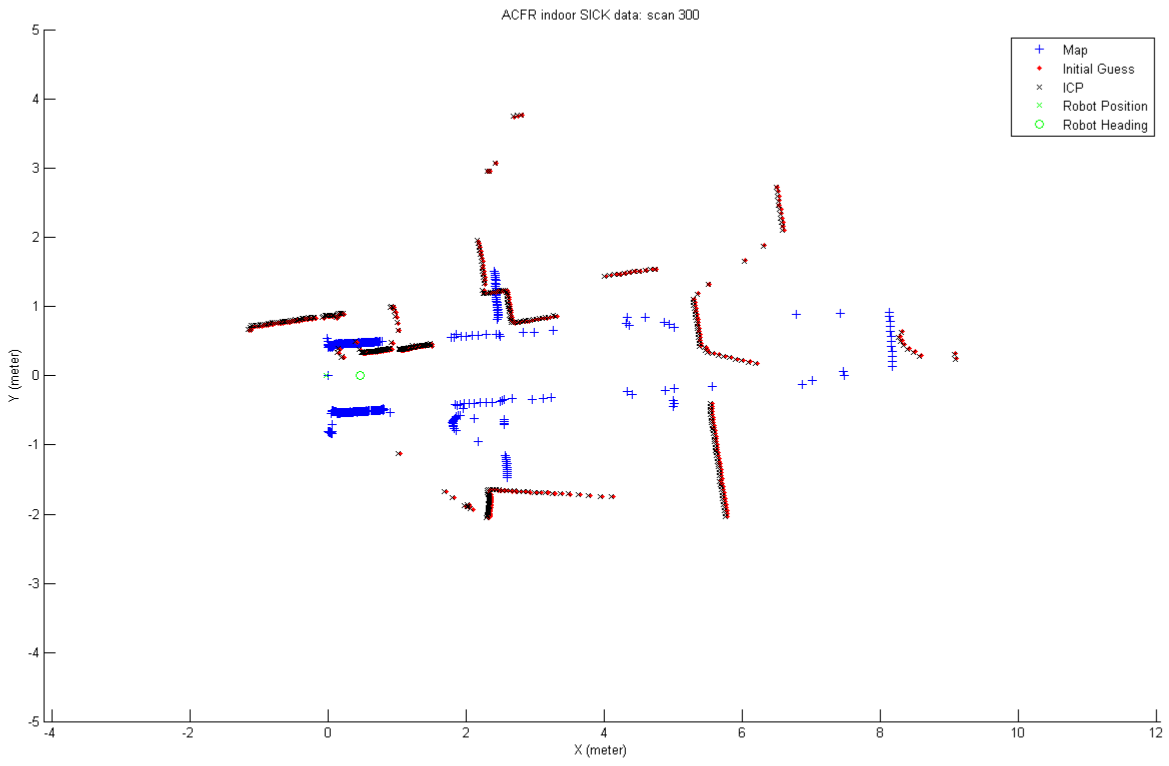

ACFR indoor SICK data: scan 240
Map
Initial Guess
ICP
Robot Position
Robot Heading
Y (meter)
X (meter)
5
4
3
2
1
0
-1
-2
-3
-4
-5
-2
0
2
4
6
8
10
12



ACFR indoor SICK data: scan 260
Map
Initial Guess
ICP
Robot Position
Robot Heading
X (meter)
Y (meter)



ACFR indoor SICK data: scan 280
Map
Initial Guess
ICP
Robot Position
Robot Heading
X (meter)
Y (meter)



As can be seen, the ICP initially tracks the map data rather well, and the vehicle remains roughly alligned with the direction of motion. Around scan 200 however, everything begins to break down. At this point, the ICP has started lagging behind, and when the vehicle suddenly rotates it is unable to keep up. The closest points now no longer correspond to the original positions, and the ICP map becomes distorted, aligining with a new set of map data points. By scan 300 the alignment is well an truly distorted, with the ICP data almost backwards on the map data. The scans remain relatively unchanged for the next thousand frames, which have not been shown here. The vehicle, suprisingly, appears to be orientated correctly. This is probably a fluke.

Considering that at some points in the scans the ICP vehicle position placed it in or beyond the walls, this would not be a particularly effect method of guidance.

Note, however, that though there was always some offset between the ICP map and the real map, this offset was almost constant. The greatest offset would occur during rotations, which the ICP was able to follow initially. However, these first few rotations where slow, enabling the ICP algorithm to keep up. Around frame 200, the first rapid rotation took place, and the ICP finally failed to keep up, recognising new points as the closest and pairing with them.
As mentioned above, this is because the ICP algorithm fails to take this rotational element into account. However, the given algorithm would still be acceptable under the correct circumstances. With either far more data points taken per seconds, enabling the ICP to 'linearise' the rotations, would enable it to handle sudden rapid rotations. Alternatively, the vehicle could be drastically slowed down, resulting in more data points through the rotation, again enabling some degree of linearity.

Code Listing

See Appendix A [3] for all code used.

4 Question 4

We have used given 10 images in camera calibration tool provided. After the calibration it provides us details on following two major parameters:

1. Intrinsic parameters(camera model)
2. Extrinsic parameters

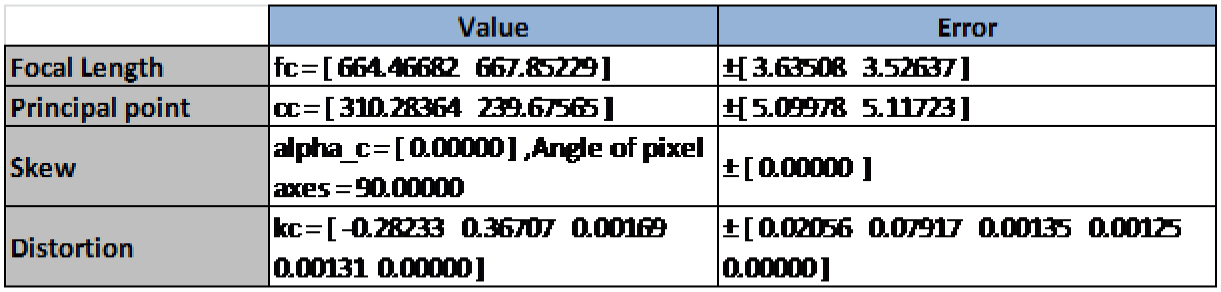
Under Intrinsic parameters it provided information about:

1. Focal length: The focal length in pixels - f_c
2. Principal point: The principle point coordinates cc
3. Skew coefficient: The skew coefficient defining the angle between the x and y pixel axes - α_c
4. Distortions: The image distortion coefficients (radial and tangential distortions) - k_c

(Reference: http://www.vision.caltech.edu/bouguetj/calib$_doc/htmls/parameters.html$)

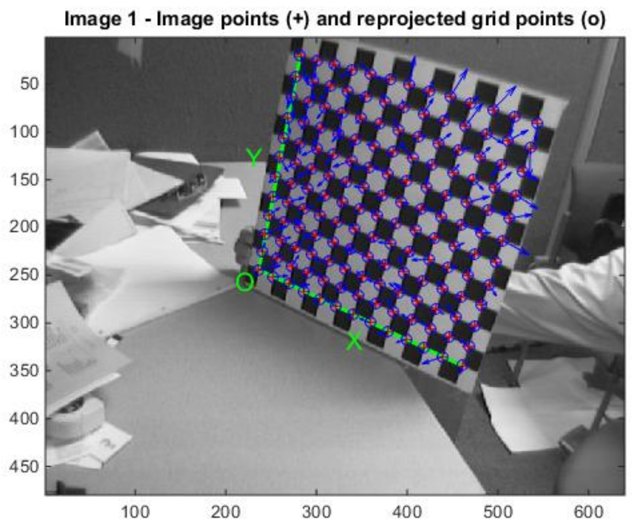
4.a Initial results after corner extraction

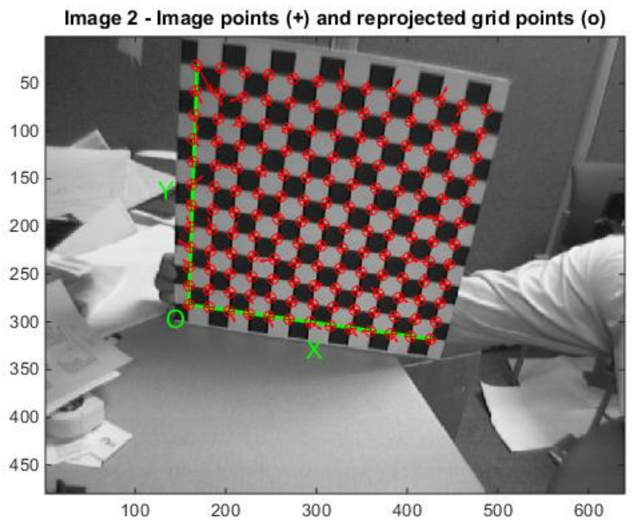
Following are the calibration results after initial calibration with a Pixel error of $[0.81041, 0.57619]$.



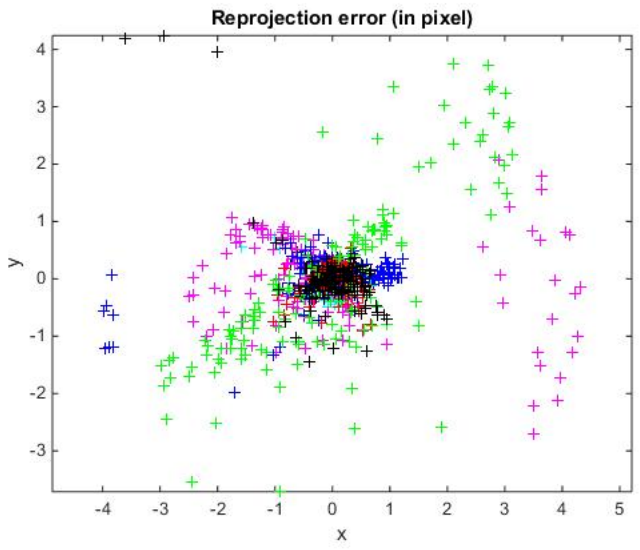
	Value	Error
Focal Length	fc=[664.46682 667.85229]	±[3.63508 3.52637]
Principal point	cc=[310.28364 239.67565]	±[5.09978 5.11723]
Skew	alpha_c=[0.00000] ,Angle of pixel axes=90.00000	± [0.00000]
Distortion	kc=[-0.28233 0.36707 0.00169 0.00131 0.00000]	± [0.02056 0.07917 0.00135 0.00125 0.00000]

fter reprojecting above resulting calibration to the images following were the results. (1st two)





The reprojection error for the initial calibration is as follows. (in the form of color-coded crosses)



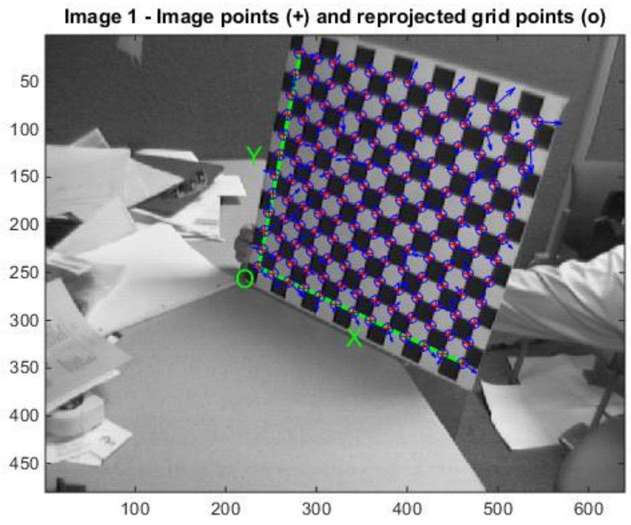
We recompute the image corners on all images using the re-projected grid as initial guess locations for the corners to reduce the error automatically. This was done using Recomp. corners. Then calibrate it again. This gives us improved results.

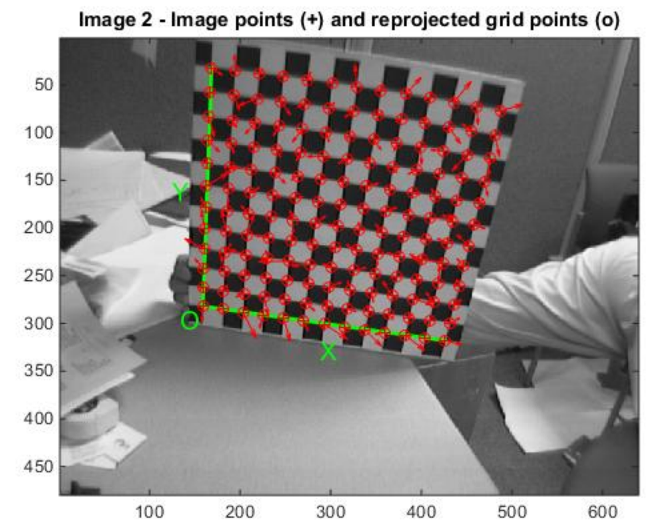
4.b Results after Final Calibration

Calibration results after optimization after initial calibration:

	Value	Error
Focal Length	fc = [657.32459 657.94394]	±[1.02139 0.96869]
Principal point	cc = [304.53771 240.83598]	±[1.45211 1.53204]
Skew	alpha_c = [0.00000] ,Angle of pixel axes = 90.00000	± [0.00000]
Distortion	kc = [-0.25623 0.13347 -0.00003 0.00026 0.00000]	± [0.00556 0.02042 0.00035 0.00030 0.00000]

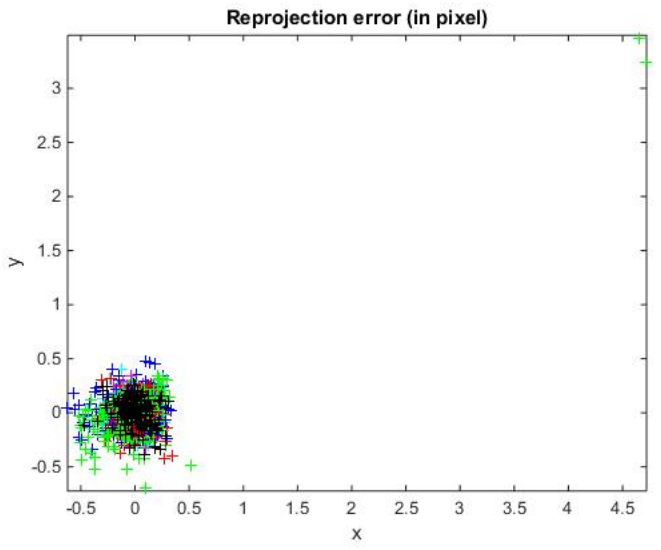
After projecting final calibration results to the images, (1st two)





We cant see a much difference in these pictures by just looking at it. But in pixel level the error has reduced a lot.

Reprojection error in final calibration:



5 Question 5

All code used can be found in Appendix A [5].

5.a Part A

Converted the images into gray scale to use in edge function to find edges of the image. (We have used the image toolbox function edge using sobel method. Following was the result.

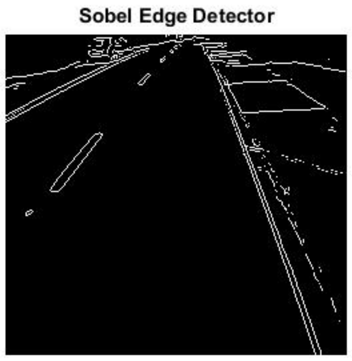


Figure 25: Sobel Edge Detector

Converted grey image has been used image toolbox function, edge using Canny method to find edges. Following was the result.

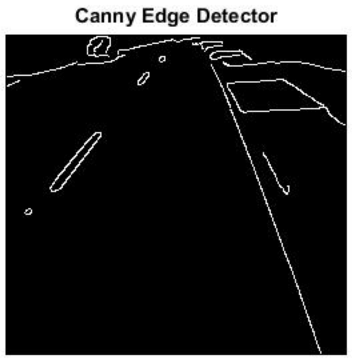


Figure 26: Canny Edge Detector

5.b Part B

Converted grey image has been used in image toolbox function, corners using Harris method to find corners. Following was the result.

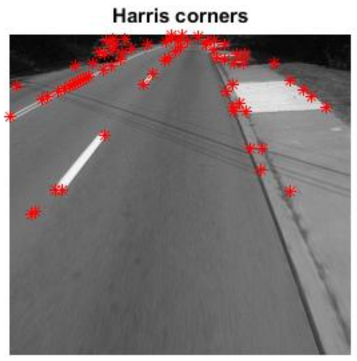


Figure 27: Harris Corners

5.c Part C

As sift function required input to be a gray image and code is written to read the file from the folder we have saved the converted gray image so that it can open it to use in sift function. Following was the result.

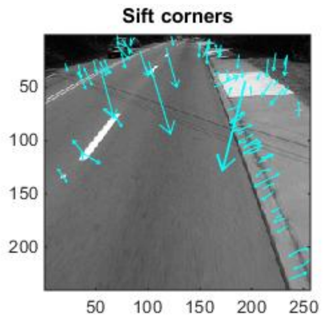


Figure 28: Sift Corners

5.d Part D

First I have used given match function to match the given two photos of Whitehouse using sift corners. Following was the result.

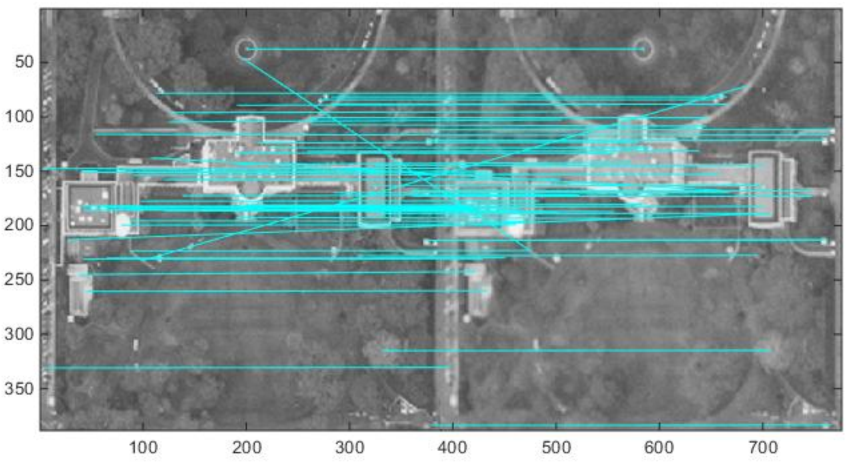


Figure 29: Feature Matching 1

The code we to match features using corners identified using harris corners. We have used feature matching method used in matlab tutorial for matlab image processing toolbox. See Appendix A [5] for the code. This works perfectly with similar (same) images. Result was as follows:

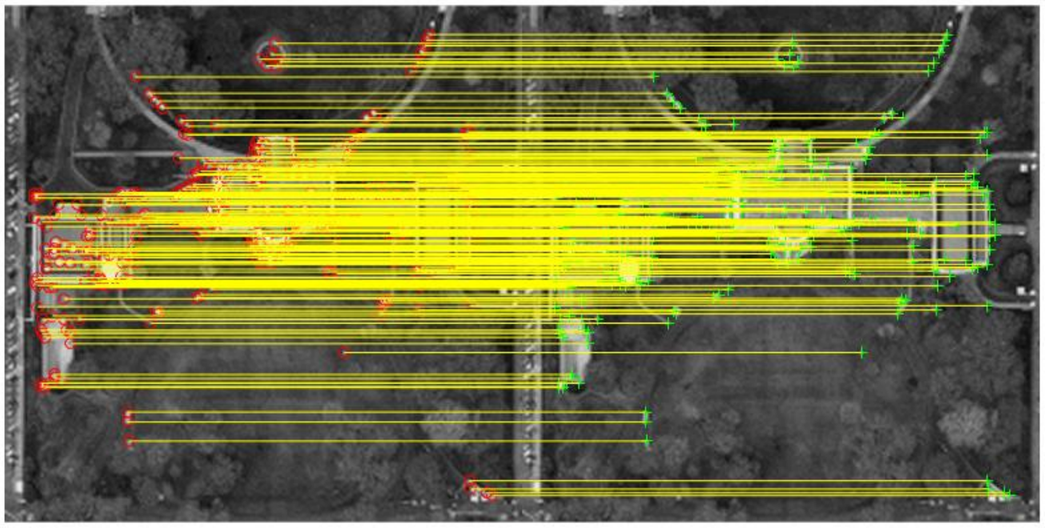


Figure 30: Feature Matching 2

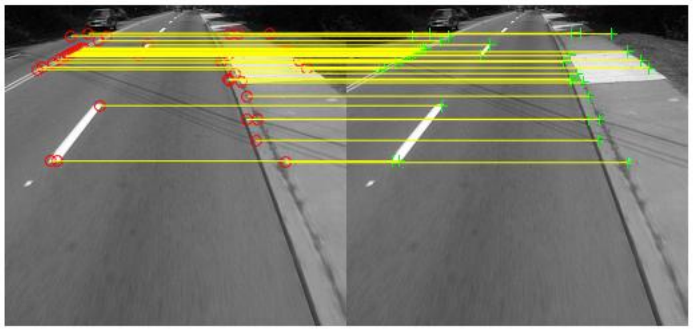


Figure 31: Feature Matching 3

For Slightly different images with similar features result was as follows. For example 1 the result must be improved a lot.

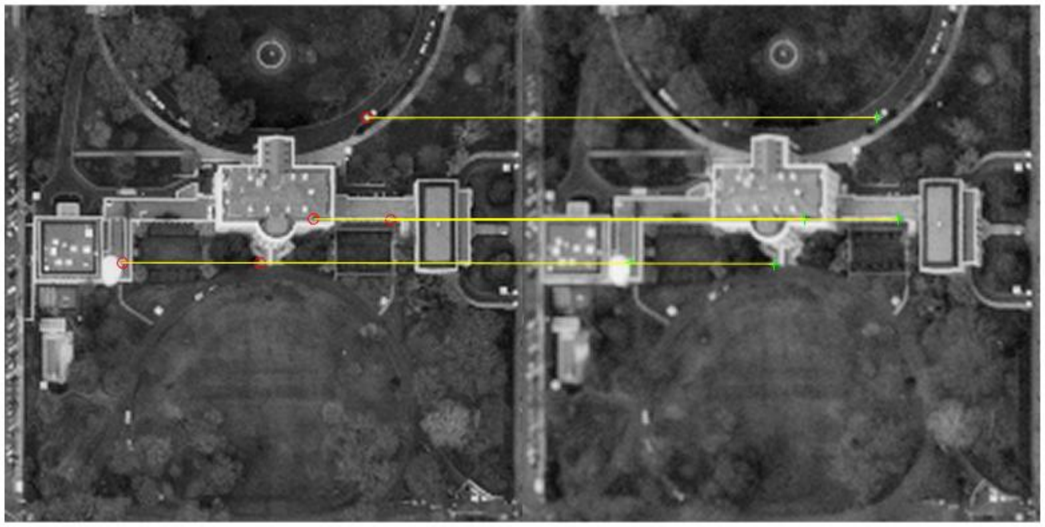


Figure 32: Different Matching 1

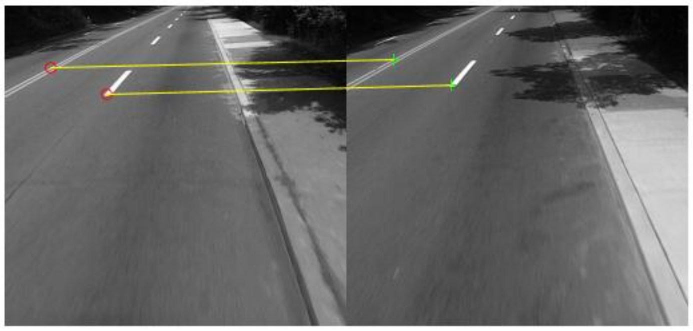


Figure 33: Different Matching 2

Code Listing

See Appendix A [5] for all code used.

6 Question 6

This section heavily involved MATLAB code. See Appendix A [6] for code listings.

6.a Image Processing

The image taken from the camera will be cropped to isolate the working space. As the camera is fixed we can use a constant coordinates for cropping. Then convert the image into double and used adapthisteq to enhance the contrast of the image using histogram equalization. This makes it easier to work on the image as the image was dark and histogram was not equally distributed. Top faces of the boxes were black. In a grey image which is double 0 is for black and 1 is for white. So we used a threshold of 0.1 to extract top faces from the image. We remove all the areas smaller than 1000 pixels to get rid of noise and used a square structuring element to morphologically close founded areas and fill all holes.

6.b Finding Centroids

To find centroids we used regionprops. First we found the coordinates of the fiducial boxes and remove them from the centroid list. (If only the fiducials were present code will give a warning).To match the coordinates with given world coordinate system we used perpendicular distance to the each centroid from the line connecting two fiducial boxes. For example y distance can be found by the perpendicular distance from the line connecting two fiducial boxes which are vertical.

6.c Finding Orientation

Several methods were tested to find the block orientation, mainly using different sub commands of regionprops. Firstly, we attempted to use the obviously named Orientation command. This met with limited success, as the orientation function works by forming an ellipse around the centroids, and then determining the angle between the major ellipse axis and the horizontal:

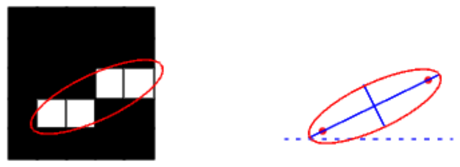


Figure 34: RegionProps Orientation

Due to the fact that the identified boxes where not perfectly smooth, this meant that the ellipse major axis did not necessarily run along the boxes diagonal, leading to somewhat erratic angle results.

The second method was to instead look at the Extrema of the boxes, as shown below:

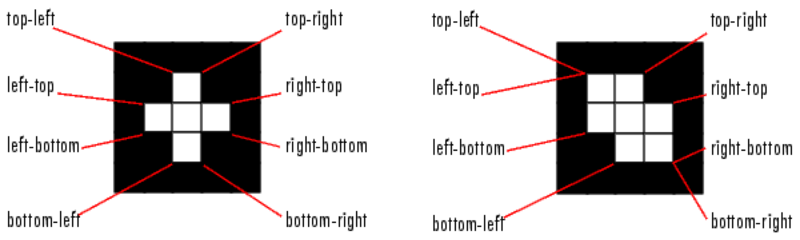


Figure 35: RegionProps Extrema

With this, the irregular nature of the boxes is tempered slightly by the eight extreme points, enabling the selection of the longest side and the determining of the angle it made with the horizontal.

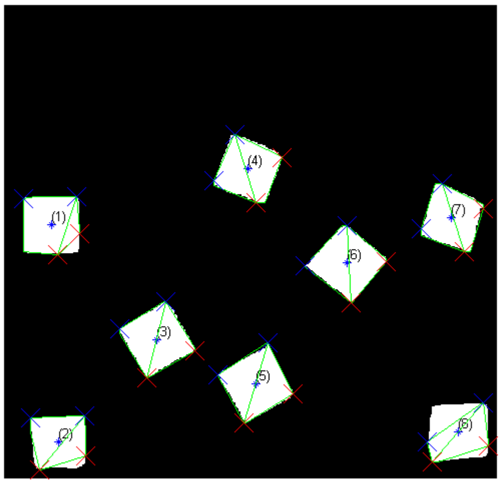


Figure 36: RegionProps Extrema in Action

This method proved simple, yet effective. Though it was not fully implemented, programming the system ti intelligently determine the longest side and determine the angle based on that was considered. However, as can be seen in the above figure, the longest side was not always a side at all. Extremas flaw was that what were determined to be the extreme points were not always the corners, sometimes resulting in diagonals being formed, making strange triangles out of the boxes. The accuracy of the extrema was limited to the clarity of the image, and we had reached the limit of image sharpening using our current techniques - any adjustment of variables resulted in warped boxes. However, it was determined that the inaccuracies in the angle derived from the extrema was within the margin of error acceptable for foam blocks being picked up by a robotic arm.

6.d Performance

When our codes was run on the robotic arm, it performed nearly perfectly (after the adjustment of a few unnoticed bugs that prevented operation at all). Unfortunately, when carrying the final block and attempting to stack it, the block was not quite lifted high enough and clipped the tower, knocking it over. Otherwise, the code performed near-perfectly. it correctly identified centroids and orientation for the majority of the blocks, with only a single block having the correct centroids but slightly incorrect orientation. However, this error was within the limits of the arm and foam blocks, and the block was picked up nethertheless.

Code Listing

See Appendix A [6] for all code used.

7 Appendix A

7.1 Question 1

7.1.i Linear Data Set Generator

```
close all
clear
clc


%for y = x without any error
x_0to10 = [0.00 0.10    0.20    0.30    0.40    0.50    0.60    0.70    0.80    0.90    1.00    1.10
    ↪    1.20    1.30    1.40    1.50    1.60    1.70    1.80    1.90    2.00    2.10    2.20    2
    ↪ .30    2.40    2.50    2.60    2.70    2.80    2.90    3.00    3.10    3.20    3.30    3.40
    ↪    3.50    3.60    3.70    3.80    3.90    4.00    4.10    4.20    4.30    4.40    4.50    4
    ↪ .60    4.70    4.80    4.90    5.00    5.10    5.20    5.30    5.40    5.50    5.60    5.70
    ↪    5.80    5.90    6.00    6.10    6.20    6.30    6.40    6.50    6.60    6.70    6.80    6
    ↪ .90    7.00    7.10    7.20    7.30    7.40    7.50    7.60    7.70    7.80    7.90    8.00
    ↪    8.10    8.20    8.30    8.40    8.50    8.60    8.70    8.80    8.90    9.00    9.10    9
    ↪ .20    9.30    9.40    9.50    9.60    9.70    9.80    9.90    10.00];
y_0to10 = [0.00 0.10    0.20    0.30    0.40    0.50    0.60    0.70    0.80    0.90    1.00    1.10
    ↪    1.20    1.30    1.40    1.50    1.60    1.70    1.80    1.90    2.00    2.10    2.20    2
    ↪ .30    2.40    2.50    2.60    2.70    2.80    2.90    3.00    3.10    3.20    3.30    3.40
    ↪    3.50    3.60    3.70    3.80    3.90    4.00    4.10    4.20    4.30    4.40    4.50    4
    ↪ .60    4.70    4.80    4.90    5.00    5.10    5.20    5.30    5.40    5.50    5.60    5.70
    ↪    5.80    5.90    6.00    6.10    6.20    6.30    6.40    6.50    6.60    6.70    6.80    6
    ↪ .90    7.00    7.10    7.20    7.30    7.40    7.50    7.60    7.70    7.80    7.90    8.00
    ↪    8.10    8.20    8.30    8.40    8.50    8.60    8.70    8.80    8.90    9.00    9.10    9
    ↪ .20    9.30    9.40    9.50    9.60    9.70    9.80    9.90    10.00];
y_1to11 = [1.00 1.10    1.20    1.30    1.40    1.50    1.60    1.70    1.80    1.90    2.00    2.10
    ↪    2.20    2.30    2.40    2.50    2.60    2.70    2.80    2.90    3.00    3.10    3.20    3
    ↪ .30    3.40    3.50    3.60    3.70    3.80    3.90    4.00    4.10    4.20    4.30    4.40
    ↪    4.50    4.60    4.70    4.80    4.90    5.00    5.10    5.20    5.30    5.40    5.50    5
    ↪ .60    5.70    5.80    5.90    6.00    6.10    6.20    6.30    6.40    6.50    6.60    6.70
    ↪    6.80    6.90    7.00    7.10    7.20    7.30    7.40    7.50    7.60    7.70    7.80    7
    ↪ .90    8.00    8.10    8.20    8.30    8.40    8.50    8.60    8.70    8.80    8.90    9.00
    ↪    9.10    9.20    9.30    9.40    9.50    9.60    9.70    9.80    9.90    10.00   10.10
    ↪ 10.20   10.30   10.40   10.50   10.60   10.70   10.80   10.90   11.00];

% y = x from 0-5 then y = -x from 5-10
y_inverseV = [0.00  0.10    0.20    0.30    0.40    0.50    0.60    0.70    0.80    0.90    1.00
    ↪ 1.10    1.20    1.30    1.40    1.50    1.60    1.70    1.80    1.90    2.00    2.10    2.20
    ↪    2.30    2.40    2.50    2.60    2.70    2.80    2.90    3.00    3.10    3.20    3.30    3
    ↪ .40    3.50    3.60    3.70    3.80    3.90    4.00    4.10    4.20    4.30    4.40    4.50
    ↪    4.60    4.70    4.80    4.90    5.00    4.90    4.80    4.70    4.60    4.50    4.40    4
    ↪ .30    4.20    4.10    4.00    3.90    3.80    3.70    3.60    3.50    3.40    3.30    3.20
    ↪    3.10    3.00    2.90    2.80    2.70    2.60    2.50    2.40    2.30    2.20    2.10    2
    ↪ .00    1.90    1.80    1.70    1.60    1.50    1.40    1.30    1.20    1.10    1.00    0.90
    ↪    0.80    0.70    0.60    0.50    0.40    0.30    0.20    0.10    0.00];

%zigzag
zigzag_broken = [0.00   0.10    0.20    0.30    0.40    0.50    0.60    0.70    0.80    0.90    1.00
    ↪    1.10    1.20    1.30    1.40    1.50    1.60    1.70    1.80    1.90    2.00    2.10    2
    ↪ .20    2.30    2.40    2.50    2.40    2.30    2.20    2.10    2.00    1.90    1.80    1.70
    ↪    1.60    1.50    1.40    1.30    1.20    1.10    1.00    0.90    0.80    0.70    0.60    0
    ↪ .50    0.40    0.30    0.20    0.10    0.00    4.90    4.80    4.70    4.60    4.50    4.40
    ↪    4.30    4.20    4.10    4.00    3.90    3.80    3.70    3.60    3.50    3.40    3.30    3
    ↪ .20    3.10    3.00    2.90    2.80    2.70    2.60    5.60    5.70    5.80    5.90    6.00
    ↪    6.10    6.20    6.30    6.40    6.50    6.60    6.70    6.80    6.90    7.00    7.10    7
    ↪ .20    7.30    7.40    7.50    7.60    7.70    7.80    7.90    8.00    8.10];
zigzag  = [0.00 0.10    0.20    0.30    0.40    0.50    0.60    0.70    0.80    0.90    1.00    1.10
    ↪    1.20    1.30    1.40    1.50    1.60    1.70    1.80    1.90    2.00    2.10    2.20    2
    ↪ .30    2.40    2.50    2.40    2.30    2.20    2.10    2.00    1.90    1.80    1.70    1.60
    ↪    1.50    1.40    1.30    1.20    1.10    1.00    0.90    0.80    0.70    0.60    0.50    0
    ↪ .40    0.30    0.20    0.10    0.00    0.10    0.20    0.30    0.40    0.50    0.60    0.70
    ↪    0.80    0.90    1.00    1.10    1.20    1.30    1.40    1.50    1.60    1.70    1.80    1
    ↪ .90    2.00    2.10    2.20    2.30    2.40    2.50    2.40    2.30    2.20    2.10    2.00
    ↪    1.90    1.80    1.70    1.60    1.50    1.40    1.30    1.20    1.10    1.00    0.90    0
    ↪ .80    0.70    0.60    0.50    0.40    0.30    0.20    0.10    0.00];

nonperpzz = [0.00   0.20    0.40    0.60    0.80    1.00    1.20    1.40    1.60    1.80    2.00
    ↪ 2.20    2.40    2.60    2.80    3.00    3.20    3.40    3.60    3.80    4.00    4.20    4.40
    ↪     4.60    4.80    5.00    4.80    4.60    4.40    4.20    4.00    3.80    3.60    3.40    3
    ↪ .20    3.00    2.80    2.60    2.40    2.20    2.00    1.80    1.60    1.40    1.20    1.00
    ↪     0.80    0.60    0.40    0.20    0.00    0.20    0.40    0.60    0.80    1.00    1.20    1
    ↪ .40    1.60    1.80    2.00    2.20    2.40    2.60    2.80    3.00    3.20    3.40    3.60
    ↪     3.80    4.00    4.20    4.40    4.60    4.80    5.00    4.80    4.60    4.40    4.20    4
    ↪ .00    3.80    3.60    3.40    3.20    3.00    2.80    2.60    2.40    2.20    2.00    1.80
    ↪     1.60    1.40    1.20    1.00    0.80    0.60    0.40    0.20    0.00];

% vertical lines
%for x = 0
x_0 = [0.00 0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
    ↪ 0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
    ↪     0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0
    ↪ .00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
    ↪     0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0
    ↪ .00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
    ↪     0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0
    ↪ .00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
    ↪     0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00];

%for x = 2
x_2 = [2.00 2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00
    ↪ 2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00
    ↪     2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2
    ↪ .00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00
    ↪     2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2
    ↪ .00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00
    ↪     2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2
    ↪ .00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00
    ↪     2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00];

% horizontal lines
%for y = 0
y_0 = [0.00 0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
    ↪ 0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
    ↪     0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0
    ↪ .00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
    ↪     0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0
    ↪ .00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
    ↪     0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0
    ↪ .00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
    ↪     0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00];

%for y = 2
y_2 = [2.00 2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00
    ↪ 2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00
    ↪     2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2
    ↪ .00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00
    ↪     2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2
    ↪ .00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00
    ↪     2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2
    ↪ .00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00
    ↪     2.00    2.00    2.00    2.00    2.00    2.00    2.00    2.00];
```

7.1.ii Test 00

```matlab
%test00
%horizontal line test (no error)
close all
clear
clc

diary './test_results'
diary ON
disp 'Q1 Test Result Corner Value outputs from LineSeg'
% import basic test data
linearDataSetGenerator;

xvals = x_0to10;    %change these to change test data
yvals = y_2;

%plot raw data
plot(xvals, yvals, 'rx'); %mark all true data values with red x

%plot our line approximation
errorThreshold = 0.1;
disp 'test00: horizontal line test (no error)'
vertices = lineseg(xvals, yvals, errorThreshold);
diary OFF
```

7.1.iii Test 01

```
%test01
%vertical line test (no error)
close all
clear
clc

diary 'test_results'
diary ON

% import basic test data
linearDataSetGenerator;

xvals = x_2;     %change these to change test data
yvals = y_0to10;

%plot raw data
plot(xvals, yvals, 'rx'); %mark all true data values with red x

%plot our line approximation
errorThreshold = 0.1;
disp 'test01: vertical line test (no error)'
vertices = lineseg(xvals, yvals, errorThreshold);
diary OFF
```

7.1.iv Test 02

```
%test02
%simple slope line test (no error)
close all
clear
clc

diary 'test_results'
diary ON

% import basic test data
linearDataSetGenerator;

xvals = x_0to10;    %change these to change test data
yvals = y_0to10;

%plot raw data
plot(xvals, yvals, 'rx'); %mark all true data values with red x

%plot our line approximation
errorThreshold = 0.1;
disp 'test02: simple slope line test (no error)'
vertices = lineseg(xvals, yvals, errorThreshold);
diary OFF
```

7.1.v Test 03

```
%test03
%slope line test (no error)
close all
clear
clc

diary 'test_results'
diary ON

% import basic test data
linearDataSetGenerator;

xvals = x_0to10;    %change these to change test data
yvals = y_1to11;

%plot raw data
plot(xvals, yvals, 'rx'); %mark all true data values with red x

%plot our line approximation
errorThreshold = 0.1;
disp 'test03: slope line test (no error)'
vertices = lineseg(xvals, yvals, errorThreshold);
diary OFF
```

7.1.vi Test 04

```
%test04
%inverse V LSM test (no error)
close all
clear
clc

diary 'test_results'
diary ON
% import basic test data
linearDataSetGenerator;

xvals = x_0to10;    %change these to change test data
yvals = y_inverseV;

%plot raw data
plot(xvals, yvals, 'rx'); %mark all true data values with red x

%plot our line approximation
errorThreshold = 0.1;
disp 'test04: vertical inverse V LSM test (no error)'
vertices = lineseg(xvals, yvals, errorThreshold);
diary OFF
```

7.1.vii Test 05

```
%test05
%vertical inverse V LSM test (no error)
close all
clear
clc

diary 'test_results'
diary ON
% import basic test data
linearDataSetGenerator;

yvals = y_0to10;     %change these to change test data
xvals = y_inverseV;

%plot raw data
plot(xvals, yvals, 'rx'); %mark all true data values with red x

%plot our line approximation
errorThreshold = 0.1;
disp 'test05: vertical inverse V LSM test (no error)'
vertices = lineseg(xvals, yvals, errorThreshold);
diary OFF
```

7.1.viii Test 06

```
%test06
%horizontal zig zag test (no error)
close all
clear
clc

diary 'test_results'
%diary ON
% import basic test data
linearDataSetGenerator;

xvals = x_0to10;    %change these to change test data
yvals = zigzag;

%plot raw data
plot(xvals, yvals, 'rx'); %mark all true data values with red x

%plot our line approximation
errorThreshold = 0.1;
disp 'test06: horizontal zig zag test (no error)'
vertices = lineseg(xvals, yvals, errorThreshold);
diary OFF
```

7.1.ix Test 07

```
%test07
%broken horizontal zig zag test (no error)
close all
clear
clc

diary 'test_results'
diary ON
% import basic test data
linearDataSetGenerator;

xvals = x_0to10;    %change these to change test data
yvals = zigzag_broken;

%plot raw data
plot(xvals, yvals, 'rx'); %mark all true data values with red x

%plot our line approximation
errorThreshold = 0.1;
disp 'test07: broken horizontal zig zag test (no error)'
vertices = lineseg(xvals, yvals, errorThreshold);
diary OFF
```

7.1.x Test 08

```
%test08
%vertical zig zag test (no error)
close all
clear
clc

diary 'test_results'
diary ON
% import basic test data
linearDataSetGenerator;

yvals = y_0to10;    %change these to change test data
xvals = zigzag;

%plot raw data
plot(xvals, yvals, 'rx'); %mark all true data values with red x

%plot our line approximation
errorThreshold = 0.1;
disp 'test08: vertical zig zag test (no error)'
vertices = lineseg(xvals, yvals, errorThreshold);
diary OFF
```

7.1.xi Test 09

```
%test09
%vertical zig zag test (no error)
close all
clear
clc

diary 'test_results'
diary ON
% import basic test data
linearDataSetGenerator;

xvals = x_0to10;    %change these to change test data
yvals = nonperpzz;

%plot raw data
plot(xvals, yvals, 'rx'); %mark all true data values with red x

%plot our line approximation
errorThreshold = 0.1;
disp 'test09: non-perpendicular zigzag tst (no error)'
vertices = lineseg(xvals, yvals, errorThreshold);
diary OFF
```

7.1.xii Test 10

```
%test10
%vertical zig zag test (no error)
close all
clear
clc

diary 'test_results'
diary ON
% import basic test data
linearDataSetGenerator;

xvals = x_0to10;    %change these to change test data
yvals = nonperpzz;

%plot raw data
plot(xvals, yvals, 'rx'); %mark all true data values with red x

%plot our line approximation
errorThreshold = 0.1;
disp 'test10: corner finder test on non-perpendicular zigzag test (no error)'
corners = findcorners(xvals, yvals, errorThreshold);
diary OFF
```

7.1.xiii Test 11

```
%test11
%vertical zig zag test (no error)
close all
clear
clc

diary 'test_results'
diary ON
% import basic test data
linearDataSetGenerator;

xvals = x_0to10;    %change these to change test data
yvals = zigzag;

%plot raw data
plot(xvals, yvals, 'rx'); %mark all true data values with red x

%plot our line approximation
errorThreshold = 0.1;
disp 'test11: corner finder test on perpendicular zigzag test (no error)'
corners = findcorners(xvals, yvals, errorThreshold);
diary OFF
```

7.1.xiv Test Results

```
Q1 Test Result Corner Value outputs from LineSeg
test00: horizontal line test (no error)
test01: vertical line test (no error)
test02: simple slope line test (no error)
test03: slope line test (no error)
test04: vertical inverse V LSM test (no error)
test05: vertical inverse V LSM test (no error)
test06: horizontal zig zag test (no error)
test07: broken horizontal zig zag test (no error)
test08: vertical zig zag test (no error)
test09: non-perpendicular zigzag tst (no error)
test10: corner finder test on non-perpendicular zigzag test (no error)

vertices =

         0         0
    2.5000    5.0000
    5.0000         0
    7.5000    5.0000
   10.0000         0


vectors =

    2.5000    5.0000
    2.5000   -5.0000
    2.5000    5.0000
    2.5000   -5.0000

no corners

corners =

N/A


cornerCount =

     0

test11: corner finder test on perpendicular zigzag test (no error)

vertices =

         0         0
    2.5000    2.5000
    5.0000         0
    7.5000    2.5000
   10.0000         0


vectors =

    2.5000    2.5000
    2.5000   -2.5000
    2.5000    2.5000
    2.5000   -2.5000


corners =

    2.5000    2.5000
    5.0000         0
    7.5000    2.5000


cornerCount =

         3
```

7.2 Question 2

7.3 Question 3

7.3.i showICP

```
% test the ICP algorithm
% Author: Chieh-Chih (Bob) Wang [bob.wang@cas.edu.au]
% Created: April 12, 2005.
% Last Modified: April 12, 2005.


clear
close all
clc

% load laser files
laser_scans=load('datasets\captureScanshornet.txt');
t0 = laser_scans(1,1);

% Scan A
i = 500;
xA = zeros(1);
yA = zeros(1);
for j = 2:size(laser_scans,2)   %Map
 range = laser_scans(i,j) / 1000;
 bearing = ((j-1)/2 - 90)*pi/180;
 if (range < 75)
     xA = [xA range*cos(bearing)];
     yA = [yA range*sin(bearing)];
 end
end

% Scan B
i = 520;
xB = zeros(1);
yB = zeros(1);
for j = 2:size(laser_scans,2)   %Initial Guess (source? stay 'constant'? trasformed by ICP)
 range = laser_scans(i,j) / 1000;
 bearing = ((j-1)/2 - 90)*pi/180;
 if (range < 75)
     xB = [xB range*cos(bearing)];
     yB = [yB range*sin(bearing)];
 end
end

deltaPose = zeros(3,1);

[deltaPose_bar, deltaPose_bar_Cov, N] = ICPv4(deltaPose, [xA;yA], [xB;yB]); %ICP

newB = head2tail_no_theta(deltaPose_bar, [xB;yB]);
new_xB = newB(1,:);
new_yB = newB(2,:);

figure
clf
hold on
plot(xA,yA,'b+')
plot(xB,yB,'r.')
plot(new_xB,new_yB,'kx')
axis equal
legend('Map','initial guess','ICP')
xlabel('X (meter)')
ylabel('Y (meter)')
title('The ICP algorithm')
```

7.3.ii Modified showICP

```
% test the ICP algorithm
% Author: Chieh-Chih (Bob) Wang [bob.wang@cas.edu.au]
% Created: April 12, 2005.
% Last Modified: April 12, 2005.
%
%Modified by James Ferris to Show ICP data in a plot


clear
close all
clc

% load laser files
laser_scans=load('datasets\captureScanshornet.txt');
t0 = laser_scans(1,1);

% Scan A
i = 500;
xA = zeros(1);
yA = zeros(1);
for j = 2:size(laser_scans,2)   %Map
 range = laser_scans(i,j) / 1000;
 bearing = ((j-1)/2 - 90)*pi/180;
 if (range < 75)
     xA = [xA range*cos(bearing)];
     yA = [yA range*sin(bearing)];
 end
end

% Scan B
i = 520;
xB = zeros(1);
yB = zeros(1);
for j = 2:size(laser_scans,2)   %Initial Guess (source? stay 'constant'? trasformed by ICP)
 range = laser_scans(i,j) / 1000;
 bearing = ((j-1)/2 - 90)*pi/180;
 if (range < 75)
     xB = [xB range*cos(bearing)];
     yB = [yB range*sin(bearing)];
 end
end

deltaPose = zeros(3,1);

[deltaPose_bar, deltaPose_bar_Cov, N] = ICPv4(deltaPose, [xA;yA], [xB;yB]); %ICP

%%
%Added to show robot position and orientation
figure
clf
hold on
Pose = deltaPose_bar;
h = 0.5;
Pose2 = [Pose(1)+h*cos(Pose(3)), Pose(2)+h*sin(Pose(3))];
plot(Pose(1),Pose(2),'gx');
plot(Pose2(1),Pose2(2),'go');
%%

newB = head2tail_no_theta(deltaPose_bar, [xB;yB]);
new_xB = newB(1,:);
new_yB = newB(2,:);

plot(xA,yA,'b+')
plot(xB,yB,'r.')
plot(new_xB,new_yB,'kx')
axis equal
legend('Robot Position', 'Robot Heading', 'Map','Initial Guess','ICP')
```

```
xlabel('X (meter)')
ylabel('Y (meter)')
title('The ICP algorithm')
```

7.3.iii laserShowAcfr

```
%
% Author: Stefan Williams (stefanw@acfr.usyd.edu.au)
%
%

clear
close all
clc

% load laser files
laser_scans=load('datasets\captureScanshornet.txt');

t0 = laser_scans(1,1);

figure
for i = 1:length(laser_scans)
     tlaser = laser_scans(i,1) - t0;
     xpoint = zeros(1);
     ypoint = zeros(1);
     for j = 2:size(laser_scans,2)
         range = laser_scans(i,j) / 1000;
         bearing = ((j-1)/2 - 90)*pi/180;
         if (range < 75)
             xpoint = [xpoint range*cos(bearing)];
             ypoint = [ypoint range*sin(bearing)];
         end
     end
     plot(xpoint(:), ypoint(:), '.');
     axis equal;
     axis([0 10 -5 5]);
     xlabel('X (meter)')
     ylabel('Y (meter)')
     title(sprintf('ACFR indoor SICK data: scan %d',i))
     drawnow

%       if i == 500
%           pause;
%       elseif i == 520
%               pause;
%       end

end
```

7.3.iv Modified laserShowACFR

```
%
% Author: Stefan Williams (stefanw@acfr.usyd.edu.au)
%
%
%Modified by James Ferris to include ICP data in a plot

clear
close all
clc

% load laser files
laser_scans=load('datasets\captureScanshornet.txt');

t0 = laser_scans(1,1);

%%
%Initialise ICP data
i = 20; %Let the 'initial guess' be starting point + 20, as in part A
xB = zeros(1);
yB = zeros(1);
for j = 2:size(laser_scans,2)
    range = laser_scans(i,j) / 1000;
    bearing = ((j-1)/2 - 90)*pi/180;
    if (range < 75)
      xB = [xB range*cos(bearing)];
      yB = [yB range*sin(bearing)];
    end
end
deltaPose = zeros(3,1);

%%

figure
for i = 1:length(laser_scans)

     tlaser = laser_scans(i,1) - t0;
     xpoint = zeros(1);
     ypoint = zeros(1);
     for j = 2:size(laser_scans,2)
         range = laser_scans(i,j) / 1000;
         bearing = ((j-1)/2 - 90)*pi/180;
         if (range < 75)
             xpoint = [xpoint range*cos(bearing)];
             ypoint = [ypoint range*sin(bearing)];
         end
     end

     %%
     %Calculate ICP
     [deltaPose_bar, deltaPose_bar_Cov, N] = ICPv4(deltaPose, [xpoint;ypoint], [xB;yB]);
     newB = head2tail_no_theta(deltaPose_bar, [xB;yB]);
     new_xB = newB(1,:);
     new_yB = newB(2,:);

     if i == 1 || mod(i,20)== 0

         Pose = deltaPose_bar;
         h = 0.5;
         Pose2 = [Pose(1)+h*cos(Pose(3)), Pose(2)+h*sin(Pose(3))];
         hold on


         plot(xpoint(:), ypoint(:), '.');

         plot(xB,yB,'r.')
         plot(new_xB,new_yB,'kx')

```

```
        plot(Pose(1),Pose(2),'gx');
        plot(Pose2(1),Pose2(2),'go');


        %%


        axis equal;

        legend('Robot Position', 'Robot Heading', 'Map','Initial Guess','ICP')
        axis([0 10 -5 5]);
        xlabel('X (meter)')
        ylabel('Y (meter)')
        title(sprintf('ACFR indoor SICK data: scan %d',i))
        drawnow

        pause
    end

    xB = new_xB;
    yB = new_yB;

    clf

end
```

7.3.v ICPv4

```
function [deltaPose_bar, deltaPose_bar_Cov, N] = ICPv4(deltaPose, a, b)

% function: ICP algorithm version 3.0
% a: point set, 2 x Na
% b: point set, 2 x Nb
% Xab: the relative tranformation between a and b
%
%
%
% Author: Chieh-Chih (Bob) Wang [bobwang@cs.cmu.edu]
% Created: Dec. 2, 2002.
% Last Modified: Dec. 27, 2003.
% (C) 2002-2004 Chieh-Chih (Bob) Wang. All Rights Reserved.


max_delta_g = 0.01;
max_iter = 40;
WinSize = 80;

% Cell 5cm x 5cm
grid_size =   0.005; % Changing this value may affect the accuracy of the ICP algorithm.
Map_x_min =   min(a(1,:));
Map_y_min =   min(a(2,:));
Map_x_max =   max(a(1,:));
Map_y_max =   max(a(2,:));

% Step 1: Create a grid map foo speed up correspondence search
GridMap_X = ceil((Map_x_max - Map_x_min)/grid_size);
GridMap_Y = ceil((Map_y_max - Map_y_min)/grid_size);
GridMap = zeros(GridMap_X, GridMap_Y);

for k=1:size(a,2)
    [Map_i,Map_j] = XY2IJ(a(1,k), a(2,k), grid_size, Map_x_min, Map_y_min);
    if Map_i>0 & Map_i<= GridMap_X & Map_j>0 & Map_j<= GridMap_Y
        if GridMap(Map_i,Map_j)~= 0
            %disp(sprintf('points collide %d,%d', Map_i, Map_j))
        end
        GridMap(Map_i,Map_j) = k;
    end
end

WinSize_org = WinSize;
delta_g = 1000000000;
j=0;
g = deltaPose;

% method 2:
Z = [];
M = [];

NoMatch_flag = 0;

while (j < max_iter) & (delta_g > max_delta_g)
    j = j+1;
    old_g = g;
    % Step 1:

    % Finding Correspondence
    Match_Pairs = [];
    %New_Scan1_Index = Scan1_Index;
    %New_Scan2_Index = Scan2_Index;

    %WinSize = round(WinSize_org/j);
    WinSize = WinSize_org- 4*j;
    if WinSize < 2
        WinSize = 2;
    end

    for k=1:size(b,2)

        Point_X = head2tail_no_theta(g,b(:,k));

        [Map_i,Map_j] = XY2IJ(Point_X(1), Point_X(2), grid_size, Map_x_min, Map_y_min);
        % Search ...
        % Define search area
        % WinSize = 10;
        Win_i_min = Map_i - WinSize;
        if Win_i_min < 1
            Win_i_min = 1;
        end
        Win_i_max = Map_i + WinSize;
        if Win_i_max > GridMap_X
            Win_i_max = GridMap_X;
        end
        Win_j_min = Map_j - WinSize;
        if Win_j_min < 1
            Win_j_min = 1;
        end
        Win_j_max = Map_j + WinSize;
        if Win_j_max > GridMap_Y
            Win_j_max = GridMap_Y;
        end
        [Search_i, Search_j] = find(GridMap(Win_i_min:Win_i_max, Win_j_min:Win_j_max) > 0);
        if size(Search_i,1)>0
            min_dis = 100000000000000;
            match_index = 0;
            for m=1:size(Search_i,1)
                a_index = GridMap(Search_i(m)+Win_i_min-1, Search_j(m)+Win_j_min-1);
                dis = sqrt((Point_X(1) - a(1,a_index))^2 ...
                    + (Point_X(2) - a(2,a_index))^2);
                if (dis < min_dis)
                    min_dis = dis;
                    match_index = a_index;
                end
            end
            % method 1:
            Match_Pairs = [Match_Pairs; ...
                    b(1,k) b(2,k) a(1,match_index) a(2,match_index)];
            % method 2:
            Mk = [1 0 -Point_X(2,1); 0 1 Point_X(1,1)];
            M = [M; Mk];
            Z = [Z; Point_X - a(:,match_index)];

            %MatchedPoints(1,k) = 1;
            %New_Scan1_Index(1, match_index) = 2; % see Readme.txt for the definition
            %New_Scan2_Index(1, k) = 2;
        end
    end
    %Method 1: the closed form solution without covariance estimate
    N = size(Match_Pairs,1);
    %disp(sprintf('Inside ICPv4: iter %d, match pairs %d',j, N));
    if N == 0
        NoMatch_flag = 1;
        break
    end

    X2_bar = sum(Match_Pairs(:,1))/N;
    Y2_bar = sum(Match_Pairs(:,2))/N;
    X1_bar = sum(Match_Pairs(:,3))/N;
    Y1_bar = sum(Match_Pairs(:,4))/N;
    Sx2x1 = sum((Match_Pairs(:,1) - X2_bar).*(Match_Pairs(:,3) - X1_bar));
    Sy2y1 = sum((Match_Pairs(:,2) - Y2_bar).*(Match_Pairs(:,4) - Y1_bar));
    Sx2y1 = sum((Match_Pairs(:,1) - X2_bar).*(Match_Pairs(:,4) - Y1_bar));
    Sy2x1 = sum((Match_Pairs(:,2) - Y2_bar).*(Match_Pairs(:,3) - X1_bar));

    g(3,1) = atan2(Sx2y1-Sy2x1, Sx2x1+ Sy2y1);
    g(1,1) = X1_bar - (X2_bar*cos(g(3,1))- Y2_bar*sin(g(3,1)));
    g(2,1) = Y1_bar - (X2_bar*sin(g(3,1))+ Y2_bar*cos(g(3,1)));

    delta_g = sqrt((old_g(1) - g(1))^2 + (old_g(2) - g(2))^2);

    %Method 2:
%     InvMM = inv(M'*M);
%     D_bar = InvMM*M'*Z;
%     ZminusMD_bar =Z-M*D_bar;
%     s_square = ZminusMD_bar'*ZminusMD_bar/(2*N-3);
%     Cov_ICP = s_square*InvMM;

    %pause
end
% Xab_bar = g+D_bar;
% Xab_Cov = Cov_ICP;
if NoMatch_flag == 0
    InvMM = inv(M'*M);
    D_bar = InvMM*M'*Z;
    ZminusMD_bar =Z-M*D_bar;
    s_square = ZminusMD_bar'*ZminusMD_bar/(2*N-3);
    Cov_ICP = s_square*InvMM;

    deltaPose_bar = g;
    deltaPose_bar_Cov = Cov_ICP;
else
    deltaPose_bar = [];
    deltaPose_bar_Cov = [];
end
```

7.3.vi XY2IJ

```
function [Map_i,Map_j] = XY2IJ(x, y, grid_size, Map_x_min, Map_y_min)

% XY2IJ
%
% Author: Chieh-Chih (Bob) Wang [bobwang@cs.cmu.edu]
% Created: Oct. 31, 2002.
% Modified: Nov. 6, 2003.
% (c) 2002-2003 Chieh-Chih Wang. All Rights Reserved.

% Changed to the vec form
Map_i = round((x - Map_x_min)/grid_size + 0.5);
Map_j = round((y - Map_y_min)/grid_size + 0.5);
```

7.3.vii head2tail no theta

```
function Xik = head2tail_no_theta(Xij, Xjk)

% Compounding Operation: head2tail
%
% Input:
%   Xij = [x_ij; y_ij; theta_ij]
%   Xjk = [x_jk; y_jk]
% Output:
%   Xik = [x_ik; y_ik]
%
% Author: Chieh-Chih (Bob) Wang [bobwang@cs.cmu.edu]
% Created: Nov. 8, 2002.
% Modified: Nov. 6, 2003.


cosTheta_ij = cos(Xij(3,1));
sinTheta_ij = sin(Xij(3,1));

% Xik(1,1) = Xjk(1,1)*cosTheta_ij - Xjk(2,1)*sinTheta_ij + Xij(1,1);
% Xik(2,1) = Xjk(1,1)*sinTheta_ij + Xjk(2,1)*cosTheta_ij + Xij(2,1);

% Change for Vec...
Xik(1,:) = Xjk(1,:)*cosTheta_ij - Xjk(2,:)*sinTheta_ij + Xij(1,1);
Xik(2,:) = Xjk(1,:)*sinTheta_ij + Xjk(2,:)*cosTheta_ij + Xij(2,1);
```

7.4 Question 4

7.5 Question 5

7.5.i Append Images

```
% im = appendimages(image1, image2)
%
% Return a new image that appends the two images side-by-side.

function im = appendimages(image1, image2)

% Select the image with the fewest rows and fill in enough empty rows
%   to make it the same height as the other image.
rows1 = size(image1,1);
rows2 = size(image2,1);

if (rows1 < rows2)
     image1(rows2,1) = 0;
else
     image2(rows1,1) = 0;
end

% Now append both images side-by-side.
im = [image1 image2];
```

7.5.ii Feature Matching

```
%This code is using harris algorithm to find corners and extractFeatures
%and matchFeatures Functions in matlab image toolbox to match features
%in two images.

close all
clear
clc
x=0;
y=0;


I1=imread('whitehouse.left.png');
I2=imread('whitehouse.right.png');

%Check whether the image is RGB and convert that to gray
if length(size(I1))==3 & length(size(I2))==3;
    BW1=rgb2gray(I1);
    BW2=rgb2gray(I2);
    x=1;
else
BW1=I1;
BW2=I2;
end

im=appendimages(I1,I2); %create a one image using two images

p1 = detectHarrisFeatures(BW1); %Find corners using Harris Algorithm
p2 = detectHarrisFeatures(BW2);


[features1, valid_points1] = extractFeatures(BW1, p1); %Extract Features and valid points using
    ↪ detected corners
[features2, valid_points2] = extractFeatures(BW2, p2);

indexPairs = matchFeatures(features1, features2,'MatchThreshold',80); %Finding Matching Features of
    ↪ two images

matchedPoints1 = valid_points1(indexPairs(:, 1), :);
matchedPoints2 = valid_points2(indexPairs(:, 2), :);

figure; showMatchedFeatures(BW1, BW2, matchedPoints1, matchedPoints2,'montage'); %show matching
    ↪ features
```

7.5.iii Harris

```
% HARRIS - Harris corner detector
%
% Usage:  [cim, r, c] = harris(im, sigma, thresh, radius, disp)
%
% Arguments:
%            im     - image to be processed.
%            sigma  - standard deviation of smoothing Gaussian. Typical
%                     values to use might be 1-3.
%            thresh - threshold (optional). Try a value ~1000.
%            radius - radius of region considered in non-maximal
%                     suppression (optional). Typical values to use might
%                     be 1-3.
%            disp   - optional flag (0 or 1) indicating whether you want
%                     to display corners overlayed on the original
%                     image. This can be useful for parameter tuning.
%
% Returns:
%            cim    - binary image marking corners.
%            r      - row coordinates of corner points.
%            c      - column coordinates of corner points.
%
% If thresh and radius are omitted from the argument list 'cim' is returned
% as a raw corner strength image and r and c are returned empty.

% References:
% C.G. Harris and M.J. Stephens. "A combined corner and edge detector",
% Proceedings Fourth Alvey Vision Conference, Manchester.
% pp 147-151, 1988.
%
% Alison Noble, "Descriptions of Image Surfaces", PhD thesis, Department
% of Engineering Science, Oxford University 1989, p45.
%
%
% Author:
% Peter Kovesi
% Department of Computer Science & Software Engineering
% The University of Western Australia
% pk @ csse uwa edu au
% http://www. csse.uwa.edu.au/~pk
%
% March 2002    - original version
% December 2002 - updated comments

function [cim, r, c] = harris(im, sigma, thresh, radius, disp)

    error(nargchk(2,5,nargin));

    if ~isa(im,'double')
    im = double(im);
    end

    dx = [-1 0 1; -1 0 1; -1 0 1]; % Derivative masks
    dy = dx';

    Ix = conv2(im, dx, 'same');    % Image derivatives
    Iy = conv2(im, dy, 'same');

    % Generate Gaussian filter of size 6*sigma (+/- 3sigma) and of
    % minimum size 1x1.
    g = fspecial('gaussian',max(1,fix(6*sigma)), sigma);

    Ix2 = conv2(Ix.^2, g, 'same'); % Smoothed squared image derivatives
    Iy2 = conv2(Iy.^2, g, 'same');
    Ixy = conv2(Ix.*Iy, g, 'same');

    % Compute the Harris corner measure. Note that there are two measures
    % that can be calculated.  I prefer the first one below as given by
    % Nobel in her thesis (reference above).  The second one (commented out)
    % requires setting a parameter, it is commonly suggested that k=0.04 - I
    % find this a bit arbitrary and unsatisfactory.

    cim = (Ix2.*Iy2 - Ixy.^2)./(Ix2 + Iy2 + eps); % My preferred  measure.
%     k = 0.04;
%     cim = (Ix2.*Iy2 - Ixy.^2) - k*(Ix2 + Iy2).^2; % Original Harris measure.

    if nargin > 2   % We should perform nonmaximal suppression and threshold

    % Extract local maxima by performing a grey scale morphological
    % dilation and then finding points in the corner strength image that
    % match the dilated image and are also greater than the threshold.
    sze = 2*radius+1;                   % Size of mask.
    mx = ordfilt2(cim,sze^2,ones(sze)); % Grey-scale dilate.
    cim = (cim==mx)&(cim>thresh);       % Find maxima.

    [r,c] = find(cim);                  % Find row,col coords.

    if nargin==5 & disp      % overlay corners on original image
        imshow(im,1), hold on
        plot(c,r,'r+'), title('corners detected');
    end

    else  % leave cim as a corner strength image and make r and c empty.
    r = []; c = [];
    end
```

7.5.iv Show Keys

```matlab
% showkeys(image, locs)
%
% This function displays an image with SIFT keypoints overlayed.
%   Input parameters:
%     image: the file name for the image (grayscale)
%     locs: matrix in which each row gives a keypoint location (row,
%           column, scale, orientation)

function showkeys(image, locs)

disp('Drawing SIFT keypoints ...');

% Draw image with keypoints
figure('Position', [50 50 size(image,2) size(image,1)]);
colormap('gray');
imagesc(image);
hold on;
imsize = size(image);
for i = 1: size(locs,1)
    % Draw an arrow, each line transformed according to keypoint parameters.
    TransformLine(imsize, locs(i,:), 0.0, 0.0, 1.0, 0.0);
    TransformLine(imsize, locs(i,:), 0.85, 0.1, 1.0, 0.0);
    TransformLine(imsize, locs(i,:), 0.85, -0.1, 1.0, 0.0);
end
hold off;


% ------ Subroutine: TransformLine -------
% Draw the given line in the image, but first translate, rotate, and
% scale according to the keypoint parameters.
%
% Parameters:
%   Arrays:
%    imsize = [rows columns] of image
%    keypoint = [subpixel_row subpixel_column scale orientation]
%
%   Scalars:
%    x1, y1; begining of vector
%    x2, y2; ending of vector
function TransformLine(imsize, keypoint, x1, y1, x2, y2)

% The scaling of the unit length arrow is set to approximately the radius
%   of the region used to compute the keypoint descriptor.
len = 6 * keypoint(3);

% Rotate the keypoints by 'ori' = keypoint(4)
s = sin(keypoint(4));
c = cos(keypoint(4));

% Apply transform
r1 = keypoint(1) - len * (c * y1 + s * x1);
c1 = keypoint(2) + len * (- s * y1 + c * x1);
r2 = keypoint(1) - len * (c * y2 + s * x2);
c2 = keypoint(2) + len * (- s * y2 + c * x2);

line([c1 c2], [r1 r2], 'Color', 'c');
```

7.5.v Sift

```
% [image, descriptors, locs] = sift(imageFile)
%
% This function reads an image and returns its SIFT keypoints.
%   Input parameters:
%     imageFile: the file name for the image.
%
%   Returned:
%     image: the image array in double format
%     descriptors: a K-by-128 matrix, where each row gives an invariant
%         descriptor for one of the K keypoints.  The descriptor is a vector
%         of 128 values normalized to unit length.
%     locs: K-by-4 matrix, in which each row has the 4 values for a
%         keypoint location (row, column, scale, orientation).  The
%         orientation is in the range [-PI, PI] radians.
%
% Credits: Thanks for initial version of this program to D. Alvaro and
%          J.J. Guerrero, Universidad de Zaragoza (modified by D. Lowe)

function [image, descriptors, locs] = sift(imageFile)

% Load image
image = imread(imageFile);

% If you have the Image Processing Toolbox, you can uncomment the following
%   lines to allow input of color images, which will be converted to grayscale.
% if isrgb(image)
%    image = rgb2gray(image);
% end

[rows, cols] = size(image);

% Convert into PGM imagefile, readable by "keypoints" executable
f = fopen('tmp.pgm', 'w');
if f == -1
    error('Could not create file tmp.pgm.');
end
fprintf(f, 'P5\n%d\n%d\n255\n', cols, rows);
fwrite(f, image', 'uint8');
fclose(f);

% Call keypoints executable
if isunix
    command = '!./sift ';
else
    command = '!siftWin32 ';
end
command = [command ' <tmp.pgm >tmp.key'];
eval(command);

% Open tmp.key and check its header
g = fopen('tmp.key', 'r');
if g == -1
    error('Could not open file tmp.key.');
end
[header, count] = fscanf(g, '%d %d', [1 2]);
if count ~= 2
    error('Invalid keypoint file beginning.');
end
num = header(1);
len = header(2);
if len ~= 128
    error('Keypoint descriptor length invalid (should be 128).');
end

% Creates the two output matrices (use known size for efficiency)
locs = double(zeros(num, 4));
descriptors = double(zeros(num, 128));
```

```matlab

% Parse tmp.key
for i = 1:num
    [vector, count] = fscanf(g, '%f %f %f %f', [1 4]); %row col scale ori
    if count ~= 4
        error('Invalid keypoint file format');
    end
    locs(i, :) = vector(1, :);

    [descrip, count] = fscanf(g, '%d', [1 len]);
    if (count ~= 128)
        error('Invalid keypoint file value.');
    end
    % Normalize each input vector to unit length
    descrip = descrip / sqrt(sum(descrip.^2));
    descriptors(i, :) = descrip(1, :);
end
fclose(g);
```

7.5.vi Main Q5 Code

```matlab
close all
clear
clc

I1=imread('image.seq10.png'); %Read image file 1
I2=imread('image.seq11.png');%Read image file 2

BW1=rgb2gray(I1); %convert to gray scale
BW2=rgb2gray(I2);

%a.1 Sobel Edge detector
E_sobel_I1=edge(BW1,'sobel');
figure
imshow(E_sobel_I1);
title('Sobel Edge Detector');
%E_sobel_I2=edge(BW2,'sobel');

%a.2 Canny Edge Detector
E_canny_I1=edge(BW1,'canny',[0.1,0.4]);
figure
imshow(E_canny_I1);
title('Canny Edge Detector');
%E_canny_I2=edge(BW2,'canny',[0.1,0.4]); %E_canny_I2=edge(I2_gray,'canny',[0.05,0.25]);

%Harris Corners
I1_harris=corner(BW1,'harris');
figure
imshow(BW1);
hold on
plot(I1_harris(:,1),I1_harris(:,2),'r*');
title('Harris corners');

%c. Sift Corners for I1
imwrite(BW1,'BW1.png') %writing gray image of I1 as 'BW1.png'
[image, descrips, locs] = sift('BW1.png');
showkeys(image, locs);

%d. Matching Features using given match function
match('whitehouse.left.png','whitehouse.right.png');
```

7.6 Question 6

7.6.i Main Code

```
%Establish a connection to the motor arm, calculate centroids and
%orientations, and send commands to manoeuvre boxes
%%Code adapted from the code given as part of the assignment specification

function ass2q6_connect()

clear all;
close all;
clc;

%% SETUP CAMERA AND OBTAIN IMAGE
vid = videoinput('pointgrey',1, 'Mono8_640x480');
pause(1);
I = getsnapshot(vid);

%% INSERT IMAGE PROCESSING CODE HERE
%Should use image I, find cubes, and determine commands for arm
%
%
%
% I = imread('tt3.png');    %Can load an image from file instead of camera feed
imshow(I);
drawnow();
%
[CentroidsX, CentroidsY, OrientationAngle] = find_centroids_orientation_grey(I);
%
%


%% OPEN NETWORK CONNECTION
t=tcpip('192.168.0.1', 2020, 'NetworkRole', 'client');
fopen(t);


%% CONTROL ARM
%Should send out commands via TCPIP as strings (format below)

%command_string = '<x0,y260>\n'
%sendCommand(t,command_string)      %Send command string to robot via network

%Include brackets in command

%Full list of commands:
% <x0,y360>\n = PositionTool(0,360)
% <h0>\n = setToolHeight(0)
% <a90>\n = setToolAngle(90)
% <o>\n = OpenGripper
% <c>\n = CloseGripper

height = 0.1;

strings = zeros(11,8);
% t=1;
for i = 1:length(CentroidsX)

   sendCommand(t,'<h4>\n');
   sendCommand(t,'<o>\n');
   sendCommand(t,sprintf('<a%.0f>\n', OrientationAngle(i)));
   sendCommand(t,sprintf('<x%.0f,y%.0f>\n',CentroidsX(i),CentroidsY(i)));
   sendCommand(t,'<h0>\n');
   sendCommand(t,'<c>\n');
   sendCommand(t,'<h4>\n');
   sendCommand(t,'<x0,y490>\n');
   sendCommand(t,'<a0>\n');
   sendCommand(t, sprintf('<h%.1f>\n', height));
   height = height + 1;
   sendCommand(t,'<o>\n');

end

for i = 1:8
    for k = i:11
        sendCommand(t, strings(k,i));
    end
end
%Example code
% sendCommand(t,'<o>\n')         %Open gripper
% sendCommand(t,'<x0,y490>\n')  %Move to (0,490)
% sendCommand(t,'<c>\n')         %Close gripper
% sendCommand(t,'<x0,y390>\n')  %Move to (0,390)
% sendCommand(t,'<o>\n')         %Open gripper
% sendCommand(t,'<h2>\n')        %Set tool height to 2 cubes
% sendCommand(t,'<a45>\n')  %Set tool angle to 45 deg



% CLOSE NETWORK PORT AND CAMERA
delete(vid)
fclose(t);

end
```

7.6.ii Image Processing and Position Finding

```
function [CentroidsX, CentroidsY, OrientationAngle] = find_centroids_orientation_grey(I)

% I=imread('tt1.png'); % read image

I2=imcrop(I,[68.5 4.5 491 472]); %isolate the workingspace
I3=double(I2)/255;  %convert to double
I4=adapthisteq(I3); %enhance contrast using adaptive histogram equalization
I5=I4<0.10;%0.1 %0.11 % find boxes using a threshold

C1=bwareaopen(I5,1000); %remove objects less than 1000 pixels/ Remove unnessosary regions
se=strel('square',15);  %create a structuring element of size 15 square
C2=imclose(C1,se); %morphologicaly close image
C4=imfill(C2,'holes'); %fill holes

s=regionprops(C4,'centroid'); %find centroids of regions
centroids=cat(1,s.Centroid); %make a array using centroids in the structured array
imshow(C4)
hold on
plot(centroids(:,1),centroids(:,2),'b*') %plot centorids with boxes

%Assigns numbers to each centroid
numbers = 1:length(centroids);
strValues = strtrim(cellstr(num2str(numbers(:),'(%d)')));
text(centroids(:,1),centroids(:,2),strValues,'VerticalAlignment','bottom');

%GetOrientation
TR = 2;
LT = 8;

p = regionprops(C4, 'Extrema');

hold on


for i = 1:length(p)

%     TR to LT  -    Best so far
    sides1 = p(i).Extrema(TR,:) - p(i).Extrema(LT,:); % Returns the sides of the square triangle
        ↪ that completes the two chosen extrema:
    plot(p(i).Extrema(LT,1),p(i).Extrema(LT,2),'bx','MarkerSize',20);
    plot(p(i).Extrema(TR,1),p(i).Extrema(TR,2),'bx','MarkerSize',20);

    OrAn1(i) = rad2deg(atan(-sides1(2)/sides1(1)));  % Note the 'minus' sign compensates for the
        ↪ inverted y-values in image coordinates

end

hold off

%Find and remove fiducial boxes from centroids and orientation

x1=centroids(:,1);
y1=centroids(:,2);

for i=1:length(x1)
    if ((x1(i)< 60 & x1(i)>50)& (y1(i)<442 & y1(i)>432 ))
        B1=[x1(i),y1(i)];
    end
end

%Remove cordinates of fiducial boxes
OrAn1(B1(1,1)==x1)=[];
x1(B1(1,1)==x1)=[];
y1(B1(1,2)==y1)=[];

%find fiducial box_2 (270,220)
for i=1:length(x1)
```

```
    if ((x1(i)< 459 & x1(i)>249)& (y1(i)<431 & y1(i)>421 ))
        B2=[x1(i),y1(i)];
    end
end


OrAn1(B2(1,1)==x1)=[];
x1(B2(1,1)==x1)=[];
y1(B2(1,2)==y1)=[];

%find fiducial box_3 (-270,523)
for i=1:length(x1)
    if ((x1(i)< 53 & x1(i)>43)& (y1(i)<223 & y1(i)>213 ))
        B3=[x1(i),y1(i)];
    end
end

OrAn1(B3(1,1)==x1)=[];
x1(B3(1,1)==x1)=[];
y1(B3(1,2)==y1)=[];

%When only fiducial boxes present
TF=isempty(x1);
if TF==1
    warning('No objects detected other than fiducial boxes')
end

Yr=abs(p_dist_q6(B1,B2,x1,y1)); %distance from line going along centroids of box1 & box 2 (reference
    ↪ )
Xr=p_dist_q6(B1,B3,x1,y1);  %distance from line going along centroids of box1 & box 3 (reference)

%Find the reference point according to the world coordinates
    %X 792.8691 = 540 real , %Y 433.3300 = 303 real
Yreal=((Yr*303)/219.3306)+220;    %REAL Y COORDINATE
Xreal=((Xr*540)/399.4796)-270;    %REAL x COORDINATE

%Return X-Y coordinates of the centroid in real world coordinates, along
%with the orientation
CentroidsX = Xreal;
CentroidsY = Yreal;
OrientationAngle = OrAn1;

end
```

7.6.iii Perpendicular Distance

```
%Find the perpendicular distances between two boxes along their centroids

function [pDist] = p_dist_q6(B1,B2,u,v)

LP1=[B1(1,1),B1(1,2)];
LP2=[B2(1,1),B2(1,2)];

D=sqrt((LP2(1)-LP1(1))^2+(LP2(2)-LP1(2))^2);

r=(u*(LP1(2)-LP2(2))+v*(LP2(1)-LP1(1))+LP2(2)*LP1(1)-LP1(2)*LP2(1));

pDist=r/D;


end
```

7.6.iv Send Command

```matlab
%Send instructions to the Robotic Arm
%%Code as presented as part of the assignment specification

function sendCommand(t,command)

    %Send command to TCPIP port
    fprintf(t,command)

%     %Pause until a message is received
    while(~t.BytesAvailable)
    end
%     %Then flush the input buffer
    flushinput(t)
end
```