

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

1.b

1.c

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:

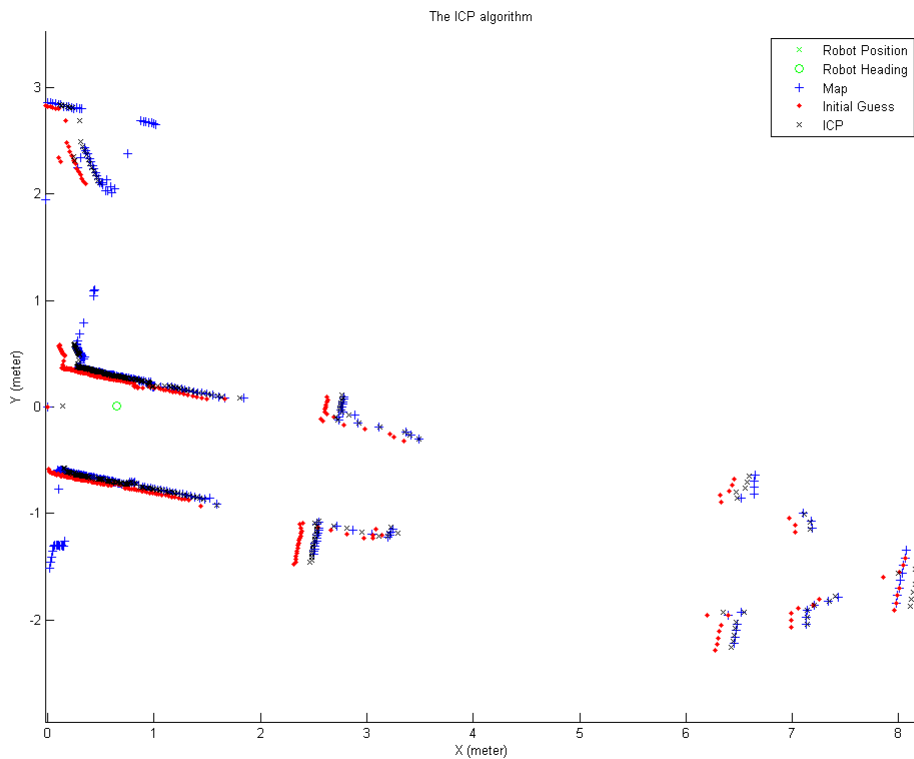


Figure 1: ICP estimate for maximum iterations of 40 and grid size of 0.005

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:

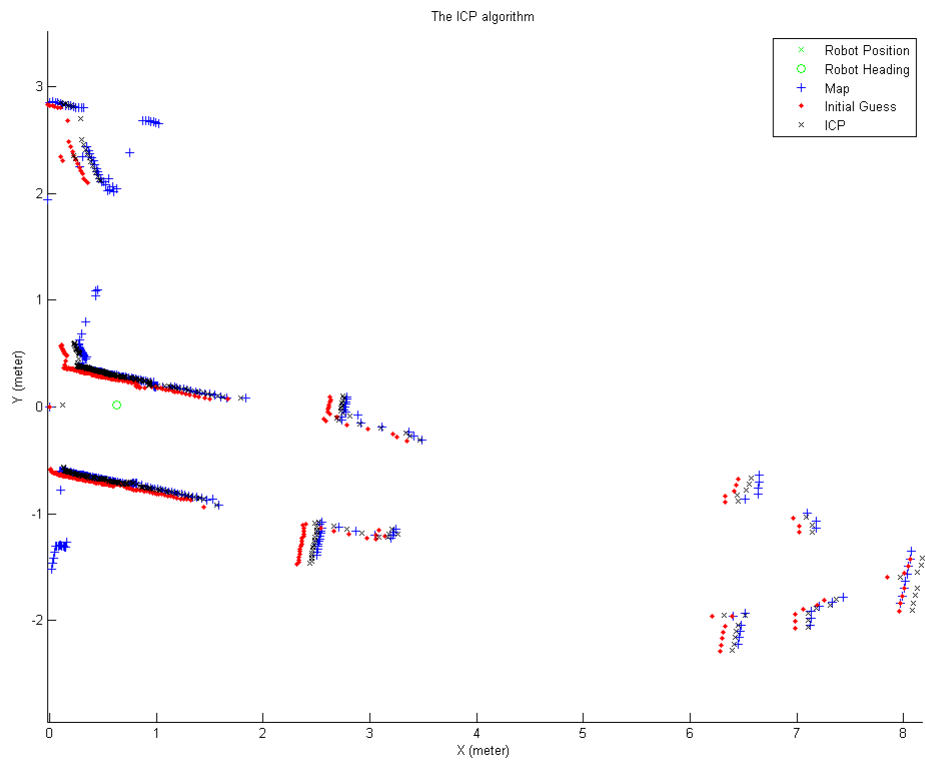


Figure 2: ICP estimate for maximum iterations of 40 and grid size of 0.001

Next, for a grid size of 0.1:

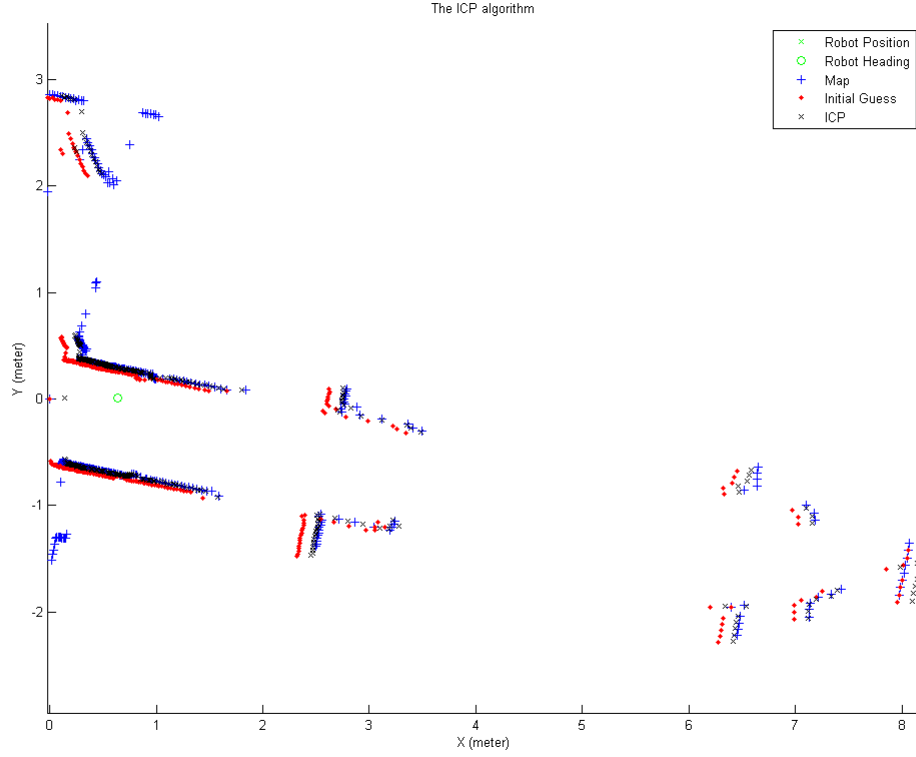


Figure 3: ICP estimate for maximum iterations of 40 and grid size of 0.005

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 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 $\Delta Pose_{bar}$, we can get an idea of the estimated heading of the vehicle, and by looking at $\Delta Pose_{bar_{cov}}$ the pose was as follows :

$\Delta Pose_{bar} = [0.1360, 0.0116, 0.0004]$ 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:

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

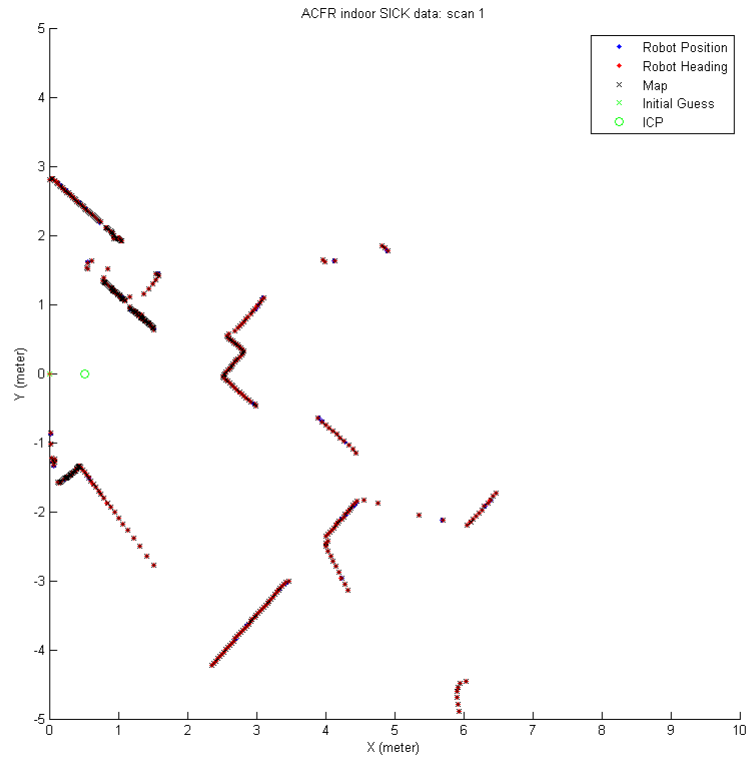
As can be seen, the covariance matrix is extremely close to zero. This is indicative of the factors involved being completely independent, though it does not confirm this. Again, seeing as this is run from a single frame, it is not an indicator for the overall relationship - we have used far too few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

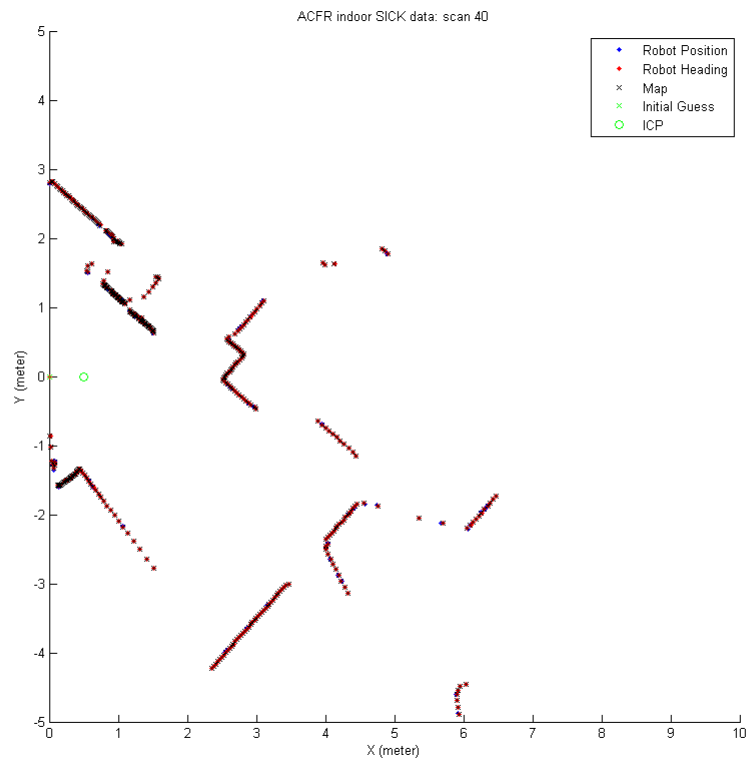
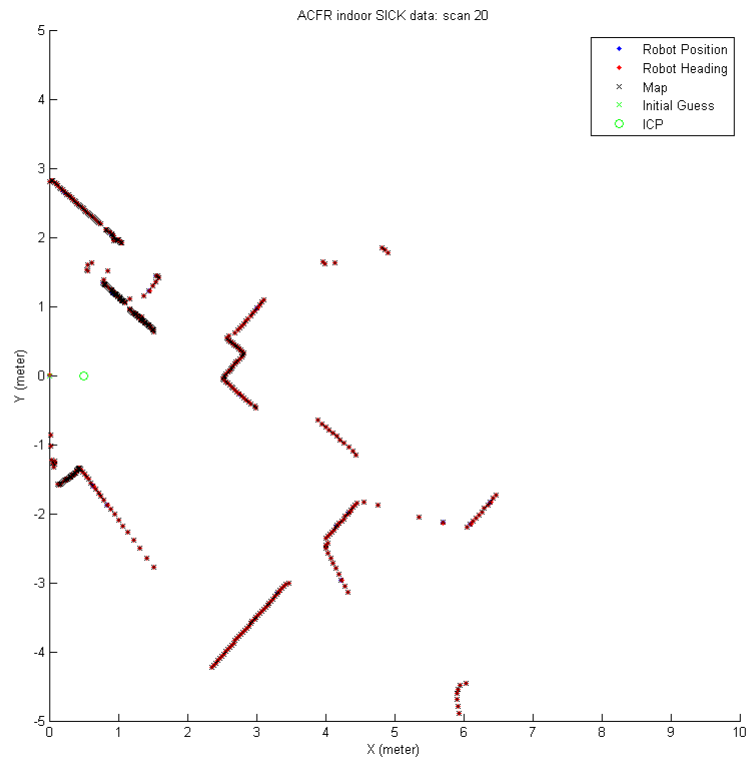
Problems with the Algorithm

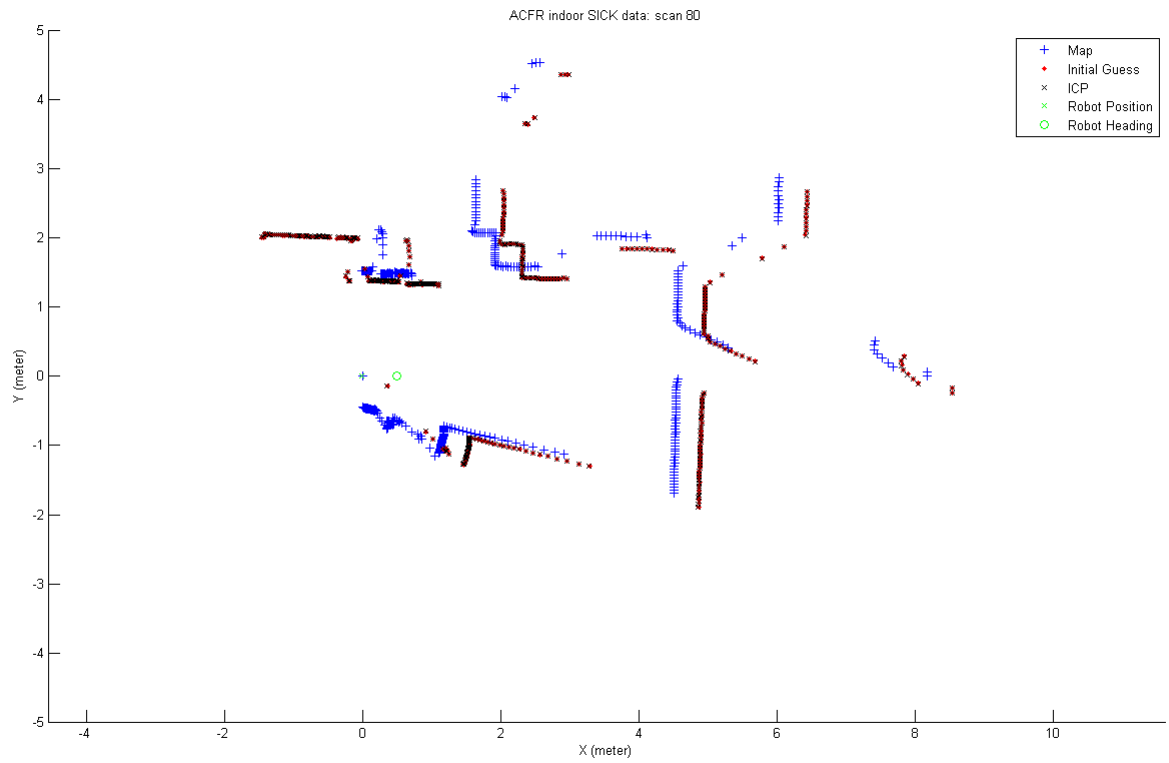
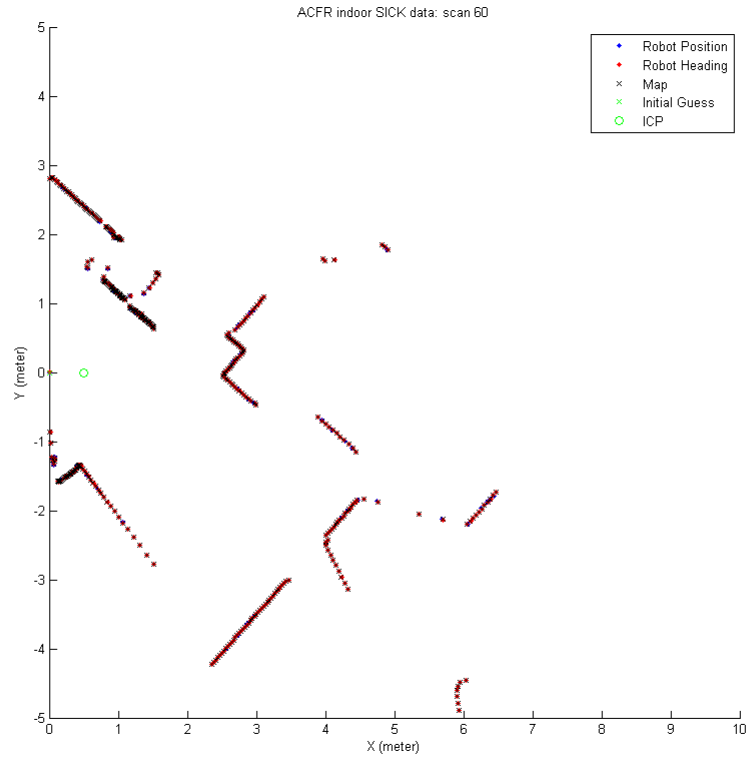
ICPv4.m takes little into account to do with angles/rotations, mostly using x and y values. This would mean that any rotational movement in the map would be taken poorly into account, as will be evident further on.

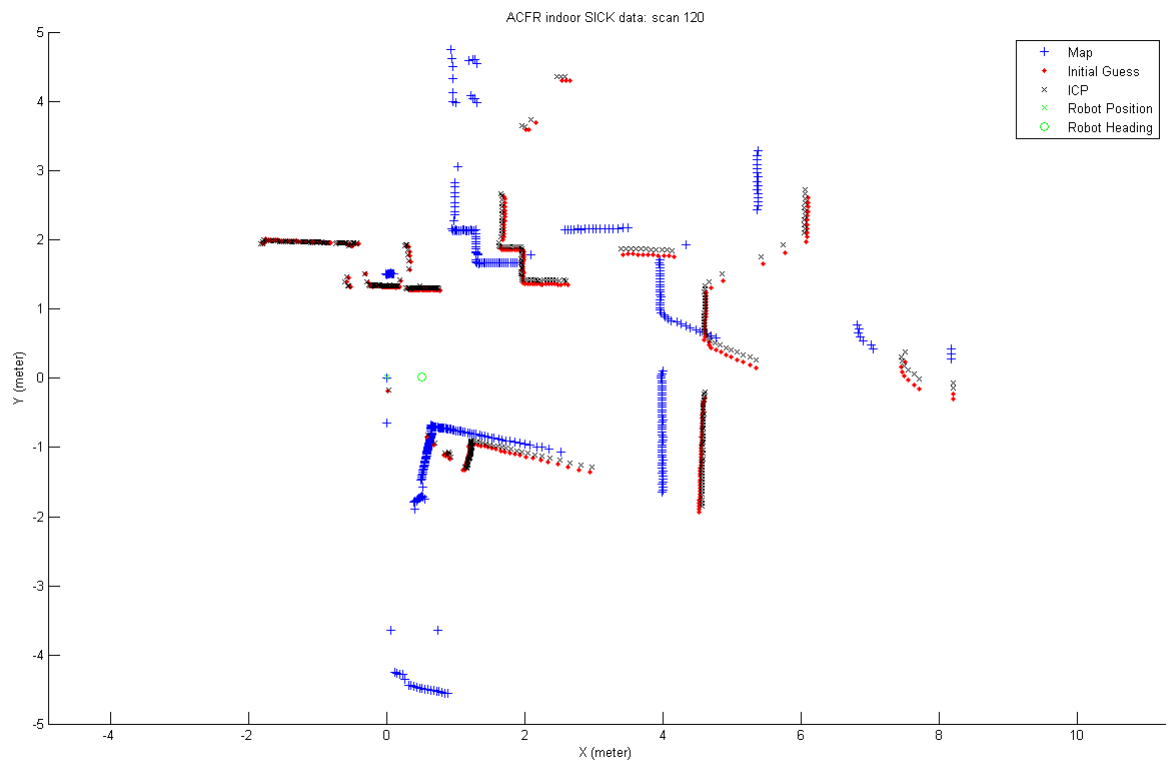
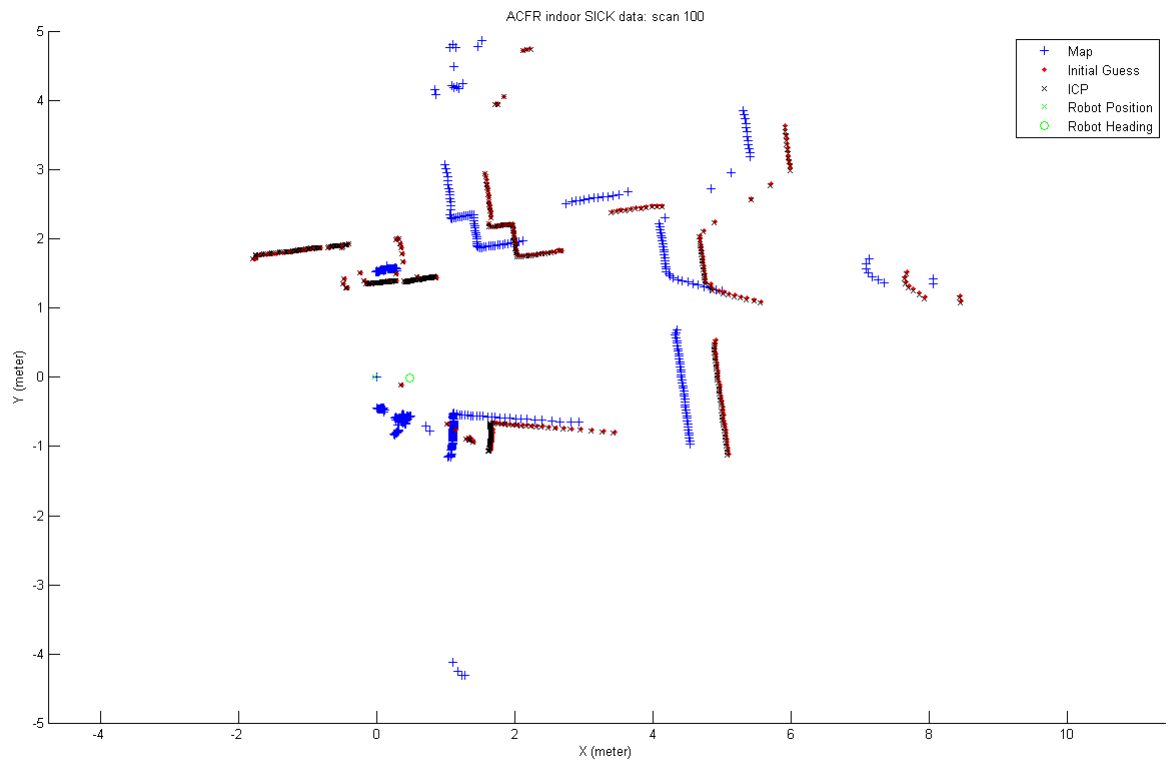
3..ii Part B

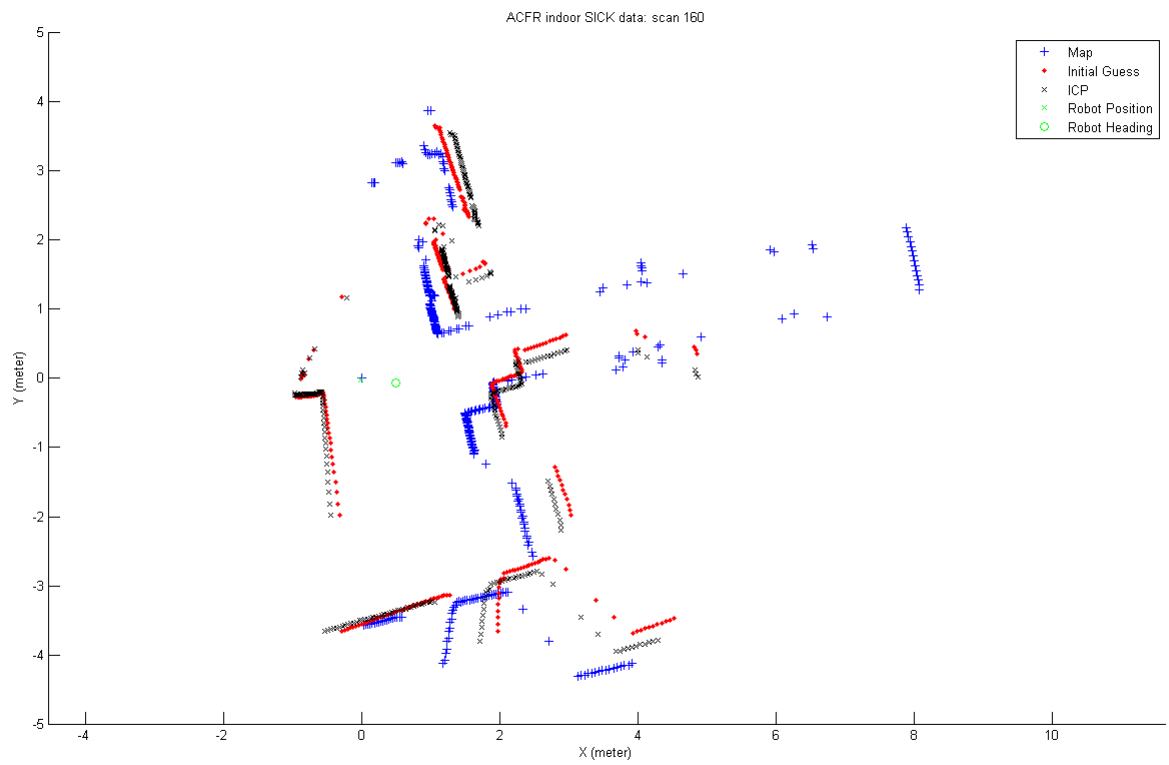
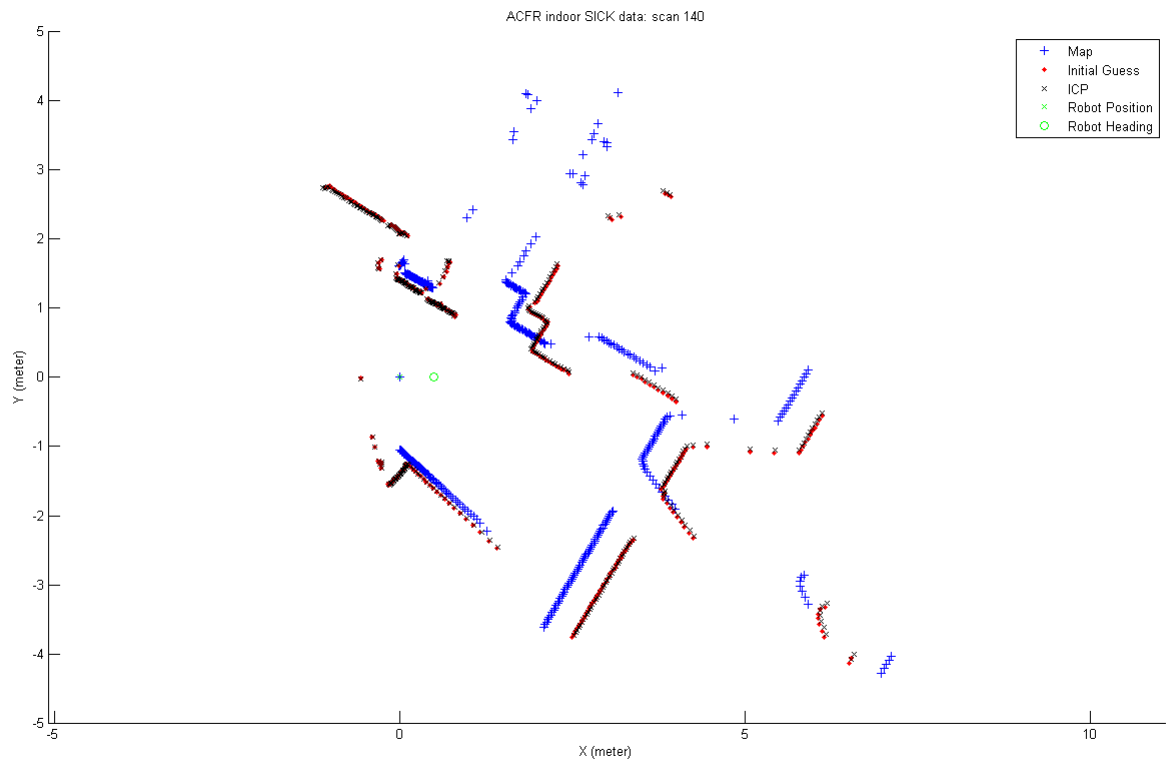
Incorporating the ICPv4.m algorithm into the laserShowACFR.m program enables us to build a real time picture of the movement of the vehicle and its perceived surroundings, relative to the actual mapped data. Observe:

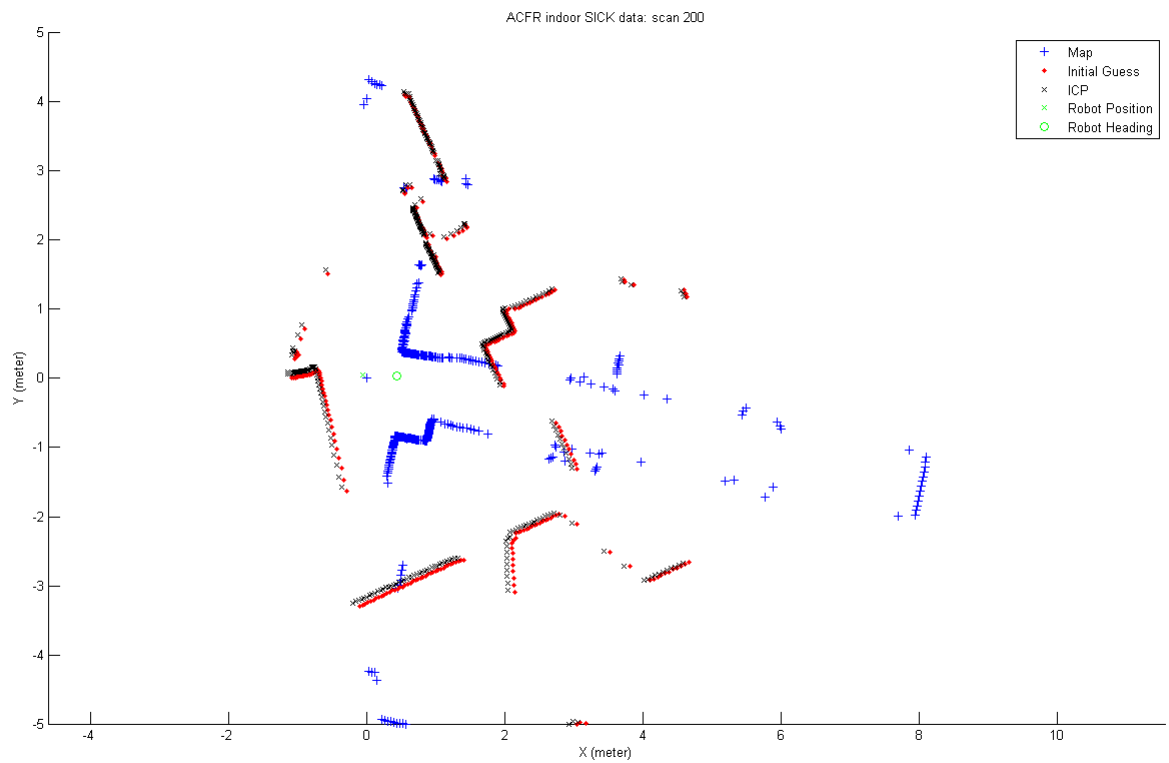
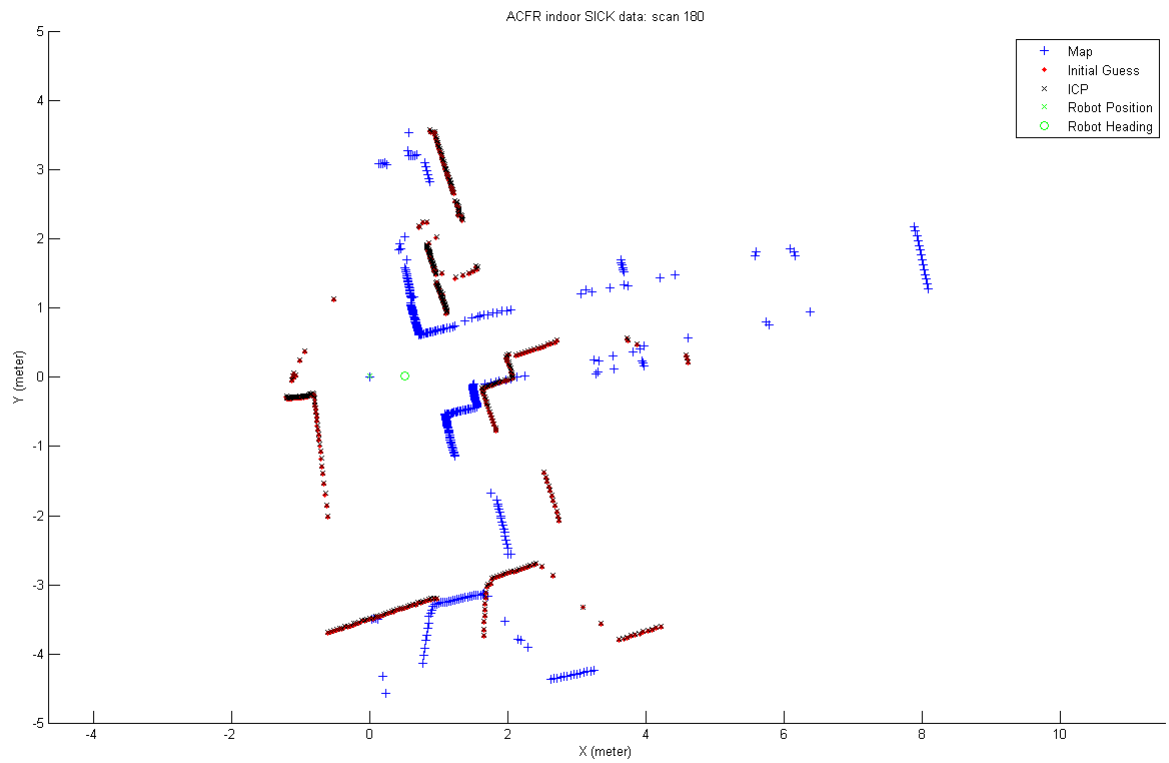


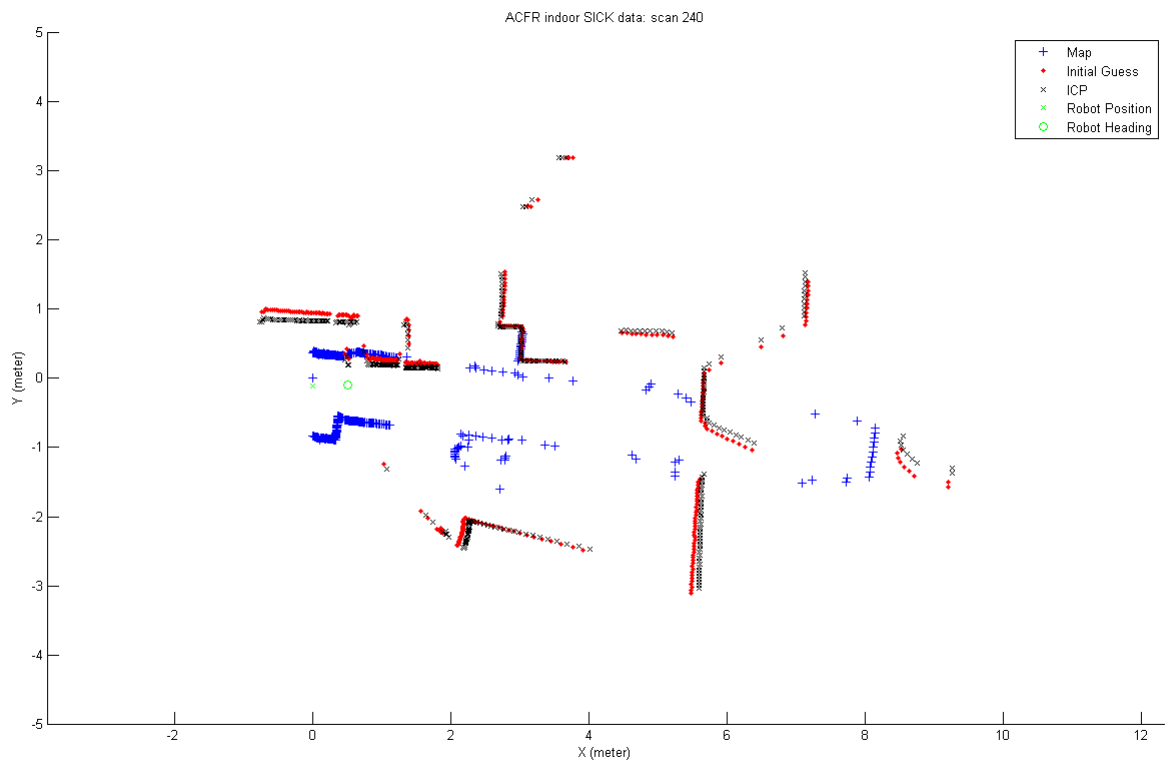
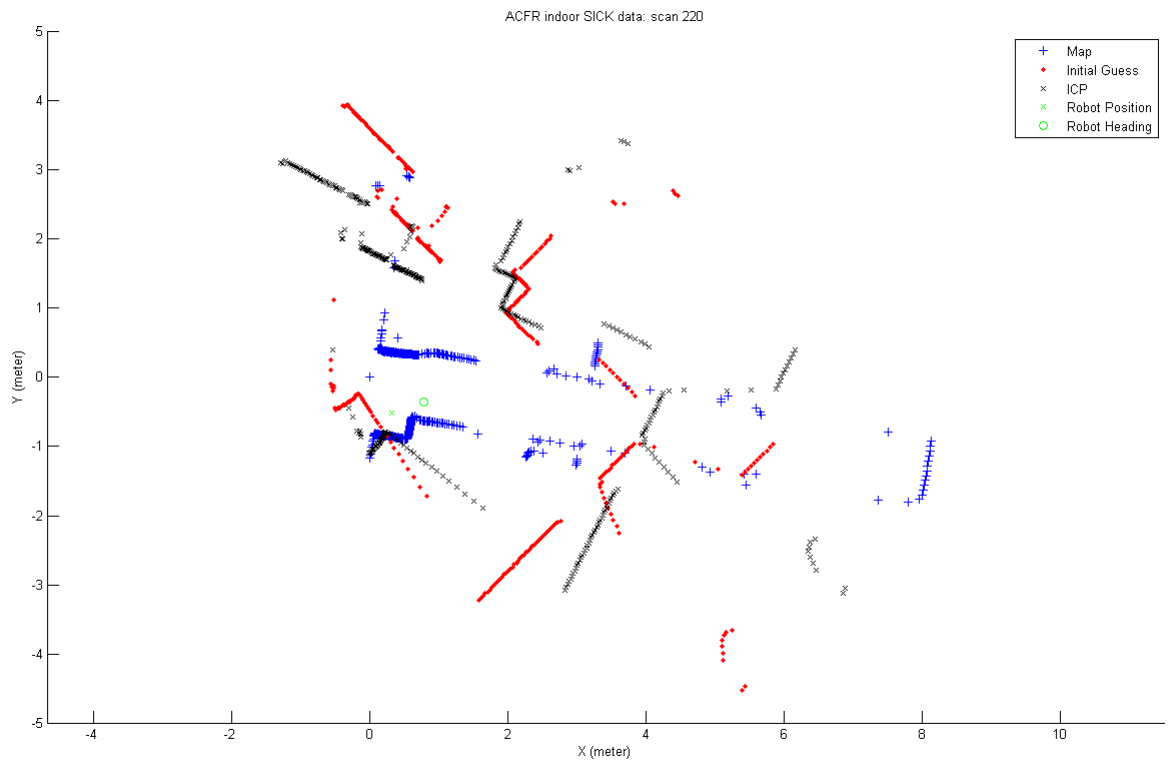


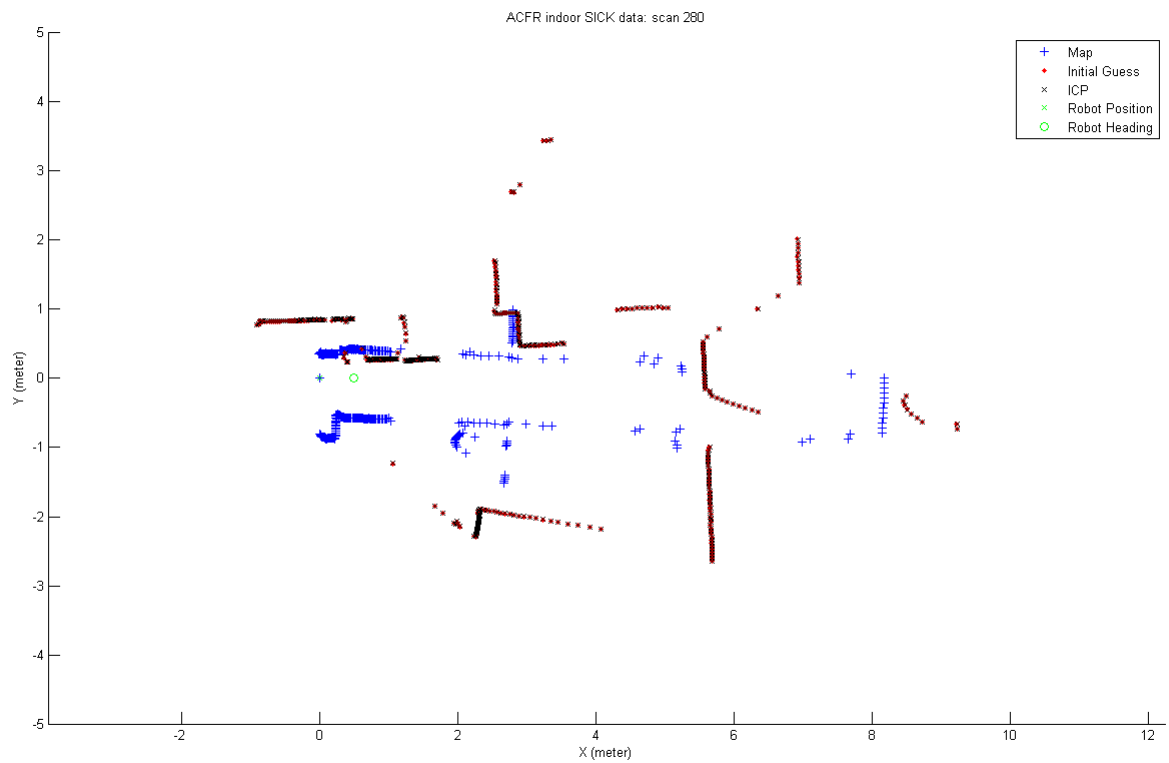
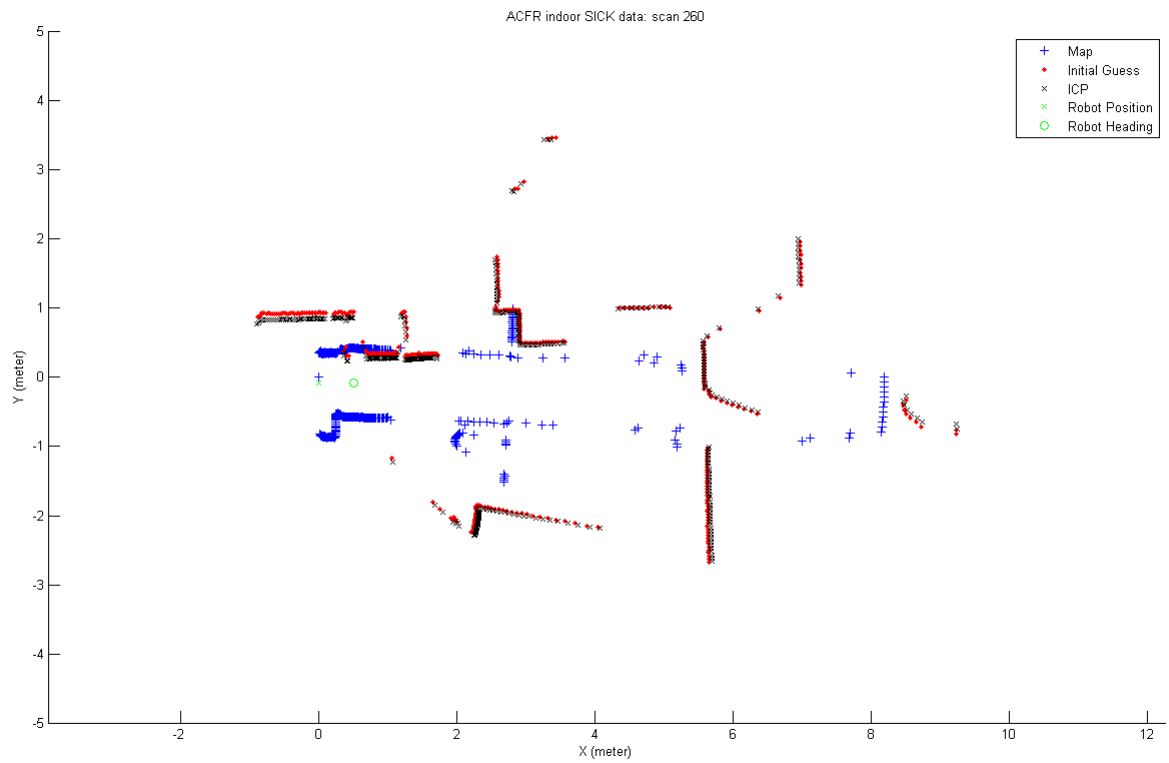


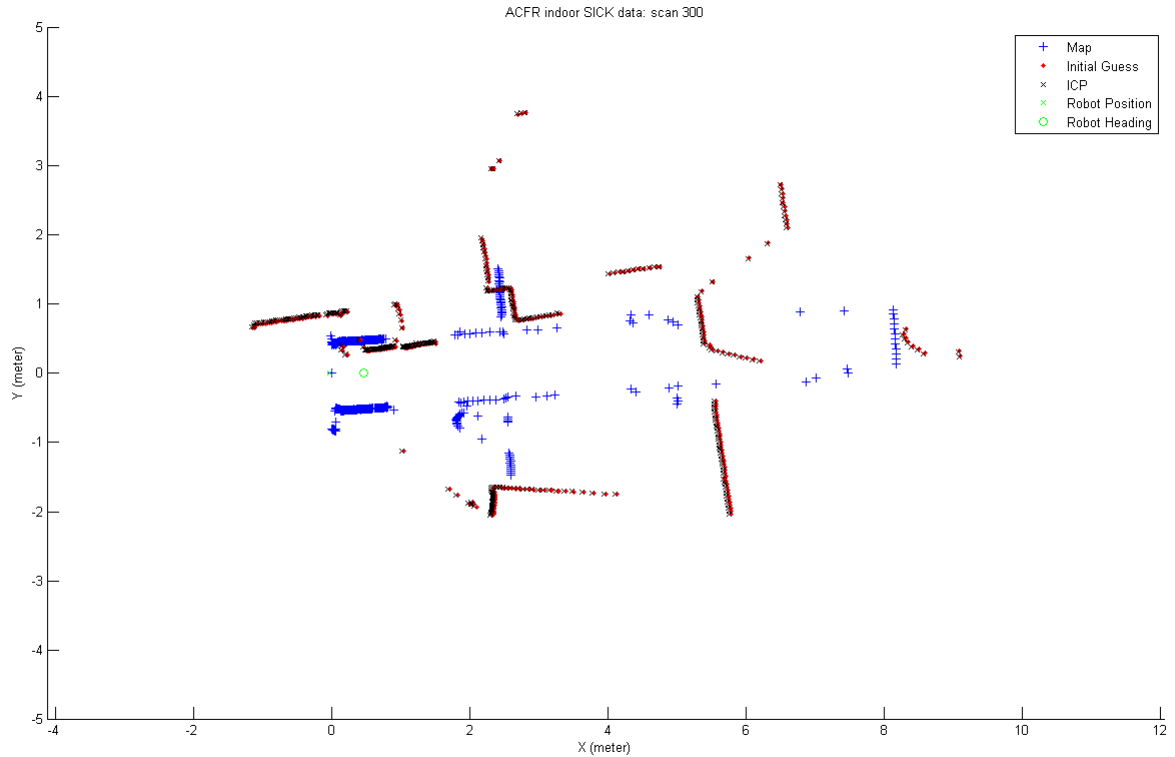












As can be seen, the ICP initially tracks the map data rather well, and the vehicle remains roughly aligned with the direction of motion. Around scan 200 however, everything begins to break down. At this point, the ICP has started lagging behind, and when the vehicle suddenly rotates it is unable to keep up. The closest points now no longer correspond to the original positions, and the ICP map becomes distorted, aligning with a new set of map data points. By scan 300 the alignment is well and 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 effective 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 were 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 second, 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

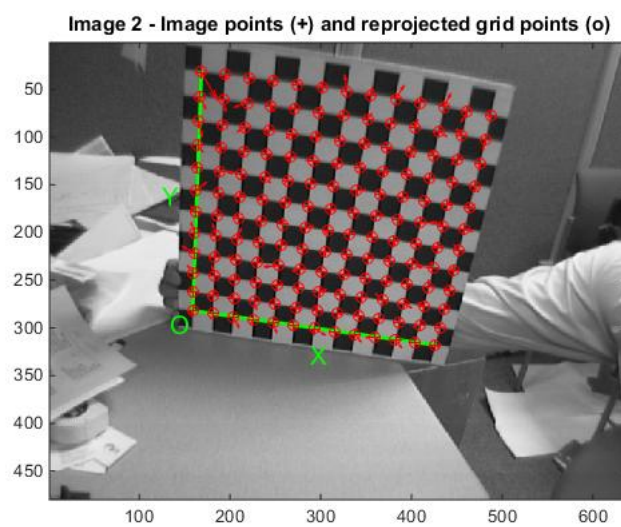
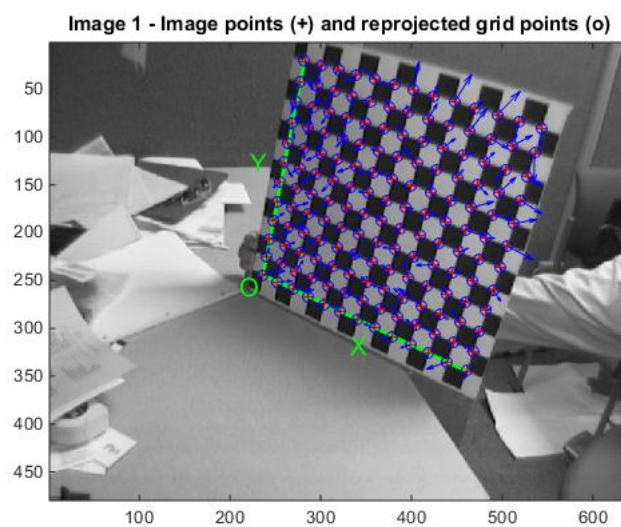
(Reference: <http://www.vision.caltech.edu/bouguetj/calibdoc/htmls/parameters.html>)

4.a Initial results after corner extraction

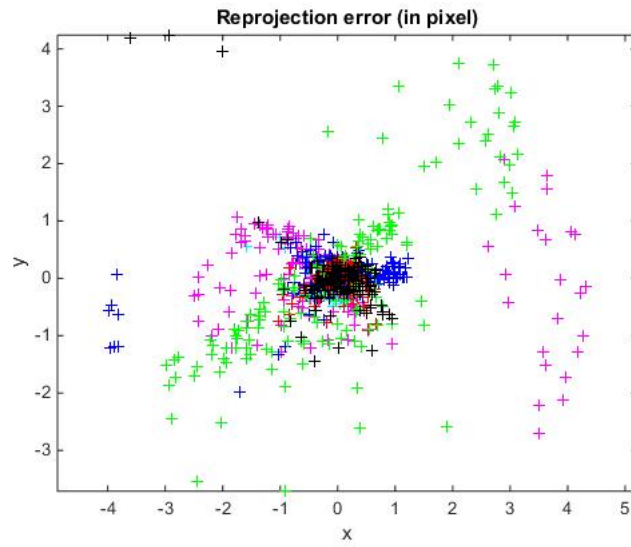
Following are the calibration results after initial calibration with a Pixel error of [0.81041,0.57619].

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

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

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

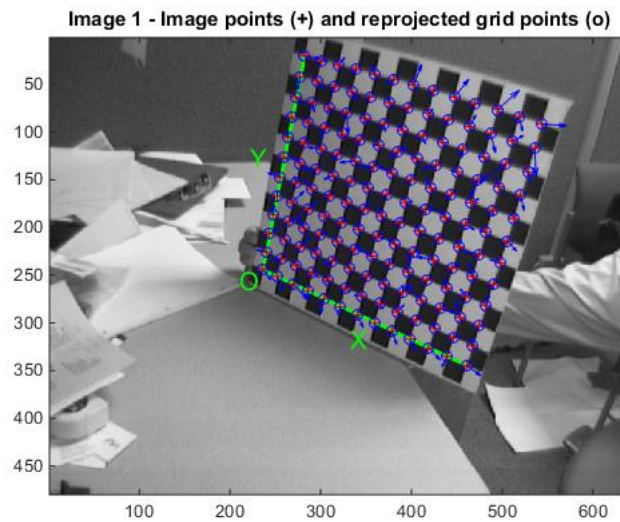
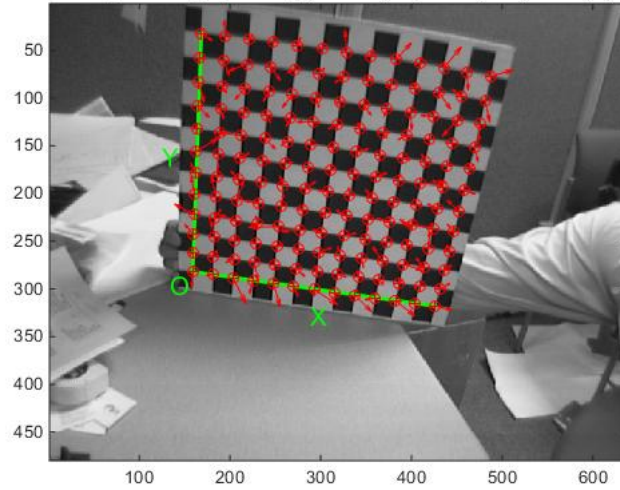
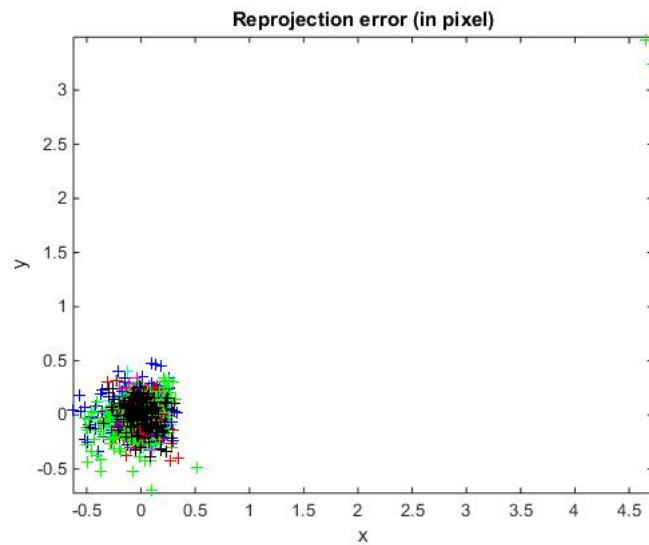


Image 2 - Image points (+) and reprojected grid points (o)



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.

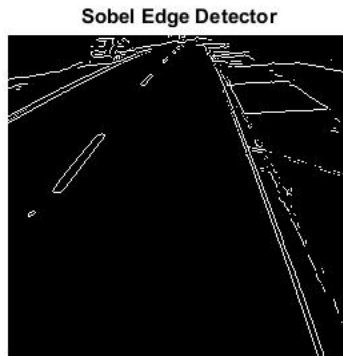


Figure 4: Sobel Edge Detector

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

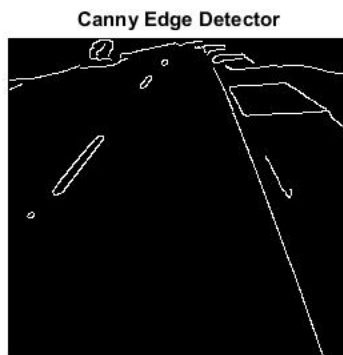


Figure 5: Canny Edge Detector

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.

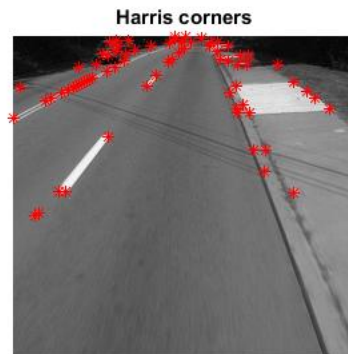


Figure 6: Harris Corners

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.

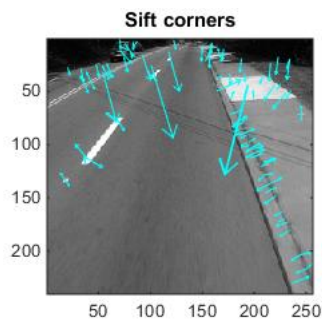


Figure 7: Sift Corners

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.

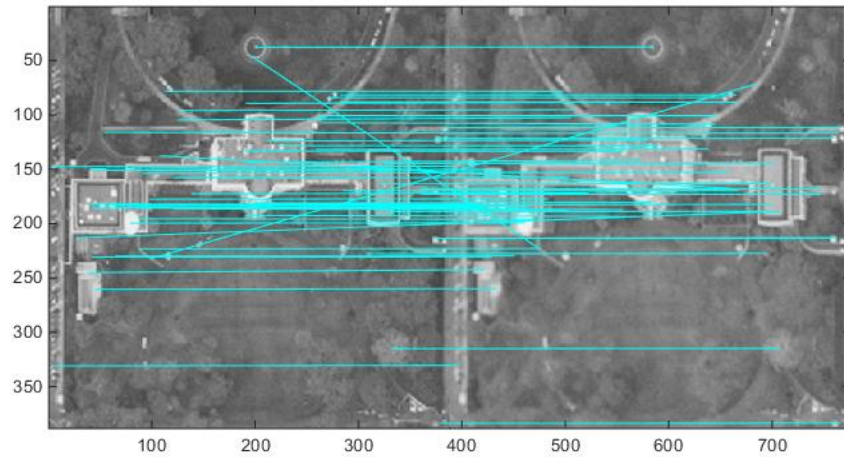


Figure 8: Feature Matching 1

The code we used 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:

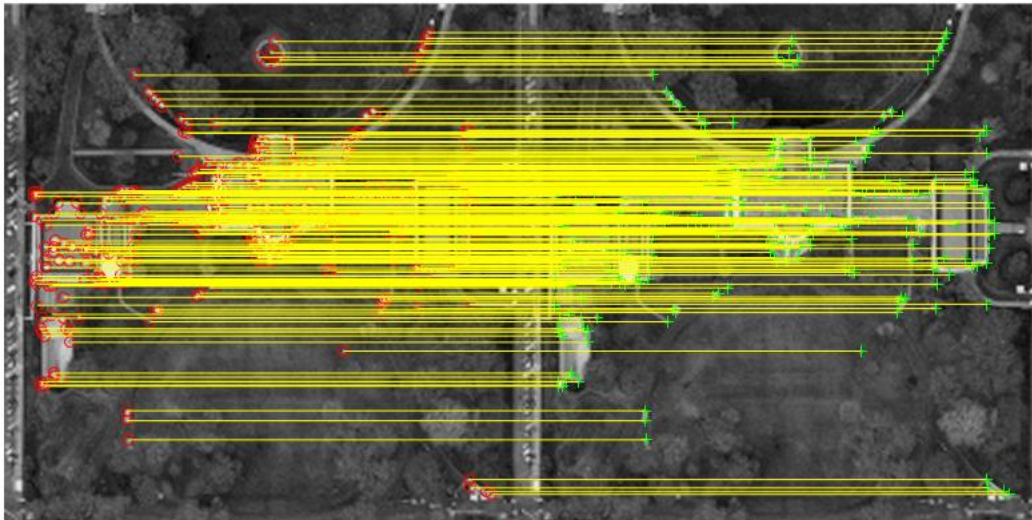


Figure 9: Feature Matching 2

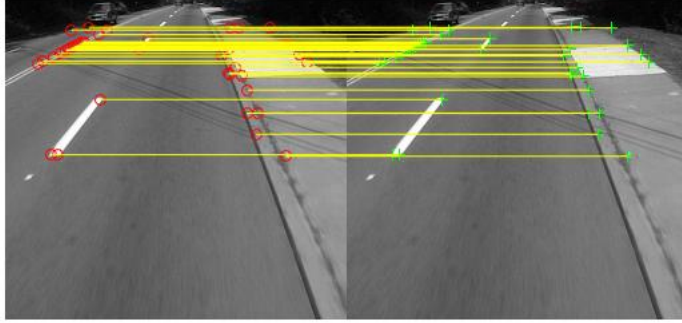


Figure 10: Feature Matching 3

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

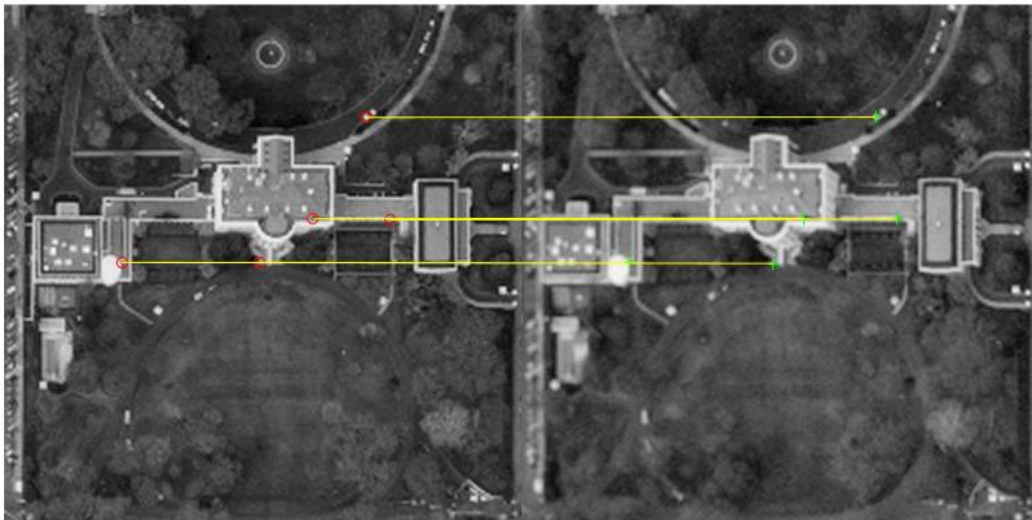


Figure 11: Different Matching 1

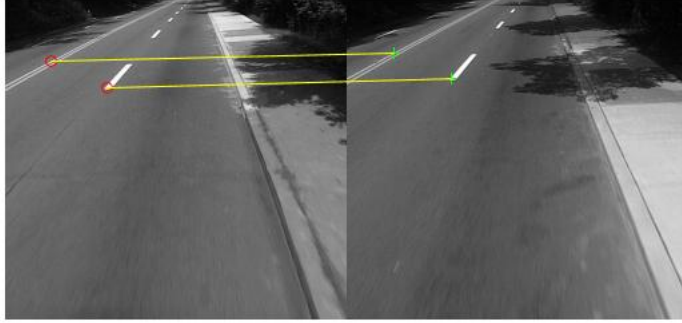


Figure 12: Different Matching 2

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 13: RegionProps Orientation

Due to the fact that the identified boxes were 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:

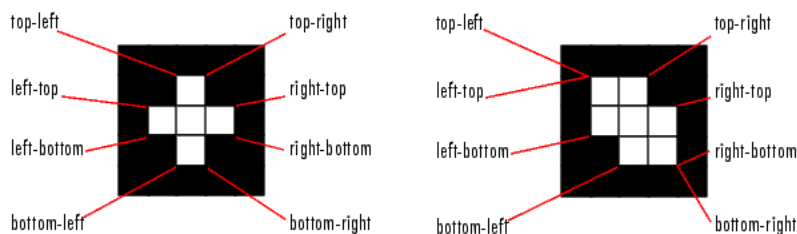


Figure 14: RegionProps Extrema

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.

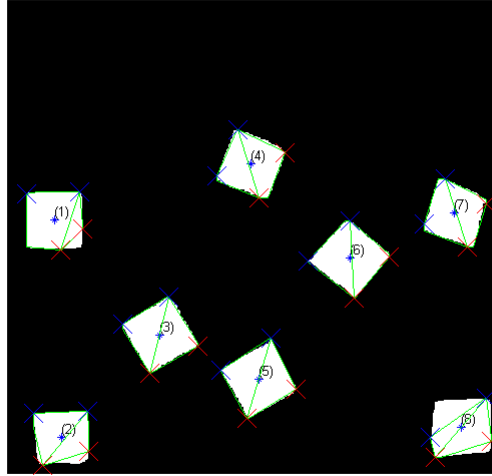


Figure 15: RegionProps Extrema in Action

This method proved simple, yet effective. Though it was not fully implemented, programming the system to 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. Extrema's 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

7.2 Question 2

7.3 Question 3

7.3.i showICP

```
1 % test the ICP algorithm
2 % Author: Chieh-Chih (Bob) Wang [bob.wang@cas.edu.au]
3 % Created: April 12, 2005.
4 % Last Modified: April 12, 2005.
5
6
7 clear
8 close all
9 clc
10
11 % load laser files
12 laser_scans=load('datasets\captureScanshornet.txt');
13 t0 = laser_scans(1,1);
14
15 % Scan A
16 i = 500;
17 xA = zeros(1);
18 yA = zeros(1);
19 for j = 2:size(laser_scans,2) %Map
20     range = laser_scans(i,j) / 1000;
21     bearing = ((j-1)/2 - 90)*pi/180;
22     if (range < 75)
23         xA = [xA range*cos(bearing)];
24         yA = [yA range*sin(bearing)];
25     end
26 end
27
28 % Scan B
29 i = 520;
30 xB = zeros(1);
31 yB = zeros(1);
32 for j = 2:size(laser_scans,2) %Initial Guess (source? stay 'constant'? trasformed by ICP)
33     range = laser_scans(i,j) / 1000;
34     bearing = ((j-1)/2 - 90)*pi/180;
35     if (range < 75)
36         xB = [xB range*cos(bearing)];
37         yB = [yB range*sin(bearing)];
38     end
39 end
40
41 deltaPose = zeros(3,1);
42
43 [deltaPose_bar, deltaPose_bar_Cov, N] = ICPv4(deltaPose, [xA;yA], [xB;yB]); %ICP
44
45 newB = head2tail_no_theta(deltaPose_bar, [xB;yB]);
46 new_xB = newB(1,:);
47 new_yB = newB(2,:);
48
49 figure
50 clf
51 hold on
52 plot(xA,yA,'b+')
53 plot(xB,yB,'r.')
54 plot(new_xB,new_yB,'kx')
55 axis equal
56 legend('Map','initial guess','ICP')
57 xlabel('X (meter)')
58 ylabel('Y (meter)')
59 title('The ICP algorithm')
```

7.3.ii Modified showICP

```
1 % test the ICP algorithm
2 % Author: Chieh-Chih (Bob) Wang [bob.wang@cas.edu.au]
3 % Created: April 12, 2005.
4 % Last Modified: April 12, 2005.
5 %
6 %Modified by James Ferris to Show ICP data in a plot
7
8
9 clear
10 close all
11 clc
12
13 % load laser files
14 laser_scans=load('datasets\captureScanshornet.txt');
15 t0 = laser_scans(1,1);
16
17 % Scan A
18 i = 500;
19 xA = zeros(1);
20 yA = zeros(1);
21 for j = 2:size(laser_scans,2) %Map
22     range = laser_scans(i,j) / 1000;
23     bearing = ((j-1)/2 - 90)*pi/180;
24     if (range < 75)
25         xA = [xA range*cos(bearing)];
26         yA = [yA range*sin(bearing)];
27     end
28 end
29
30 % Scan B
31 i = 520;
32 xB = zeros(1);
33 yB = zeros(1);
34 for j = 2:size(laser_scans,2) %Initial Guess (source? stay 'constant'? trasformed by ICP)
35     range = laser_scans(i,j) / 1000;
36     bearing = ((j-1)/2 - 90)*pi/180;
37     if (range < 75)
38         xB = [xB range*cos(bearing)];
39         yB = [yB range*sin(bearing)];
40     end
41 end
42
43 deltaPose = zeros(3,1);
44
45 [deltaPose_bar, deltaPose_bar_Cov, N] = ICPv4(deltaPose, [xA;yA], [xB;yB]); %ICP
46
47 %%
48 %Added to show robot position and orientation
49 figure
50 clf
51 hold on
52 Pose = deltaPose_bar;
53 h = 0.5;
54 Pose2 = [Pose(1)+h*cos(Pose(3)), Pose(2)+h*sin(Pose(3))];
55 plot(Pose(1),Pose(2), 'gx');
56 plot(Pose2(1),Pose2(2), 'go');
57 %%
58
59 newB = head2tail_no_theta(deltaPose_bar, [xB;yB]);
60 new_xB = newB(1,:);
61 new_yB = newB(2,:);
62
63 plot(xA,yA, 'b+')
64 plot(xB,yB, 'r.')
65 plot(new_xB,new_yB, 'kx')
66 axis equal
67 legend('Robot Position', 'Robot Heading', 'Map','Initial Guess','ICP')
```

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

7.3.iii laserShowAcfr

```
1 %
2 % Author: Stefan Williams (stefanw@acfr.usyd.edu.au)
3 %
4 %
5
6 clear
7 close all
8 clc
9
10 % load laser files
11 laser_scans=load('datasets\captureScanshornet.txt');
12
13 t0 = laser_scans(1,1);
14
15 figure
16 for i = 1:length(laser_scans)
17     t_laser = laser_scans(i,1) - t0;
18     xpoint = zeros(1);
19     ypoint = zeros(1);
20     for j = 2:size(laser_scans,2)
21         range = laser_scans(i,j) / 1000;
22         bearing = ((j-1)/2 - 90)*pi/180;
23         if (range < 75)
24             xpoint = [xpoint range*cos(bearing)];
25             ypoint = [ypoint range*sin(bearing)];
26         end
27     end
28     plot(xpoint(:), ypoint(:), '.');
29     axis equal;
30     axis([0 10 -5 5]);
31     xlabel('X (meter)')
32     ylabel('Y (meter)')
33     title(sprintf('ACFR indoor SICK data: scan %d',i))
34     drawnow
35
36     %     if i == 500
37     %         pause;
38     %     elseif i == 520
39     %         pause;
40     %     end
41
42 end
```

7.3.iv Modified laserShowACFR

```
1 %
2 % Author: Stefan Williams (stefanw@acfr.usyd.edu.au)
3 %
4 %
5 %Modified by James Ferris to include ICP data in a plot
6
7 clear
8 close all
9 clc
10
11 % load laser files
12 laser_scans=load('datasets\captureScanshornet.txt');
13
14 t0 = laser_scans(1,1);
15
16 %%
17 %Initialise ICP data
18 i = 20; %Let the 'initial guess' be starting point + 20, as in part A
19 xB = zeros(1);
20 yB = zeros(1);
21 for j = 2:size(laser_scans,2)
22     range = laser_scans(i,j) / 1000;
23     bearing = ((j-1)/2 - 90)*pi/180;
24     if (range < 75)
25         xB = [xB range*cos(bearing)];
26         yB = [yB range*sin(bearing)];
27     end
28 end
29 deltaPose = zeros(3,1);
30
31 %%
32
33 figure
34 for i = 1:length(laser_scans)
35
36     tlaser = laser_scans(i,1) - t0;
37     xpoint = zeros(1);
38     ypoint = zeros(1);
39     for j = 2:size(laser_scans,2)
40         range = laser_scans(i,j) / 1000;
41         bearing = ((j-1)/2 - 90)*pi/180;
42         if (range < 75)
43             xpoint = [xpoint range*cos(bearing)];
44             ypoint = [ypoint range*sin(bearing)];
45         end
46     end
47
48     %%
49     %Calculate ICP
50     [deltaPose_bar, deltaPose_bar_Cov, N] = ICPv4(deltaPose, [xpoint;ypoint], [xB;yB]);
51     newB = head2tail_no_theta(deltaPose_bar, [xB;yB]);
52     new_xB = newB(1,:);
53     new_yB = newB(2,:);
54
55     if i == 1 || mod(i,20) == 0
56
57         Pose = deltaPose_bar;
58         h = 0.5;
59         Pose2 = [Pose(1)+h*cos(Pose(3)), Pose(2)+h*sin(Pose(3))];
60         hold on
61
62
63         plot(xpoint(:), ypoint(:), '.');
64
65         plot(xB,yB,'r.')
66         plot(new_xB,new_yB,'kx')
67
68     end
69 end
```

```
68     plot(Pose(1),Pose(2),'gx');
69     plot(Pose2(1),Pose2(2),'go');
70
71
72     %%
73
74
75     axis equal;
76
77     legend('Robot Position', 'Robot Heading', 'Map','Initial Guess','ICP')
78     axis([0 10 -5 5]);
79     xlabel('X (meter)')
80     ylabel('Y (meter)')
81     title(sprintf('ACFR indoor SICK data: scan %d',i))
82     drawnow
83
84     pause
85 end
86
87     xB = new.xB;
88     yB = new.yB;
89
90     clf
91
92 end
```

7.3.v ICPv4

```
1 function [deltaPose_bar, deltaPose_bar_Cov, N] = ICPv4(deltaPose, a, b)
2
3 % function: ICP algorithm version 3.0
4 % a: point set, 2 x Na
5 % b: point set, 2 x Nb
6 % Xab: the relative tranformation between a and b
7 %
8 %
9 %
10 % Author: Chieh-Chih (Bob) Wang [bobwang@cs.cmu.edu]
11 % Created: Dec. 2, 2002.
12 % Last Modified: Dec. 27, 2003.
13 % (C) 2002-2004 Chieh-Chih (Bob) Wang. All Rights Reserved.
14
15
16 max_delta_g = 0.01;
17 max_iter = 40;
18 WinSize = 80;
19
20 % Cell 5cm x 5cm
21 grid_size = 0.005; % Changing this value may affect the accuracy of the ICP algorithm.
22 Map_x_min = min(a(1,:));
23 Map_y_min = min(a(2,:));
24 Map_x_max = max(a(1,:));
25 Map_y_max = max(a(2,:));
26
27 % Step 1: Create a grid map foo speed up correspondence search
28 GridMap_X = ceil((Map_x_max - Map_x_min)/grid_size);
29 GridMap_Y = ceil((Map_y_max - Map_y_min)/grid_size);
30 GridMap = zeros(GridMap_X, GridMap_Y);
31
32 for k=1:size(a,2)
33     [Map_i,Map_j] = XY2IJ(a(1,k), a(2,k), grid_size, Map_x_min, Map_y_min);
34     if Map_i>0 & Map_i<= GridMap_X & Map_j>0 & Map_j<= GridMap_Y
35         if GridMap(Map_i,Map_j)~= 0
36             %disp(sprintf('points collide %d,%d', Map_i, Map_j))
37         end
38         GridMap(Map_i,Map_j) = k;
39     end
40 end
41
42 WinSize_org = WinSize;
43 delta_g = 1000000000;
44 j=0;
45 g = deltaPose;
46
47 % method 2:
48 Z = [];
49 M = [];
50
51 NoMatch_flag = 0;
52
53 while (j < max_iter) & (delta_g > max_delta_g)
54     j = j+1;
55     old_g = g;
56     % Step 1:
57
58     % Finding Correspondence
59     Match_Pairs = [];
60     %New_Scan1_Index = Scan1_Index;
61     %New_Scan2_Index = Scan2_Index;
62
63     %WinSize = round(WinSize_org/j);
64     WinSize = WinSize_org- 4*j;
65     if WinSize < 2
66         WinSize = 2;
67     end
```



```

68
69 for k=1:size(b,2)
70
71     Point_X = head2tail_no_theta(g,b(:,k));
72
73     [Map_i,Map_j] = XY2IJ(Point_X(1), Point_X(2), grid_size, Map_x_min, Map_y_min);
74     % Search ...
75     % Define search area
76     % WinSize = 10;
77     Win_i_min = Map_i - WinSize;
78     if Win_i_min < 1
79         Win_i_min = 1;
80     end
81     Win_i_max = Map_i + WinSize;
82     if Win_i_max > GridMap_X
83         Win_i_max = GridMap_X;
84     end
85     Win_j_min = Map_j - WinSize;
86     if Win_j_min < 1
87         Win_j_min = 1;
88     end
89     Win_j_max = Map_j + WinSize;
90     if Win_j_max > GridMap_Y
91         Win_j_max = GridMap_Y;
92     end
93     [Search_i, Search_j] = find(GridMap(Win_i_min:Win_i_max, Win_j_min:Win_j_max) > 0);
94     if size(Search_i,1)>0
95         min_dis = 1000000000000000;
96         match_index = 0;
97         for m=1:size(Search_i,1)
98             a_index = GridMap(Search_i(m)+Win_i_min-1, Search_j(m)+Win_j_min-1);
99             dis = sqrt((Point_X(1) - a(1,a_index))^2 ...
100                 + (Point_X(2) - a(2,a_index))^2);
101             if (dis < min_dis)
102                 min_dis = dis;
103                 match_index = a_index;
104             end
105         end
106         % method 1:
107         Match_Pairs = [Match_Pairs; ...
108             b(1,k) b(2,k) a(1,match_index) a(2,match_index)];
109         % method 2:
110         Mk = [1 0 -Point_X(2,1); 0 1 Point_X(1,1)];
111         M = [M; Mk];
112         Z = [Z; Point_X - a(:,match_index)];
113
114         %MatchedPoints(1,k) = 1;
115         %New_Scan1_Index(1, match_index) = 2; % see Readme.txt for the definition
116         %New_Scan2_Index(1, k) = 2;
117     end
118 end
119 %Method 1: the closed form solution without covariance estimate
120 N = size(Match_Pairs,1);
121 %disp(sprintf('Inside ICPv4: iter %d, match pairs %d',j, N));
122 if N == 0
123     NoMatch_flag = 1;
124     break
125 end
126
127 X2_bar = sum(Match_Pairs(:,1))/N;
128 Y2_bar = sum(Match_Pairs(:,2))/N;
129 X1_bar = sum(Match_Pairs(:,3))/N;
130 Y1_bar = sum(Match_Pairs(:,4))/N;
131 Sx2x1 = sum((Match_Pairs(:,1) - X2_bar).*(Match_Pairs(:,3) - X1_bar));
132 Sy2y1 = sum((Match_Pairs(:,2) - Y2_bar).*(Match_Pairs(:,4) - Y1_bar));
133 Sx2y1 = sum((Match_Pairs(:,1) - X2_bar).*(Match_Pairs(:,4) - Y1_bar));
134 Sy2x1 = sum((Match_Pairs(:,2) - Y2_bar).*(Match_Pairs(:,3) - X1_bar));
135
136 g(3,1) = atan2(Sx2y1-Sy2x1, Sx2x1+ Sy2y1);
137 g(1,1) = X1_bar - (X2_bar*cos(g(3,1))- Y2_bar*sin(g(3,1)));
138 g(2,1) = Y1_bar - (X2_bar*sin(g(3,1))+ Y2_bar*cos(g(3,1)));

```

```

139
140     delta_g = sqrt((old_g(1) - g(1))^2 + (old_g(2) - g(2))^2);
141
142     %Method 2:
143     %     InvMM = inv(M'*M);
144     %     D_bar = InvMM*M'*Z;
145     %     ZminusMD_bar = Z-M*D_bar;
146     %     s_square = ZminusMD_bar'*ZminusMD_bar/(2*N-3);
147     %     Cov_ICP = s_square*InvMM;
148
149     %pause
150 end
151 % Xab_bar = g+D_bar;
152 % Xab_Cov = Cov_ICP;
153 if NoMatch_flag == 0
154     InvMM = inv(M'*M);
155     D_bar = InvMM*M'*Z;
156     ZminusMD_bar = Z-M*D_bar;
157     s_square = ZminusMD_bar'*ZminusMD_bar/(2*N-3);
158     Cov_ICP = s_square*InvMM;
159
160     deltaPose_bar = g;
161     deltaPose_bar_Cov = Cov_ICP;
162 else
163     deltaPose_bar = [];
164     deltaPose_bar_Cov = [];
165 end

```

7.3.vi XY2IJ

```
1 function [Map_i,Map_j] = XY2IJ(x, y, grid_size, Map_x_min, Map_y_min)
2
3 % XY2IJ
4 %
5 % Author: Chieh-Chih (Bob) Wang [bobwang@cs.cmu.edu]
6 % Created: Oct. 31, 2002.
7 % Modified: Nov. 6, 2003.
8 % (c) 2002-2003 Chieh-Chih Wang. All Rights Reserved.
9
10 % Changed to the vec form
11 Map_i = round((x - Map_x_min)/grid_size + 0.5);
12 Map_j = round((y - Map_y_min)/grid_size + 0.5);
```

7.3.vii head2tail no theta

```
1 function Xik = head2tail.no_theta(Xij, Xjk)
2
3 % Compounding Operation: head2tail
4 %
5 % Input:
6 %   Xij = [x_ij; y_ij; theta_ij]
7 %   Xjk = [x_jk; y_jk]
8 % Output:
9 %   Xik = [x_ik; y_ik]
10 %
11 % Author: Chieh-Chih (Bob) Wang [bobwang@cs.cmu.edu]
12 % Created: Nov. 8, 2002.
13 % Modified: Nov. 6, 2003.
14
15
16 cosTheta_ij = cos(Xij(3,1));
17 sinTheta_ij = sin(Xij(3,1));
18
19 % Xik(1,1) = Xjk(1,1)*cosTheta_ij - Xjk(2,1)*sinTheta_ij + Xij(1,1);
20 % Xik(2,1) = Xjk(1,1)*sinTheta_ij + Xjk(2,1)*cosTheta_ij + Xij(2,1);
21
22 % Change for Vec...
23 Xik(1,:) = Xjk(1,:)*cosTheta_ij - Xjk(2,:)*sinTheta_ij + Xij(1,1);
24 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

```
1 % im = appendimages(image1, image2)
2 %
3 % Return a new image that appends the two images side-by-side.
4
5 function im = appendimages(image1, image2)
6
7 % Select the image with the fewest rows and fill in enough empty rows
8 %   to make it the same height as the other image.
9 rows1 = size(image1,1);
10 rows2 = size(image2,1);
11
12 if (rows1 < rows2)
13     image1(rows2,1) = 0;
14 else
15     image2(rows1,1) = 0;
16 end
17
18 % Now append both images side-by-side.
19 im = [image1 image2];
```

7.5.ii Feature Matching

```
1 %This code is using harris algorithm to find corners and extractFeatures
2 %and matchFeatures Functions in matlab image toolbox to match features
3 %in two images.
4
5 close all
6 clear
7 clc
8 x=0;
9 y=0;
10
11
12 I1=imread('whitehouse.left.png');
13 I2=imread('whitehouse.right.png');
14
15 %Check whether the image is RGB and convert that to gray
16 if length(size(I1))==3 & length(size(I2))==3;
17     BW1=rgb2gray(I1);
18     BW2=rgb2gray(I2);
19     x=1;
20 else
21     BW1=I1;
22     BW2=I2;
23 end
24
25 im=appendimages(I1,I2); %create a one image using two images
26
27 p1 = detectHarrisFeatures(BW1); %Find corners using Harris Algorithm
28 p2 = detectHarrisFeatures(BW2);
29
30
31 [features1, valid_points1] = extractFeatures(BW1, p1); %Extract Features and valid points using
    ↳ detected corners
32 [features2, valid_points2] = extractFeatures(BW2, p2);
33
34 indexPairs = matchFeatures(features1, features2, 'MatchThreshold',80); %Finding Matching Features of
    ↳ two images
35
36 matchedPoints1 = valid_points1(indexPairs(:, 1), :);
37 matchedPoints2 = valid_points2(indexPairs(:, 2), :);
38
39 figure; showMatchedFeatures(BW1, BW2, matchedPoints1, matchedPoints2, 'montage'); %show matching
    ↳ features
```

7.5.iii Harris

```
1 % HARRIS - Harris corner detector
2 %
3 % Usage: [cim, r, c] = harris(im, sigma, thresh, radius, disp)
4 %
5 % Arguments:
6 %     im      - image to be processed.
7 %     sigma   - standard deviation of smoothing Gaussian. Typical
8 %               values to use might be 1-3.
9 %     thresh  - threshold (optional). Try a value ~1000.
10 %    radius  - radius of region considered in non-maximal
11 %               suppression (optional). Typical values to use might
12 %               be 1-3.
13 %    disp    - optional flag (0 or 1) indicating whether you want
14 %               to display corners overlayed on the original
15 %               image. This can be useful for parameter tuning.
16 %
17 % Returns:
18 %     cim     - binary image marking corners.
19 %     r       - row coordinates of corner points.
20 %     c       - column coordinates of corner points.
21 %
22 % If thresh and radius are omitted from the argument list 'cim' is returned
23 % as a raw corner strength image and r and c are returned empty.
24 %
25 % References:
26 % C.G. Harris and M.J. Stephens. "A combined corner and edge detector",
27 % Proceedings Fourth Alvey Vision Conference, Manchester.
28 % pp 147-151, 1988.
29 %
30 % Alison Noble, "Descriptions of Image Surfaces", PhD thesis, Department
31 % of Engineering Science, Oxford University 1989, p45.
32 %
33 %
34 % Author:
35 % Peter Kovesi
36 % Department of Computer Science & Software Engineering
37 % The University of Western Australia
38 % pk @ csse uwa edu au
39 % http://www. csse.uwa.edu.au/~pk
40 %
41 % March 2002    - original version
42 % December 2002 - updated comments
43 %
44 function [cim, r, c] = harris(im, sigma, thresh, radius, disp)
45
46     error(nargchk(2,5,nargin));
47
48     if ~isa(im,'double')
49         im = double(im);
50     end
51
52     dx = [-1 0 1; -1 0 1; -1 0 1]; % Derivative masks
53     dy = dx';
54
55     Ix = conv2(im, dx, 'same');    % Image derivatives
56     Iy = conv2(im, dy, 'same');
57
58     % Generate Gaussian filter of size 6*sigma (+/- 3sigma) and of
59     % minimum size 1x1.
60     g = fspecial('gaussian',max(1,fix(6*sigma)), sigma);
61
62     Ix2 = conv2(Ix.^2, g, 'same'); % Smoothed squared image derivatives
63     Iy2 = conv2(Iy.^2, g, 'same');
64     Ixy = conv2(Ix.*Iy, g, 'same');
65
66     % Compute the Harris corner measure. Note that there are two measures
67     % that can be calculated. I prefer the first one below as given by
```



```

68 % Nobel in her thesis (reference above). The second one (commented out)
69 % requires setting a parameter, it is commonly suggested that k=0.04 - I
70 % find this a bit arbitrary and unsatisfactory.
71
72 cim = (Ix2.*Iy2 - Ixy.^2)./(Ix2 + Iy2 + eps); % My preferred measure.
73 % k = 0.04;
74 % cim = (Ix2.*Iy2 - Ixy.^2) - k*(Ix2 + Iy2).^2; % Original Harris measure.
75
76 if nargin > 2 % We should perform nonmaximal suppression and threshold
77
78 % Extract local maxima by performing a grey scale morphological
79 % dilation and then finding points in the corner strength image that
80 % match the dilated image and are also greater than the threshold.
81 size = 2*radius+1; % Size of mask.
82 mx = ordfilt2(cim,size^2,ones(size)); % Grey-scale dilate.
83 cim = (cim==mx)&(cim>thresh); % Find maxima.
84
85 [r,c] = find(cim); % Find row,col coords.
86
87 if nargin==5 & disp % overlay corners on original image
88     imshow(im,1), hold on
89     plot(c,r,'r+'), title('corners detected');
90 end
91
92 else % leave cim as a corner strength image and make r and c empty.
93     r = []; c = [];
94 end

```

7.5.iv Show Keys

```
1 % showkeys(image, locs)
2 %
3 % This function displays an image with SIFT keypoints overlaid.
4 %   Input parameters:
5 %     image: the file name for the image (grayscale)
6 %     locs: matrix in which each row gives a keypoint location (row,
7 %           column, scale, orientation)
8
9 function showkeys(image, locs)
10
11 disp('Drawing SIFT keypoints ...');
12
13 % Draw image with keypoints
14 figure('Position', [50 50 size(image,2) size(image,1)]);
15 colormap('gray');
16 imagesc(image);
17 hold on;
18 imsize = size(image);
19 for i = 1: size(locs,1)
20     % Draw an arrow, each line transformed according to keypoint parameters.
21     TransformLine(imsize, locs(i,:), 0.0, 0.0, 1.0, 0.0);
22     TransformLine(imsize, locs(i,:), 0.85, 0.1, 1.0, 0.0);
23     TransformLine(imsize, locs(i,:), 0.85, -0.1, 1.0, 0.0);
24 end
25 hold off;
26
27
28 % ----- Subroutine: TransformLine -----
29 % Draw the given line in the image, but first translate, rotate, and
30 % scale according to the keypoint parameters.
31 %
32 % Parameters:
33 %   Arrays:
34 %     imsize = [rows columns] of image
35 %     keypoint = [subpixel.row subpixel.column scale orientation]
36 %
37 %   Scalars:
38 %     x1, y1; beginning of vector
39 %     x2, y2; ending of vector
40 function TransformLine(imsize, keypoint, x1, y1, x2, y2)
41
42 % The scaling of the unit length arrow is set to approximately the radius
43 % of the region used to compute the keypoint descriptor.
44 len = 6 * keypoint(3);
45
46 % Rotate the keypoints by 'ori' = keypoint(4)
47 s = sin(keypoint(4));
48 c = cos(keypoint(4));
49
50 % Apply transform
51 r1 = keypoint(1) - len * (c * y1 + s * x1);
52 c1 = keypoint(2) + len * (-s * y1 + c * x1);
53 r2 = keypoint(1) - len * (c * y2 + s * x2);
54 c2 = keypoint(2) + len * (-s * y2 + c * x2);
55
56 line([c1 c2], [r1 r2], 'Color', 'c');
```

7.5.v Sift

```
1 % [image, descriptors, locs] = sift(imageFile)
2 %
3 % This function reads an image and returns its SIFT keypoints.
4 %   Input parameters:
5 %     imageFile: the file name for the image.
6 %
7 %   Returned:
8 %     image: the image array in double format
9 %     descriptors: a K-by-128 matrix, where each row gives an invariant
10 %       descriptor for one of the K keypoints. The descriptor is a vector
11 %       of 128 values normalized to unit length.
12 %     locs: K-by-4 matrix, in which each row has the 4 values for a
13 %       keypoint location (row, column, scale, orientation). The
14 %       orientation is in the range  $[-\pi, \pi]$  radians.
15 %
16 % Credits: Thanks for initial version of this program to D. Alvaro and
17 %       J.J. Guerrero, Universidad de Zaragoza (modified by D. Lowe)
18
19 function [image, descriptors, locs] = sift(imageFile)
20
21 % Load image
22 image = imread(imageFile);
23
24 % If you have the Image Processing Toolbox, you can uncomment the following
25 % lines to allow input of color images, which will be converted to grayscale.
26 % if isrgb(image)
27 %     image = rgb2gray(image);
28 % end
29
30 [rows, cols] = size(image);
31
32 % Convert into PGM imagefile, readable by "keypoints" executable
33 f = fopen('tmp.pgm', 'w');
34 if f == -1
35     error('Could not create file tmp.pgm.');
```

```
36 end
37 fprintf(f, 'P5\n%d\n%d\n255\n', cols, rows);
38 fwrite(f, image, 'uint8');
39 fclose(f);
40
41 % Call keypoints executable
42 if isunix
43     command = '!.sift ';
44 else
45     command = '!siftWin32 ';
46 end
47 command = [command ' <tmp.pgm >tmp.key'];
48 eval(command);
49
50 % Open tmp.key and check its header
51 g = fopen('tmp.key', 'r');
52 if g == -1
53     error('Could not open file tmp.key.');
```

```
54 end
55 [header, count] = fscanf(g, '%d %d', [1 2]);
56 if count ~= 2
57     error('Invalid keypoint file beginning.');
```

```
58 end
59 num = header(1);
60 len = header(2);
61 if len ~= 128
62     error('Keypoint descriptor length invalid (should be 128).');
```

```
63 end
64
65 % Creates the two output matrices (use known size for efficiency)
66 locs = double(zeros(num, 4));
67 descriptors = double(zeros(num, 128));
```

```
68
69 % Parse tmp.key
70 for i = 1:num
71     [vector, count] = fscanf(g, '%f %f %f %f', [1 4]); %row col scale ori
72     if count ~= 4
73         error('Invalid keypoint file format');
74     end
75     locs(i, :) = vector(1, :);
76
77     [descrip, count] = fscanf(g, '%d', [1 len]);
78     if (count ~= 128)
79         error('Invalid keypoint file value. ');
80     end
81     % Normalize each input vector to unit length
82     descrip = descrip / sqrt(sum(descrip.^2));
83     descriptors(i, :) = descrip(1, :);
84 end
85 fclose(g);
```

7.5.vi Main Q5 Code

```
1 close all
2 clear
3 clc
4
5 I1=imread('image.seq10.png'); %Read image file 1
6 I2=imread('image.seq11.png');%Read image file 2
7
8 BW1=rgb2gray(I1); %convert to gray scale
9 BW2=rgb2gray(I2);
10
11 %a.1 Sobel Edge detector
12 E_sobel_I1=edge(BW1,'sobel');
13 figure
14 imshow(E_sobel_I1);
15 title('Sobel Edge Detector');
16 %E_sobel_I2=edge(BW2,'sobel');
17
18 %a.2 Canny Edge Detector
19 E_canny_I1=edge(BW1,'canny',[0.1,0.4]);
20 figure
21 imshow(E_canny_I1);
22 title('Canny Edge Detector');
23 %E_canny_I2=edge(BW2,'canny',[0.1,0.4]); %E_canny_I2=edge(I2_gray,'canny',[0.05,0.25]);
24
25 %Harris Corners
26 I1_harris=corner(BW1,'harris');
27 figure
28 imshow(BW1);
29 hold on
30 plot(I1_harris(:,1),I1_harris(:,2),'r*');
31 title('Harris corners');
32
33 %c. Sift Corners for I1
34 imwrite(BW1,'BW1.png') %writing gray image of I1 as 'BW1.png'
35 [image, descripts, locs] = sift('BW1.png');
36 showkeys(image, locs);
37
38 %d. Matching Features using given match function
39 match('whitehouse.left.png','whitehouse.right.png');
```

7.6 Question 6

7.6.i Main Code

```
1  %Establish a connection to the motor arm, calculate centroids and
2  %orientations, and send commands to manoeuvre boxes
3  %%Code adapted from the code given as part of the assignment specification
4
5  function ass2q6_connect()
6
7  clear all;
8  close all;
9  clc;
10
11  %% SETUP CAMERA AND OBTAIN IMAGE
12  vid = videoinput('pointgrey',1, 'Mono8.640x480');
13  pause(1);
14  I = getsnapshot(vid);
15
16  %% INSERT IMAGE PROCESSING CODE HERE
17  %Should use image I, find cubes, and determine commands for arm
18  %
19  %
20  %
21  % I = imread('tt3.png');    %Can load an image from file instead of camera feed
22  imshow(I);
23  drawnow();
24  %
25  [CentroidsX, CentroidsY, OrientationAngle] = find_centroids_orientation_grey(I);
26  %
27  %
28
29
30  %% OPEN NETWORK CONNECTION
31  t=tcip('192.168.0.1', 2020, 'NetworkRole', 'client');
32  fopen(t);
33
34
35  %% CONTROL ARM
36  %Should send out commands via TCPIP as strings (format below)
37
38  %command.string = '<x0,y260>\n'
39  %sendCommand(t,command.string)    %Send command string to robot via network
40
41  %Include brackets in command
42
43  %Full list of commands:
44  % <x0,y360>\n = PositionTool(0,360)
45  % <h0>\n = setToolHeight(0)
46  % <a90>\n = setToolAngle(90)
47  % <o>\n = OpenGripper
48  % <c>\n = CloseGripper
49
50  height = 0.1;
51
52  strings = zeros(11,8);
53  % t=1;
54  for i = 1:length(CentroidsX)
55
56      sendCommand(t, '<h4>\n');
57      sendCommand(t, '<o>\n');
58      sendCommand(t, sprintf('<a%.0f>\n', OrientationAngle(i)));
59      sendCommand(t, sprintf('<x%.0f,y%.0f>\n',CentroidsX(i),CentroidsY(i)));
60      sendCommand(t, '<h0>\n');
61      sendCommand(t, '<c>\n');
62      sendCommand(t, '<h4>\n');
63      sendCommand(t, '<x0,y490>\n');
64      sendCommand(t, '<a0>\n');
65      sendCommand(t, sprintf('<h%.1f>\n', height));
```

```
66     height = height + 1;
67     sendCommand(t, '<o>\n');
68
69 end
70
71 for i = 1:8
72     for k = i:11
73         sendCommand(t, strings(k,i));
74     end
75 end
76 %Example code
77 % sendCommand(t, '<o>\n')           %Open gripper
78 % sendCommand(t, '<x0,y490>\n')    %Move to (0,490)
79 % sendCommand(t, '<c>\n')           %Close gripper
80 % sendCommand(t, '<x0,y390>\n')    %Move to (0,390)
81 % sendCommand(t, '<o>\n')           %Open gripper
82 % sendCommand(t, '<h2>\n')         %Set tool height to 2 cubes
83 % sendCommand(t, '<a45>\n')        %Set tool angle to 45 deg
84
85
86
87 % CLOSE NETWORK PORT AND CAMERA
88 delete(vid)
89 fclose(t);
90
91 end
```

7.6.ii Image Processing and Position Finding

```
1 function [CentroidsX, CentroidsY, OrientationAngle] = find.centroids.orientation.grey(I)
2
3 % I=imread('ttl.png'); % read image
4
5 I2=imcrop(I,[68.5 4.5 491 472]); %isolate the workingspace
6 I3=double(I2)/255; %convert to double
7 I4=adapthisteq(I3); %enhance contrast using adaptive histogram equalization
8 I5=I4<0.10;%0.1 %0.11 % find boxes using a threshold
9
10 C1=bwareaopen(I5,1000); %remove objects less than 1000 pixels/ Remove unnecessory regions
11 se=strel('square',15); %create a structuring element of size 15 square
12 C2=imclose(C1,se); %morphologically close image
13 C4=imfill(C2,'holes'); %fill holes
14
15 s=regionprops(C4,'centroid'); %find centroids of regions
16 centroids=cat(1,s.Centroid); %make a array using centroids in the structured array
17 imshow(C4)
18 hold on
19 plot(centroids(:,1),centroids(:,2),'b*') %plot centorids with boxes
20
21 %Assigns numbers to each centroid
22 numbers = 1:length(centroids);
23 strValues = strtrim(cellstr(num2str(numbers(:),'(%d)')));
24 text(centroids(:,1),centroids(:,2),strValues,'VerticalAlignment','bottom');
25
26 %GetOrientation
27 TR = 2;
28 LT = 8;
29
30 p = regionprops(C4, 'Extrema');
31
32 hold on
33
34
35 for i = 1:length(p)
36
37     %      TR to LT - Best so far
38     sides1 = p(i).Extrema(TR,:) - p(i).Extrema(LT,:); % Returns the sides of the square triangle
39     %      ↳ that completes the two chosen extrema:
40     plot(p(i).Extrema(LT,1),p(i).Extrema(LT,2),'bx','MarkerSize',20);
41     plot(p(i).Extrema(TR,1),p(i).Extrema(TR,2),'bx','MarkerSize',20);
42
43     OrAn1(i) = rad2deg(atan(-sides1(2)/sides1(1))); % Note the 'minus' sign compensates for the
44     %      ↳ inverted y-values in image coordinates
45
46 end
47
48 hold off
49
50 %Find and remove fiducial boxes from centroids and orientation
51
52 x1=centroids(:,1);
53 y1=centroids(:,2);
54
55 for i=1:length(x1)
56     if ((x1(i)< 60 & x1(i)>50)& (y1(i)<442 & y1(i)>432 ))
57         B1=[x1(i),y1(i)];
58     end
59 end
60
61 %Remove coordinates of fiducial boxes
62 OrAn1(B1(1,1)==x1)=[];
63 x1(B1(1,1)==x1)=[];
64 y1(B1(1,2)==y1)=[];
65
66 %find fiducial box.2 (270,220)
67 for i=1:length(x1)
```



```

66     if ((x1(i)< 459 & x1(i)>249)& (y1(i)<431 & y1(i)>421 ))
67         B2=[x1(i),y1(i)];
68     end
69 end
70
71
72 OrAn1(B2(1,1)==x1)=[];
73 x1(B2(1,1)==x1)=[];
74 y1(B2(1,2)==y1)=[];
75
76 %find fiducial box_3 (-270,523)
77 for i=1:length(x1)
78     if ((x1(i)< 53 & x1(i)>43)& (y1(i)<223 & y1(i)>213 ))
79         B3=[x1(i),y1(i)];
80     end
81 end
82
83 OrAn1(B3(1,1)==x1)=[];
84 x1(B3(1,1)==x1)=[];
85 y1(B3(1,2)==y1)=[];
86
87 %When only fiducial boxes present
88 TF=isempty(x1);
89 if TF==1
90     warning('No objects detected other than fiducial boxes')
91 end
92
93 Yr=abs(p.dist.q6(B1,B2,x1,y1)); %distance from line going along centroids of box1 & box 2 (reference
    ↪ )
94 Xr=p.dist.q6(B1,B3,x1,y1); %distance from line going along centroids of box1 & box 3 (reference)
95
96 %Find the reference point according to the world coordinates
97 %X 792.8691 = 540 real , %Y 433.3300 = 303 real
98 Yreal=((Yr*303)/219.3306)+220; %REAL Y COORDINATE
99 Xreal=((Xr*540)/399.4796)-270; %REAL x COORDINATE
100
101 %Return X-Y coordinates of the centroid in real world coordinates, along
102 %with the orientation
103 CentroidsX = Xreal;
104 CentroidsY = Yreal;
105 OrientationAngle = OrAn1;
106
107 end

```

7.6.iii Perpendicular Distance

```
1 %Find the perpendicular distances between two boxes along their centroids
2
3 function [pDist] = p-dist-q6(B1,B2,u,v)
4
5 LP1=[B1(1,1),B1(1,2)];
6 LP2=[B2(1,1),B2(1,2)];
7
8 D=sqrt((LP2(1)-LP1(1))^2+(LP2(2)-LP1(2))^2);
9
10 r=(u*(LP1(2)-LP2(2))+v*(LP2(1)-LP1(1))+LP2(2)*LP1(1)-LP1(2)*LP2(1));
11
12 pDist=r/D;
13
14
15 end
```

7.6.iv Send Command

```
1 %Send instructions to the Robotic Arm
2 %%Code as presented as part of the assignment specification
3
4 function sendCommand(t,command)
5
6     %Send command to TCP/IP port
7     fprintf(t,command)
8
9     %     %Pause until a message is received
10    while(~t.BytesAvailable)
11        end
12    %     %Then flush the input buffer
13    flushinput(t)
14 end
```