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:

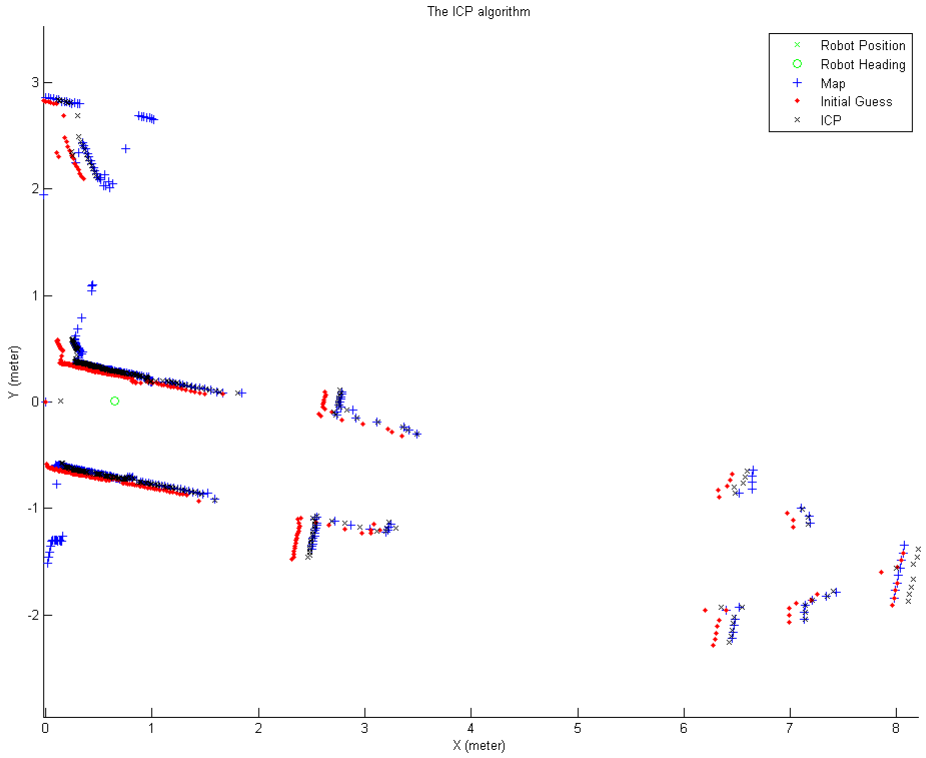


Figure 1: ICP estimate for maximum iterations of 40 and grid size of 0.005

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:

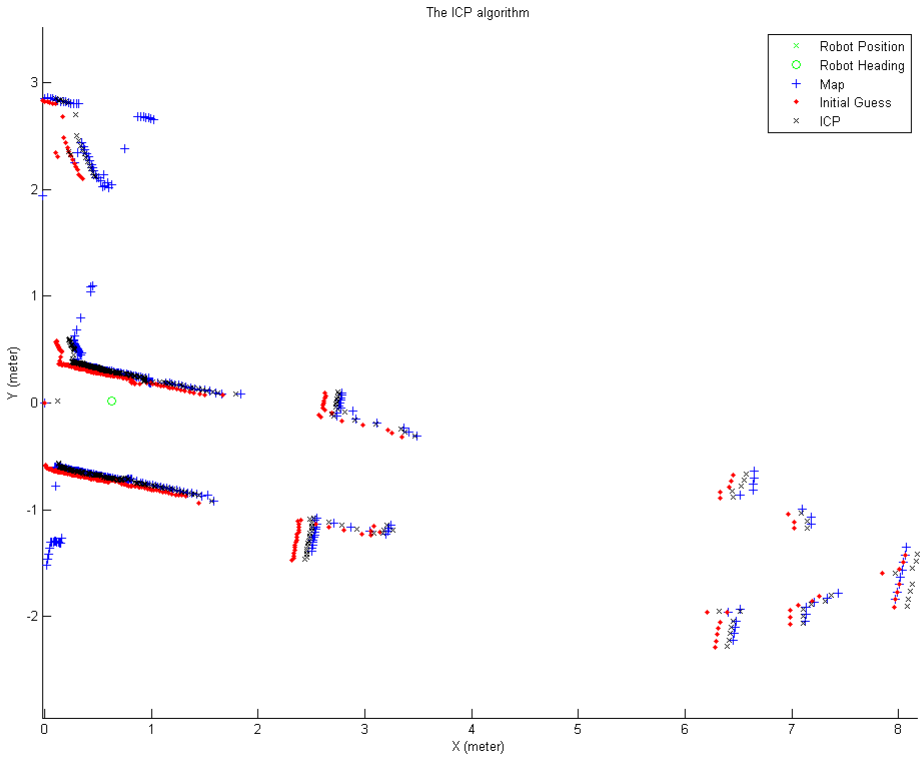


Figure 2: ICP estimate for maximum iterations of 40 and grid size of 0.001

Next, for a grid size of 0.1:

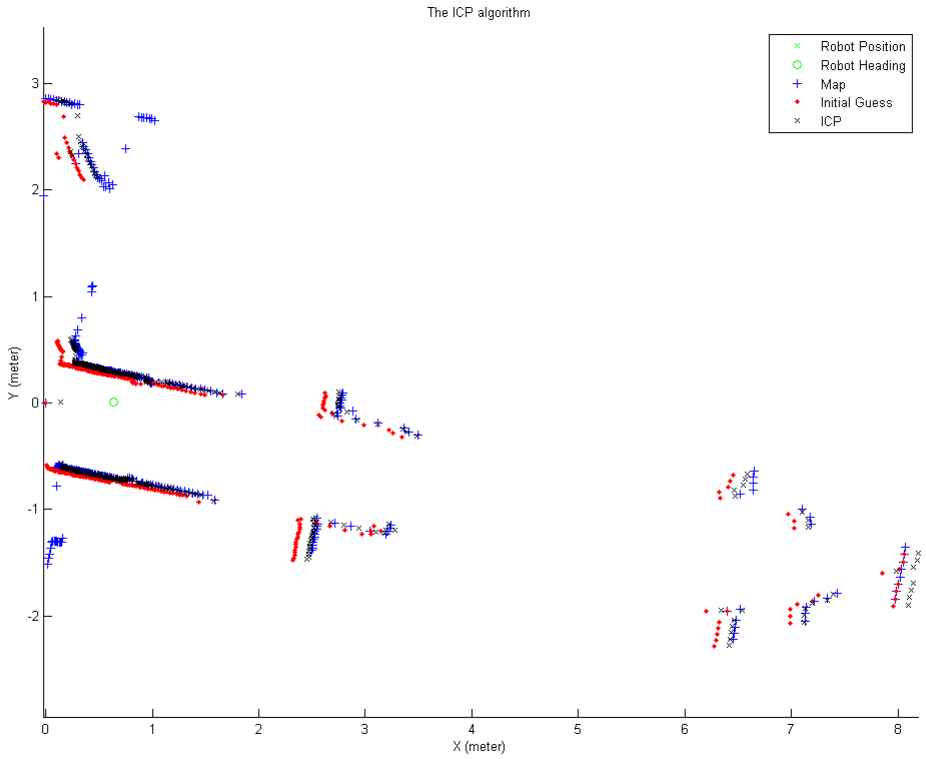


Figure 3: ICP estimate for maximum iterations of 40 and grid size of 0.005

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 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:

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

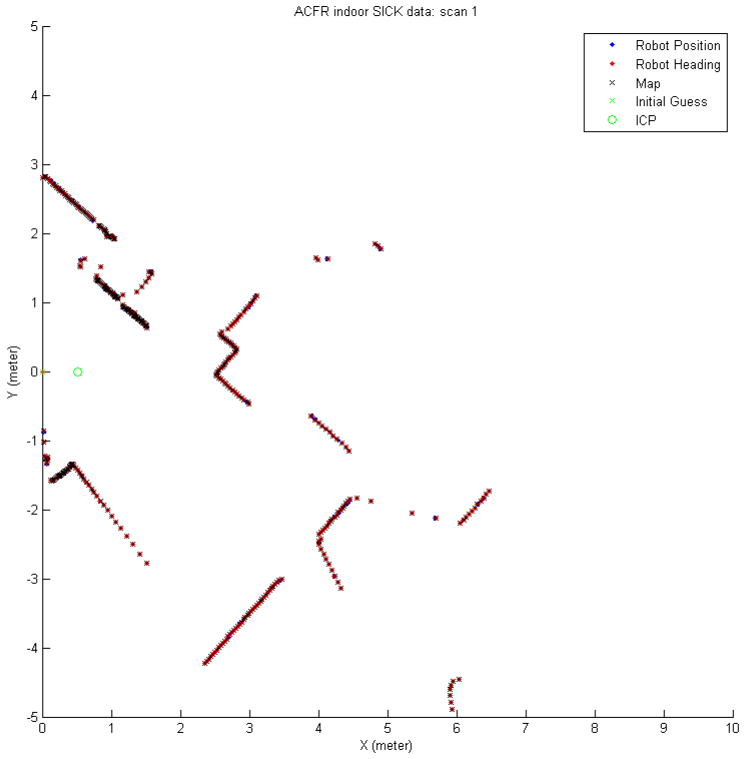
As can be seen, the covariance matrix is extremely close to zero. This is indicative of the factors involved being completely independent, though it does not confirm this. Again, seeing as this is run from a single frame, it is not an indicator for the overall relationship - we have used far too few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

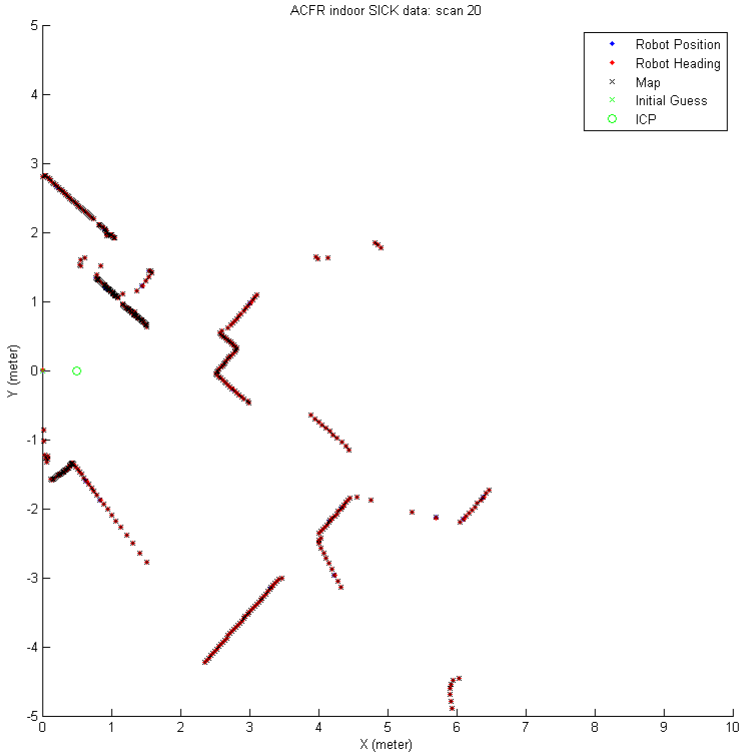
Problems with the Algorithm

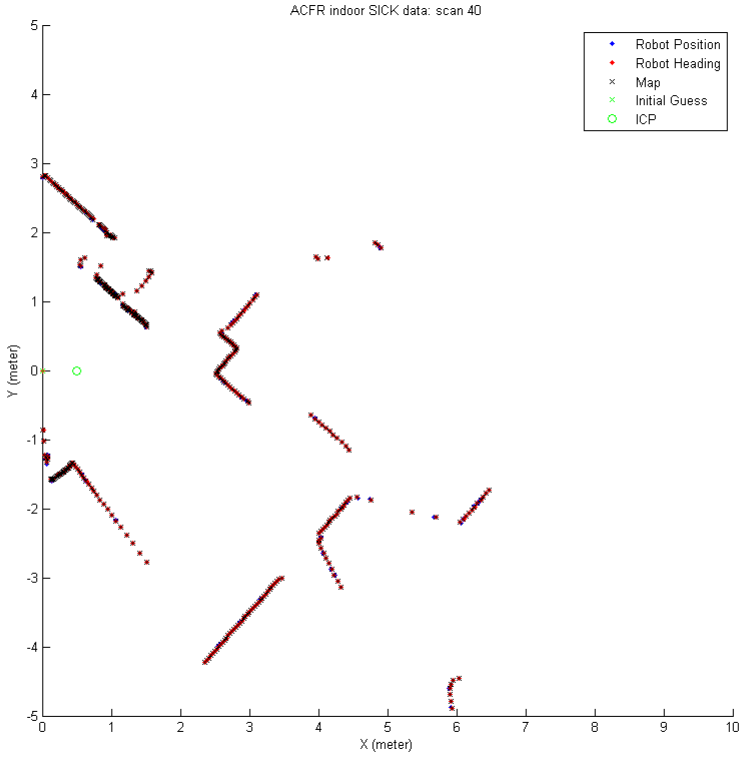
ICPv4.m takes little into account to do with angles/rotations, mostly using x and y values. This would mean that any rotational movement in the map would be taken poorly into account, as will be evident further on.

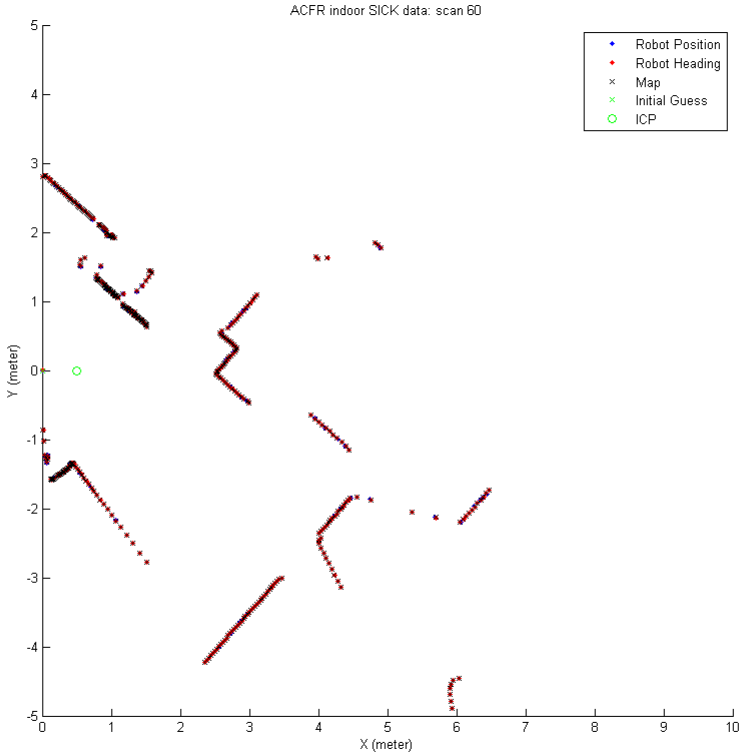
3..ii Part B

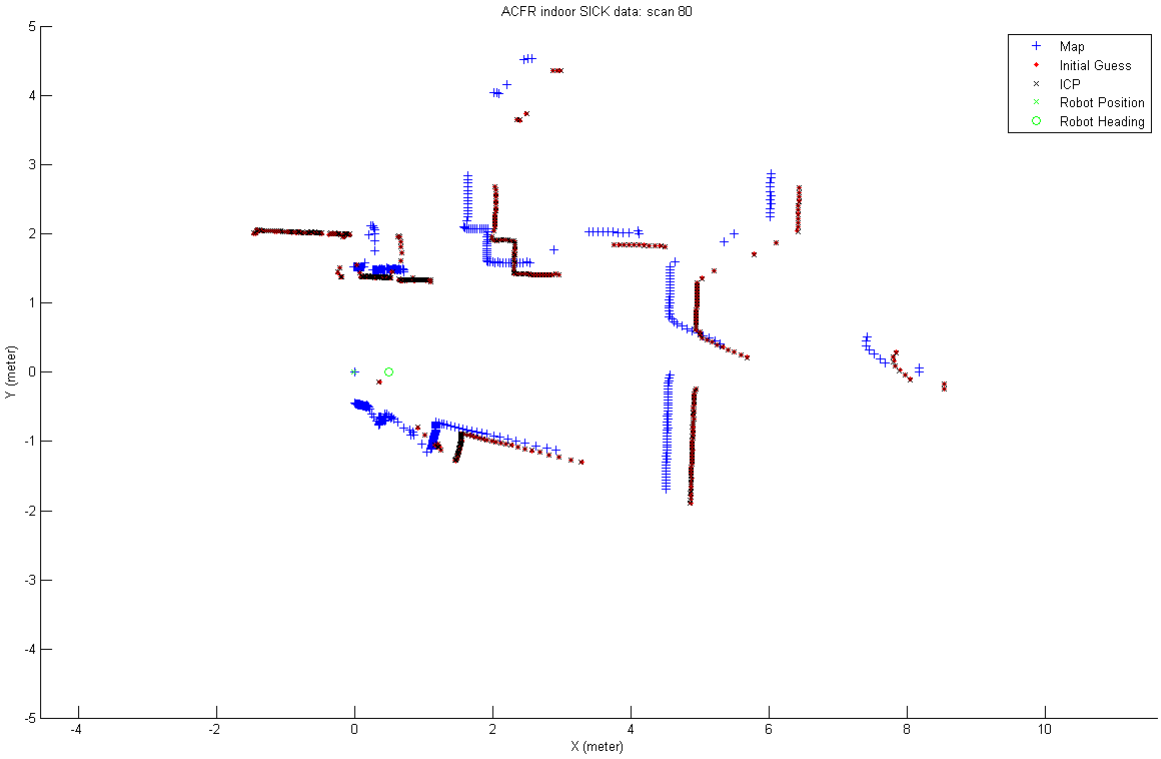
Incorporating the ICPv4.m algorithm into the laserShowACFR.m program enables us to build a real time picture of the movement of the vehicle and its perceived surroundings, relative to the actual mapped data. Observe:

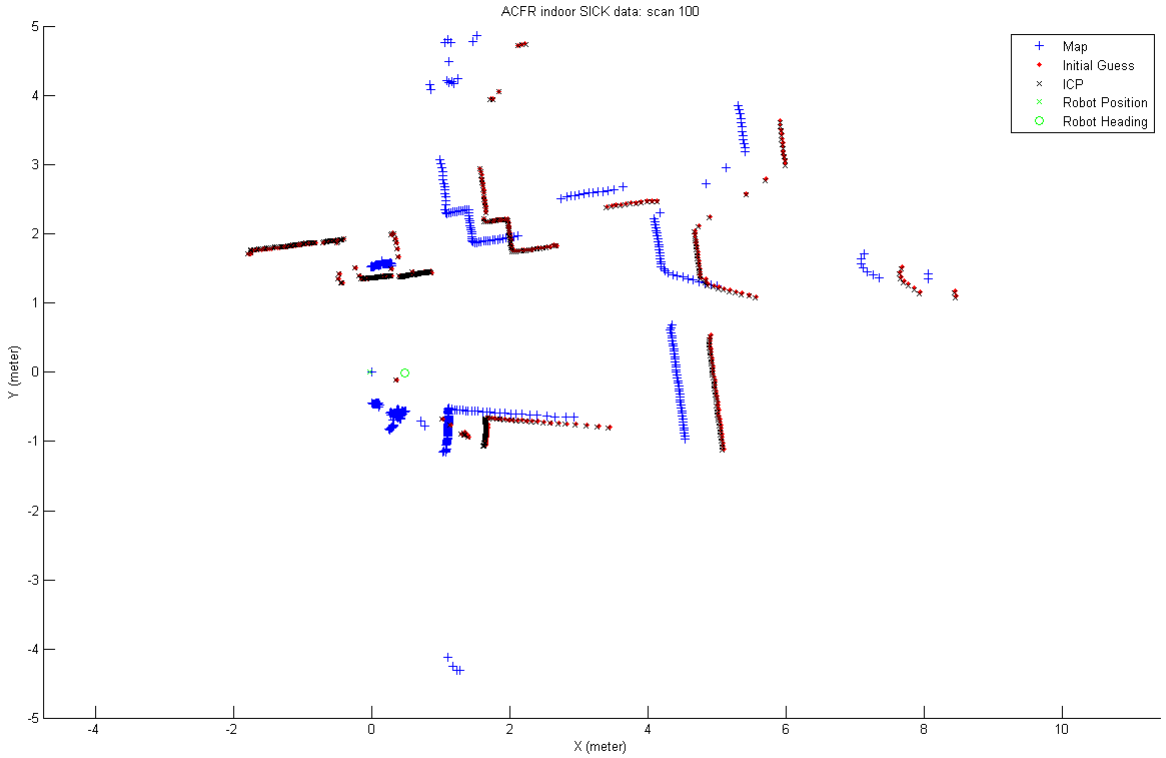


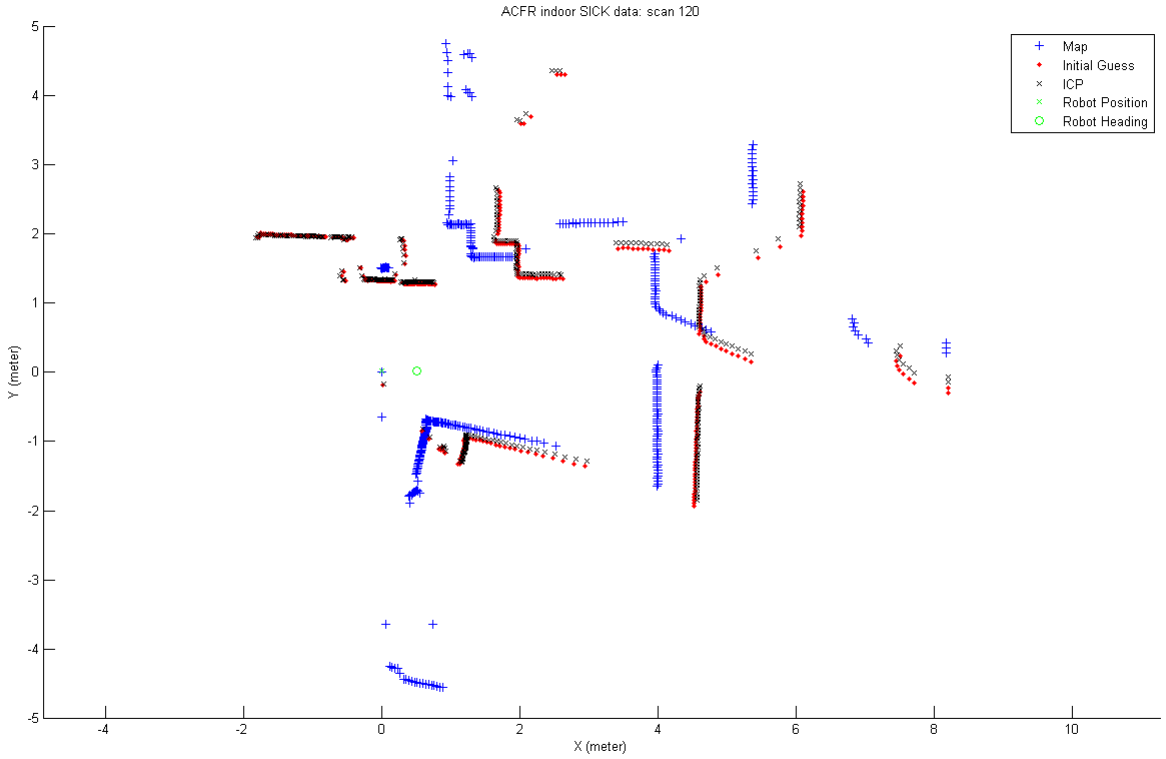

ACFR indoor SICK data: scan 20
Robot Position
Robot Heading
Map
Initial Guess
ICP
Y (meter)
X (meter)

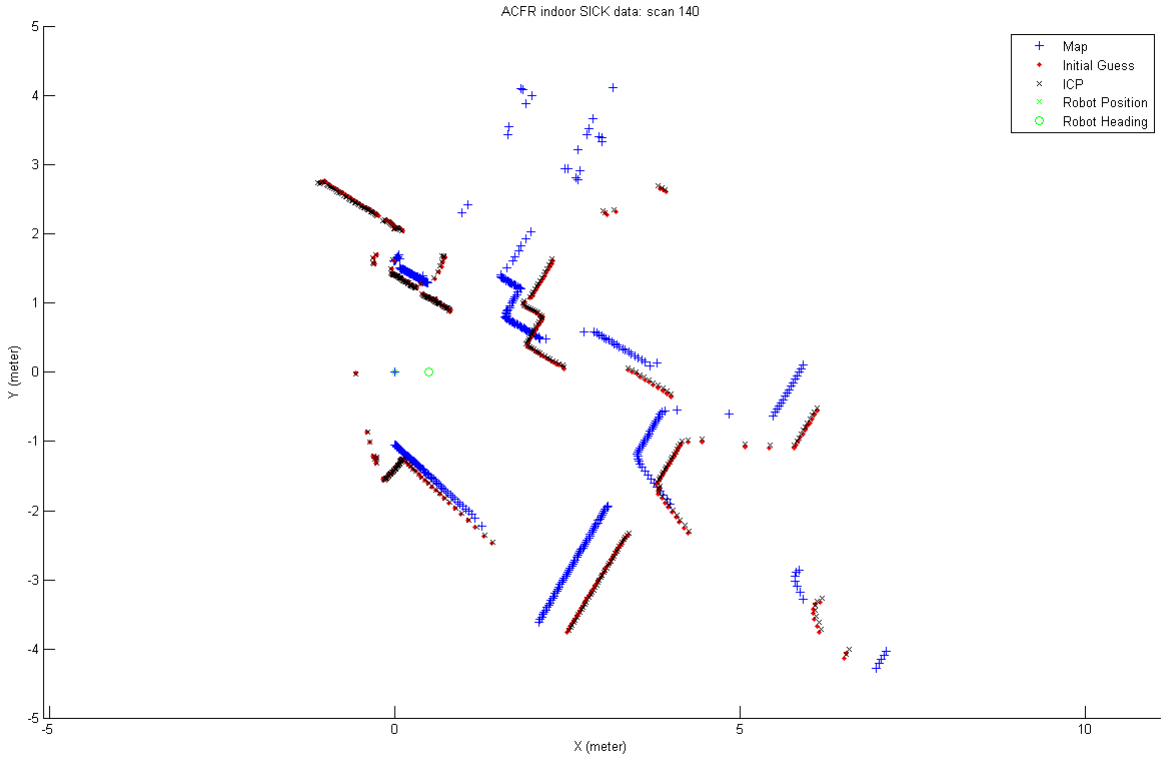

ACFR indoor SICK data: scan 40
Robot Position
Robot Heading
Map
Initial Guess
ICP
Y (meter)
X (meter)

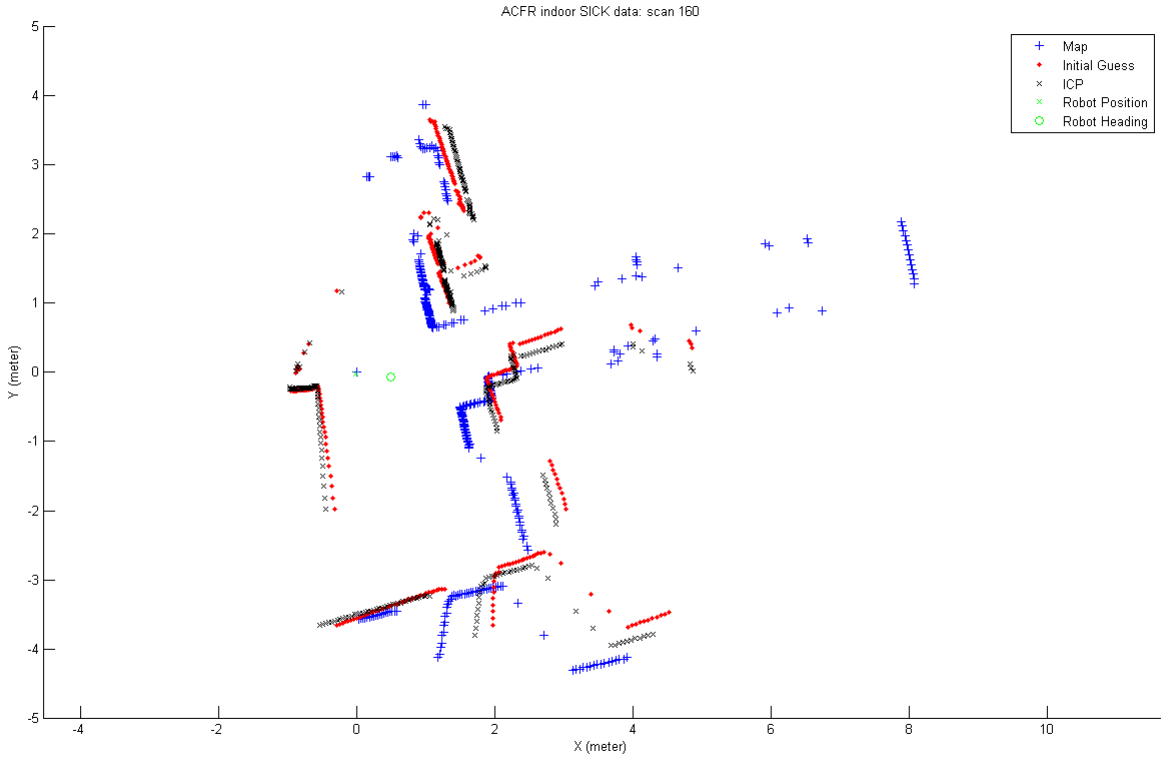

ACFR indoor SICK data: scan 60
Robot Position
Robot Heading
Map
Initial Guess
ICP
Y (meter)
X (meter)

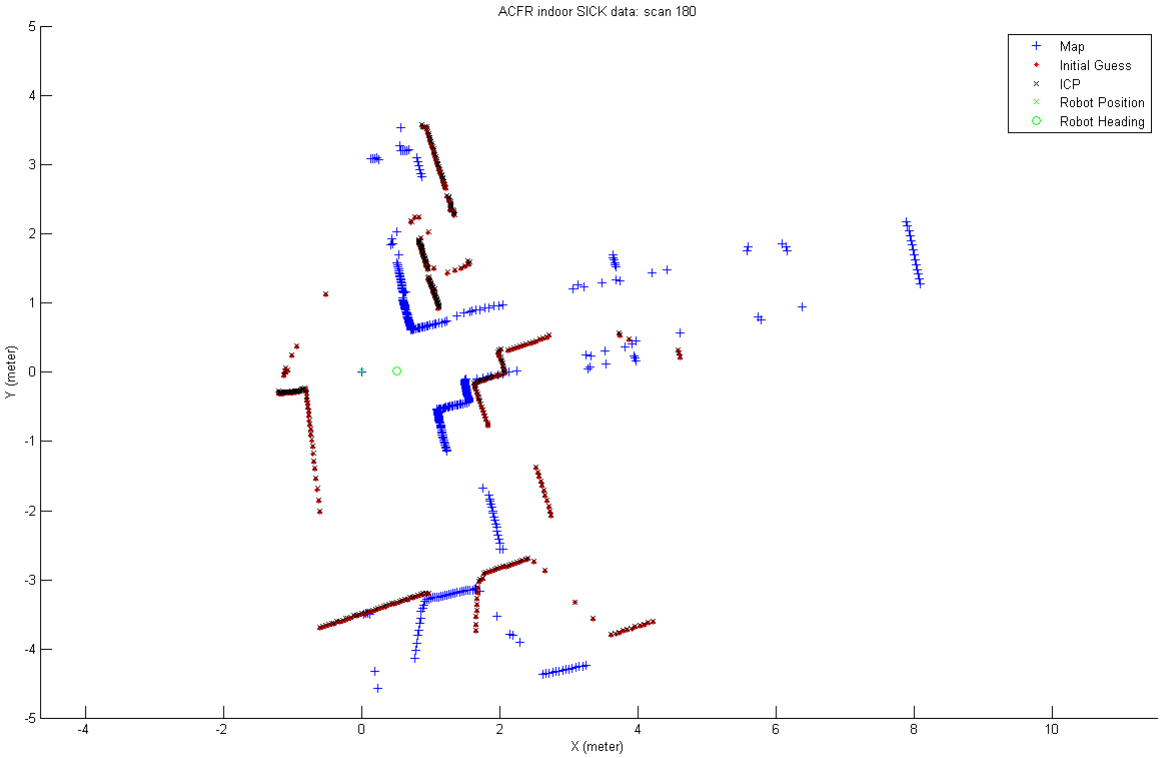

ACFR indoor SICK data: scan 80
Map
Initial Guess
ICP
Robot Position
Robot Heading
Y (meter)
X (meter)

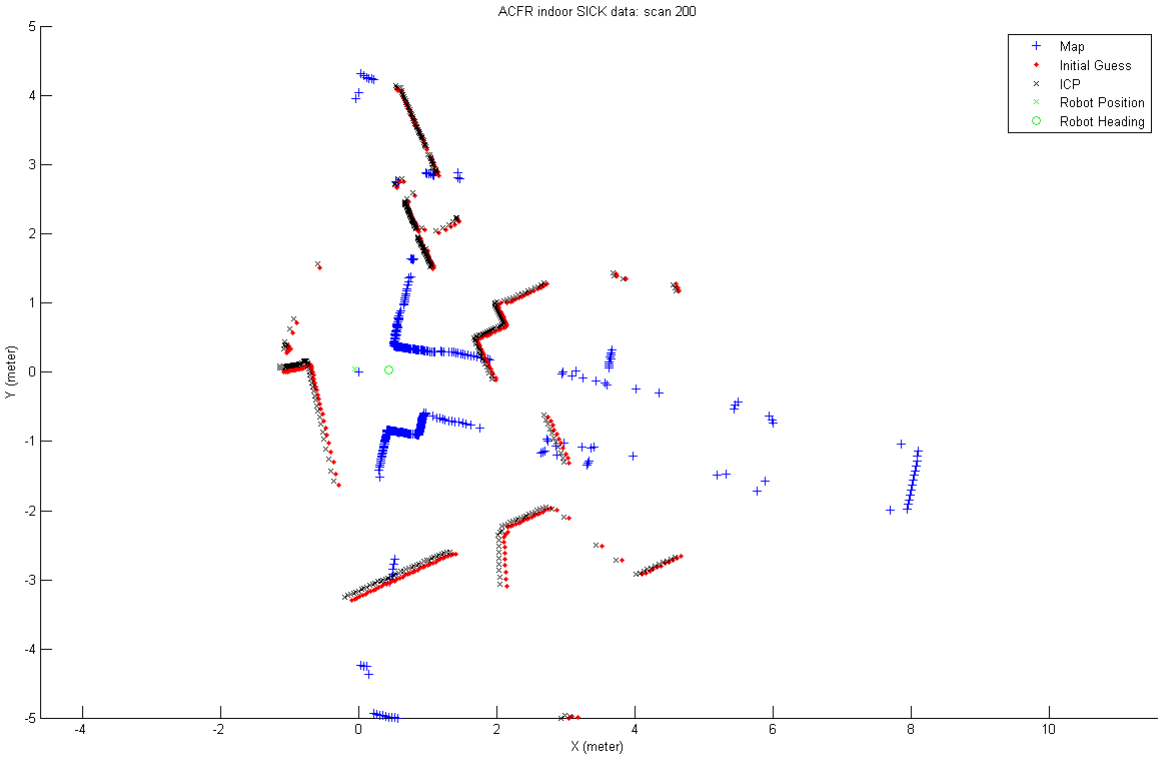

ACFR indoor SICK data: scan 100
Map
Initial Guess
ICP
Robot Position
Robot Heading
Y (meter)
X (meter)
5
4
3
2
1
0
-1
-2
-3
-4
-5
-4
-2
0
2
4
6
8
10

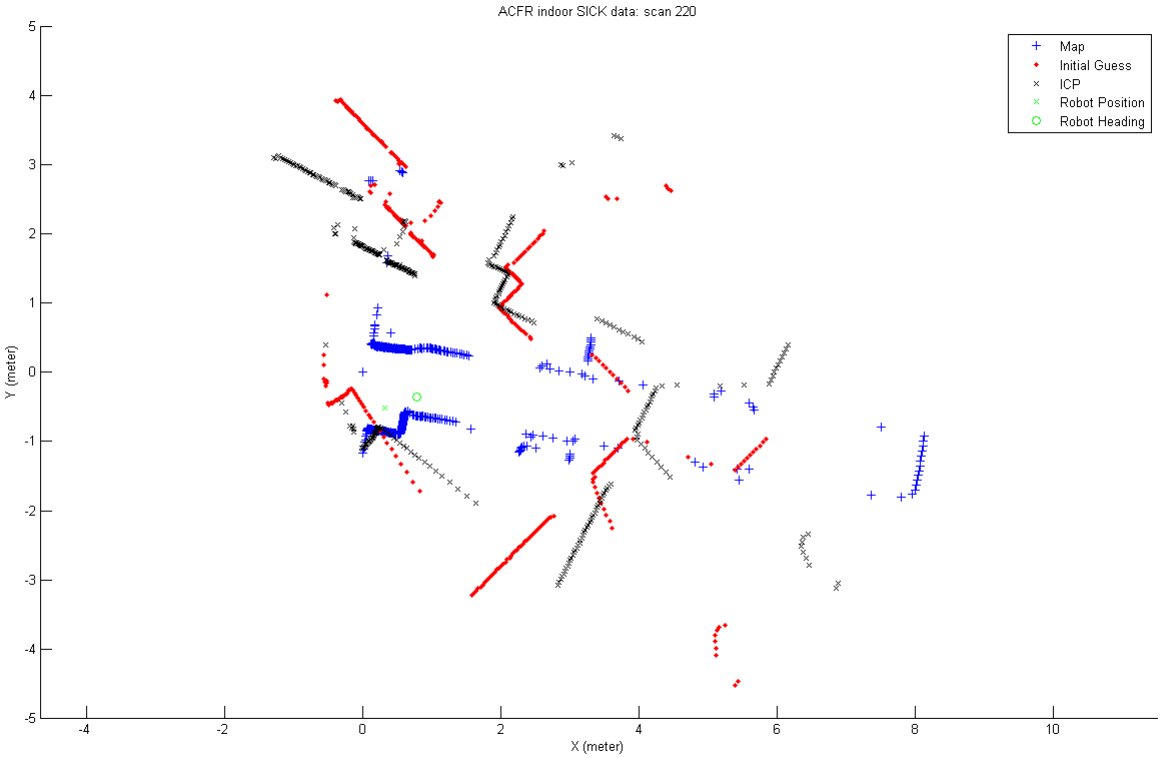

ACFR indoor SICK data: scan 120
Map
Initial Guess
ICP
Robot Position
Robot Heading
Y (meter)
X (meter)
5
4
3
2
1
0
-1
-2
-3
-4
-5
-4
-2
0
2
4
6
8
10

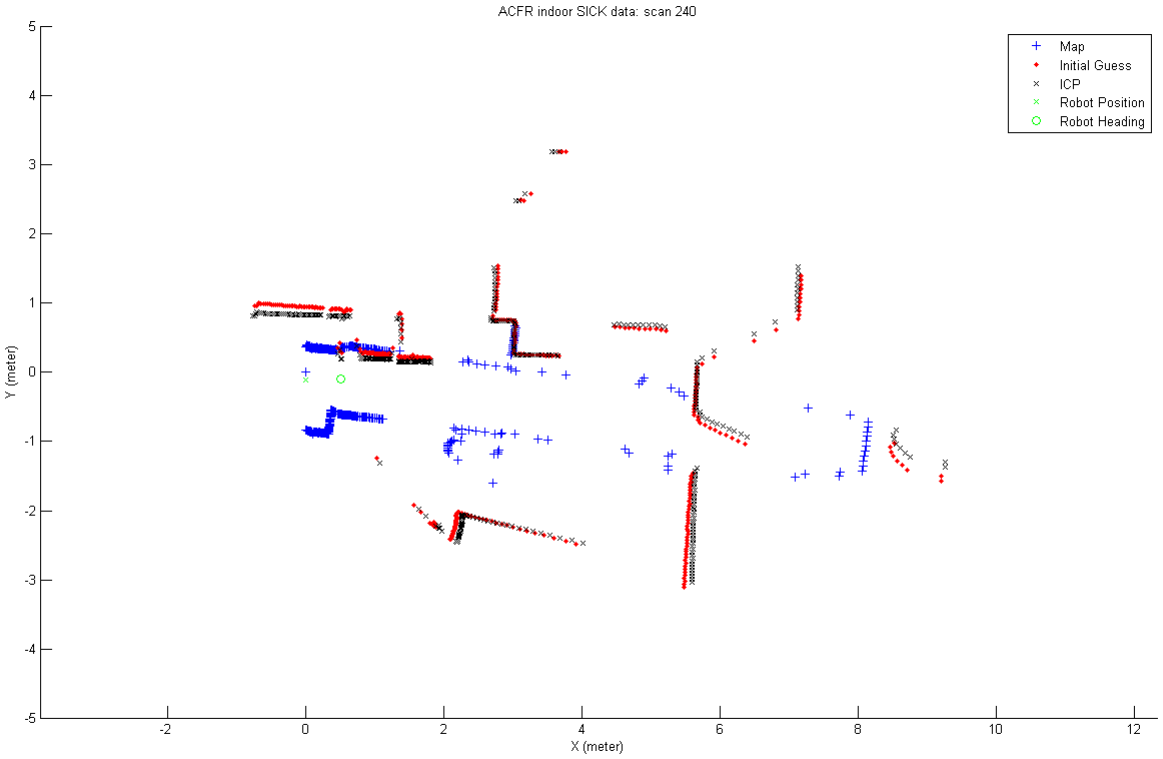

ACFR indoor SICK data: scan 140
Map
Initial Guess
ICP
Robot Position
Robot Heading
Y (meter)
X (meter)
5
4
3
2
1
0
-1
-2
-3
-4
-5
-5
0
5
10

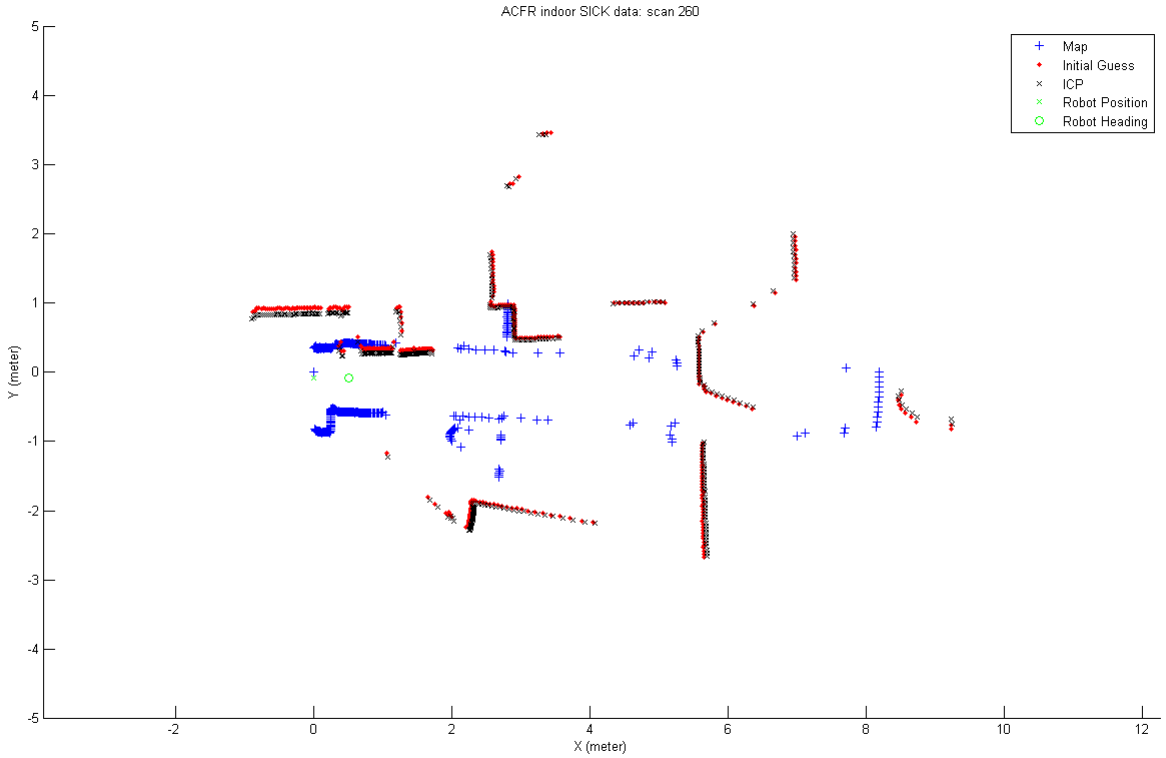

ACFR indoor SICK data: scan 160
Map
Initial Guess
ICP
Robot Position
Robot Heading
Y (meter)
X (meter)
5
4
3
2
1
0
-1
-2
-3
-4
-5
-4
-2
0
2
4
6
8
10

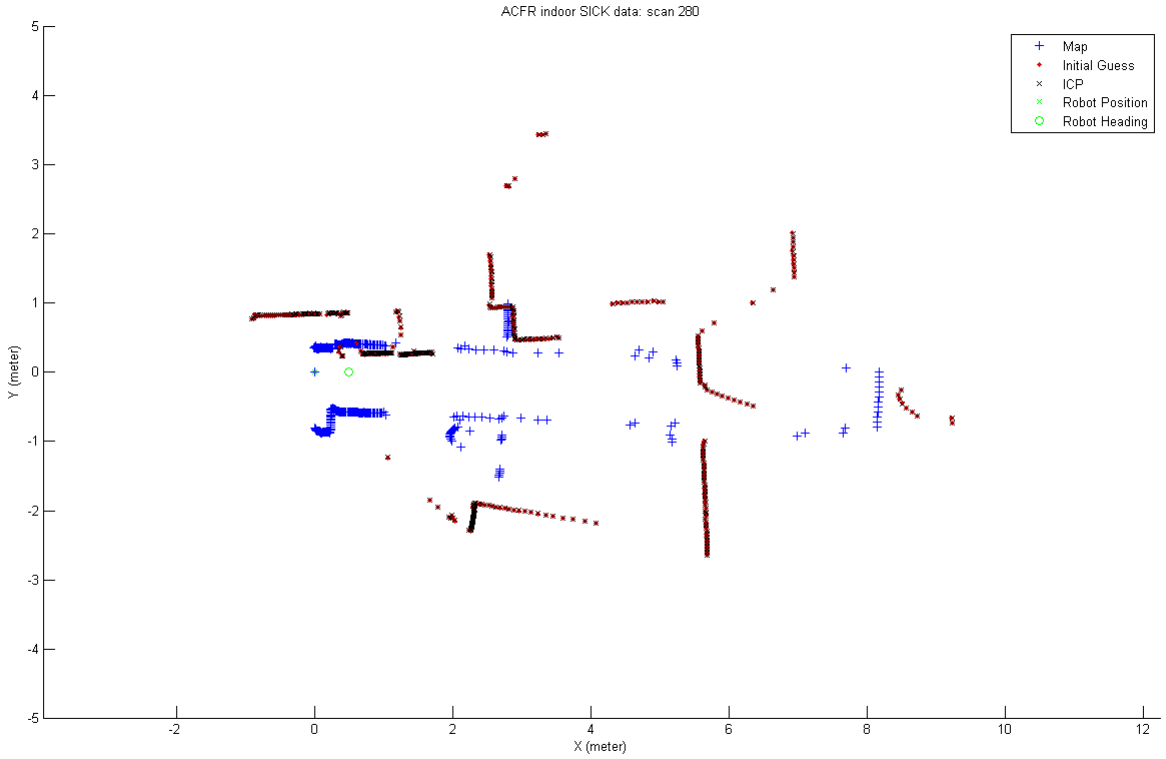

ACFR indoor SICK data: scan 180
Map
Initial Guess
ICP
Robot Position
Robot Heading
Y (meter)
X (meter)
5
4
3
2
1
0
-1
-2
-3
-4
-5
-4
-2
0
2
4
6
8
10

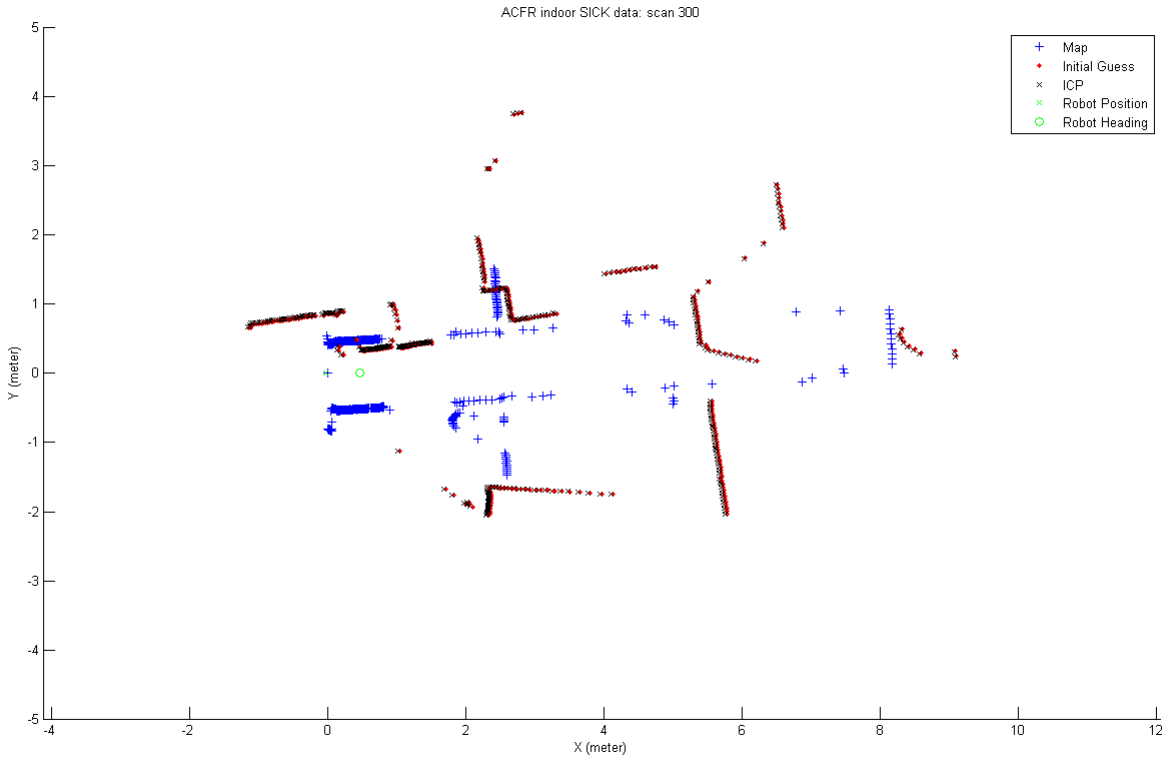

ACFR indoor SICK data: scan 200
Map
Initial Guess
ICP
Robot Position
Robot Heading
Y (meter)
X (meter)
5
4
3
2
1
0
-1
-2
-3
-4
-5
-4
-2
0
2
4
6
8
10


ACFR indoor SICK data: scan 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
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 280
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



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

4.a Modified DH Notation

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.

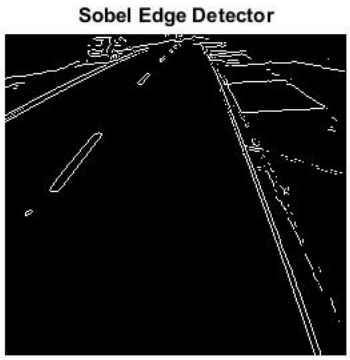


Figure 4: Sobel Edge Detector

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

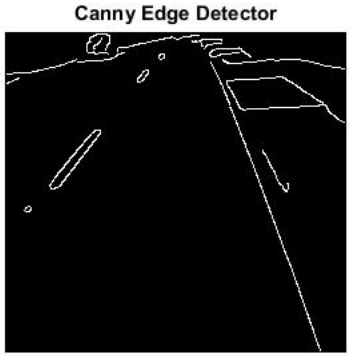


Figure 5: Canny Edge Detector

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.

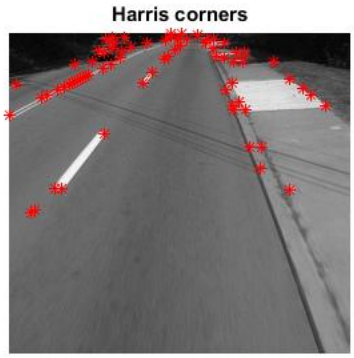


Figure 6: Harris Corners

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.

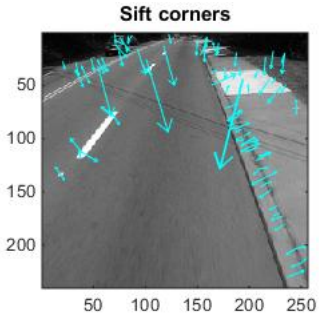


Figure 7: Sift Corners

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.

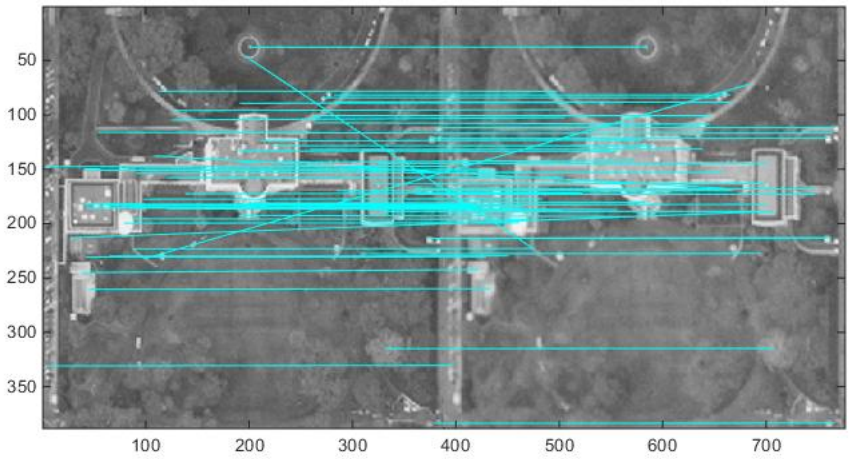


Figure 8: Feature Matching 1

The code we 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:

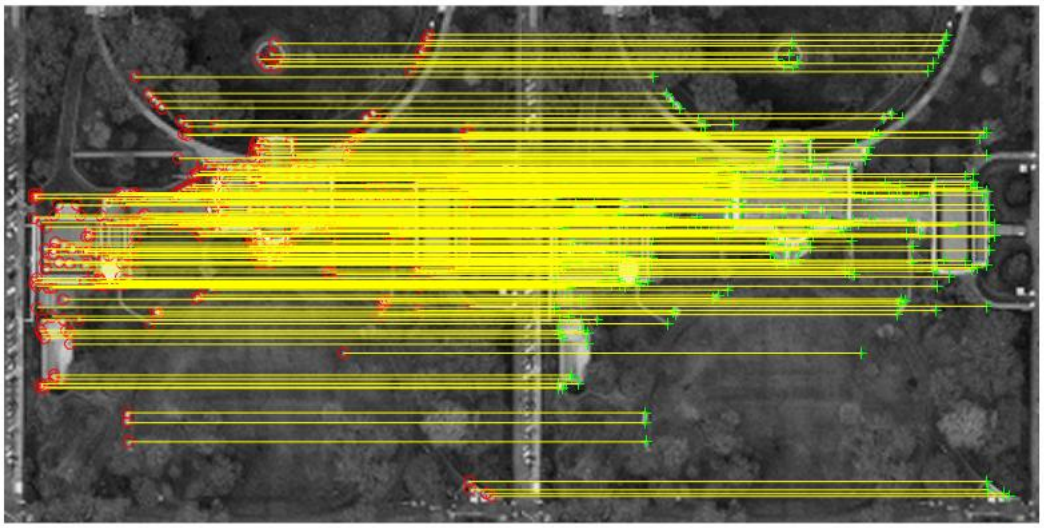


Figure 9: Feature Matching 2

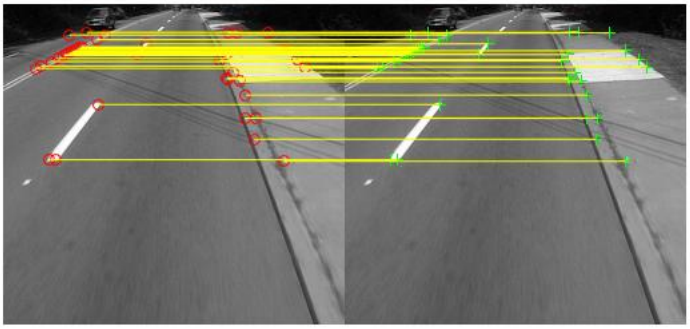


Figure 10: Feature Matching 3

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

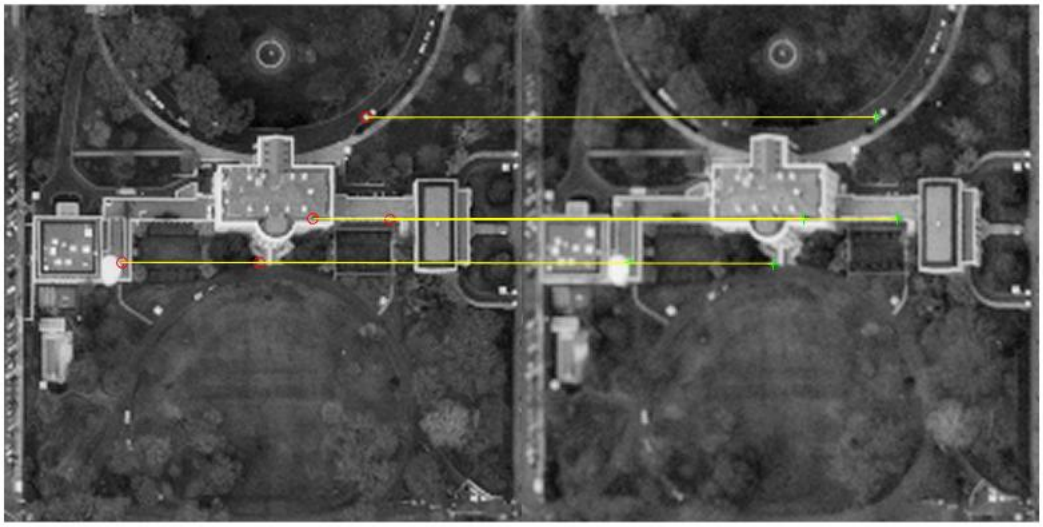


Figure 11: Different Matching 1

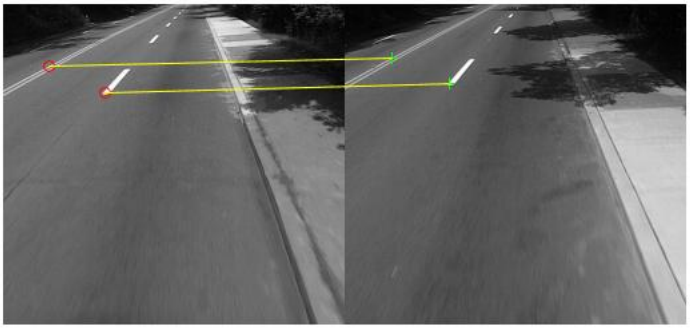


Figure 12: Different Matching 2

Code Listing

See Appendix A [5] for all code used.

6 Question 6

6.a Inverse Kinematics

6.a.i Theoretical Method

6.a.ii Results

6.a.iii Code Listing

See Appendix A [9.5]

6.b Simulation

7 Appendix A

7.1 Question 1

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')
```

```
68 xlabel('X (meter)')
69 ylabel('Y (meter)')
70 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')

```

```
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

```
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
```

```
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 = 100000000000000;
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

```
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
```

```
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      sze = 2*radius+1;                   % Size of mask.
82      mx = ordfilt2(cim,sze^2,ones(sze)); % 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

```
% 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));
```

```
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

```
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