Design of Control Algorithms for Automation of a Full Dimension Continuous Haulage System

Aishwarya Varadhan

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Industrial and Systems Engineering

Dr. Michael P. Deisenroth, Chair
Dr. Robert H. Sturges Jr., Co-Chair
Dr. Charles F. Reinholtz

Volume II – Source Code Manual

April 18, 2000
Blacksburg, Virginia

Keywords: Automation algorithms, Mobile robotics, Sensor based navigation

CONTENTS

Table of contents **i**

List of Figures **ii**

Introduction **1**

APPENDIX I **3**

i Velocity Control Simulation 4

ii Single MBC path following 11

iii Path feasibility analysis 21

iv DMM & DPP Simulations 29

APPENDIX II **42**

i Source code listing for CRC 16 Checksum 43

ii The Multi-unit control interface 44

iii Data sharing Sub-VI's 66

iv RS422 NULL MODEM Pin-out diagram 67

APPENDIX III **68**

i Simple Line-finding – SICK laser scanner 69

ii PMS Line-finding 77

iii Iterative end point line splitting technique 78

iv Simple Motion control & SSS Algorithm 124

Figures

Figure 1: Front Panel of the Multi-unit control VI.....44

Figure 2: Frame 0 of 14 – PMS_MULTI_UNIT_DELAY.VI.....51

Figure 3: Frame 1 of 14 – PMS_MULTI_UNIT_DELAY.VI.....52

Figure 4: Frame 2 of 14 – PMS_MULTI_UNIT_DELAY.VI.....53

Figure 5: Frame 3 of 14 – PMS_MULTI_UNIT_DELAY.VI.....54

Figure 6: Frame 4 of 14 – PMS_MULTI_UNIT_DELAY.VI.....55

Figure 7: Frame 5 of 14 – PMS_MULTI_UNIT_DELAY.VI.....56

Figure 8: Frame 6 of 14 – PMS_MULTI_UNIT_DELAY.VI.....57

Figure 9: Frame 7 of 14 – PMS_MULTI_UNIT_DELAY.VI.....58

Figure 10: Frame 8 of 14 – PMS_MULTI_UNIT_DELAY.VI.....59

Figure 11: Frame 9 of 14 – PMS_MULTI_UNIT_DELAY.VI.....60

Figure 12: Frame 10 of 14 – PMS_MULTI_UNIT_DELAY.VI.....61

Figure 13: Frame 11 of 14 – PMS_MULTI_UNIT_DELAY.VI.....62

Figure 14: Frame 12 of 14 – PMS_MULTI_UNIT_DELAY.VI.....63

Figure 15: Frame 13 of 14 – PMS_MULTI_UNIT_DELAY.VI.....64

Figure 16: Frame 14 of 14 – PMS_MULTI_UNIT_DELAY.VI.....65

Figure 17: LAVT REC Sub-VI Block Diagram……66

Figure 18: LAVT SND Sub-VI Block Diagram……66

Figure 19: NULL MODEM Pin-out for the RS422 PCMCIA Serial card……67

INTRODUCTION

This purpose of this document is to serve as an addendum to the main document describing the research efforts of the LACHS automation. This document is essentially a collection of appendices of source codes developed for this research, and discusses elaborately on each of the different code sections.

The material in this volume is categorized under 3 major sections viz., The simulation source codes in MATLAB®, the LabVIEW® Virtual Instruments, and the Control algorithm codes developed in C. These are grouped under 3 appendix sections in the order described above.

It should be noted that a much greater emphasis in documenting the codes is directed towards describing the LabVIEW® interface and the control algorithm codes. The simulation codes written in MATLAB® are not described in great detail, but included as sectional source code listings. The reason for this omission, or bias in description is attributed to the fact that the simulation codes will not be used extensively by the current or the future LAVT team. Moreover, as the initial stage of simulation and modeling were completed in sufficient and required depth, it is anticipated that there will be no further requirement for extending the simulations developed by the author. The only simulation routines that may be of experimental use are the DMM and the DPP codes. On the other hand, to enable complete and unbiased technology transfer to Long-Airdox personnel, the interface codes in LabVIEW® and the control codes in C will be of great value. It is vital to document these sections due to their direct portability to LA personnel for developments on the full-scale system. Hence, this document maintains a proper balance by placing a higher emphasis on the interface and control codes.

Though the interface and the control code developments spanned long periods of development time, only the codes that are currently being employed by the team are described in this document to avoid repeatability. From the LAVT automation efforts, it is clear that the most recent developments are based off the former research, and hence, illustrating and describing the these codes will suffice this task of source code documentation. The document is organized into

the following sections with each section concentrating on different aspects of the development.

1. Appendix I, lists the MATLAB® simulation codes developed. All the source codes developed for MBC simulations are listed under this section. Though no additional documentation is provided on these codes, the codes are well commented and provides understandability for future developments if a need may exist.
2. Appendix II, discusses in detail, the most recent and functional LabVIEW® VI developed for a multi-unit system. The control interface system is broken down into various frames and each of the independent frames are detailed and illustrated with screen shots.
3. Appendix III, elaborates on the C codes, written as object codes for the Code Interface Nodes. These codes implement the control algorithms described under Volume I of this document. The document follows a standard format for this section. The format is listed here for enhanced clarity.

    i. Introduction – A functional overview of the code and the algorithm.
    ii. Code structure.
    iii. Variable list – A table detailing the code variables.
    iv. Function list – A table illustrating the functions used in the code.
    v. The complete code listing.

The above described format is followed for the codes implementing the control algorithms.

APPENDIX I

This section is dedicated to detailing the development of the simulation routines used in the CHS automation effort. All the simulation routines were written and compiled using MATLAB® Version 5.0, and this chapter, provides a commented listing of all these codes. The following is the organization of this section…

i. Source code listing for the velocity control simulation.
ii. Code listing for the 'Single MBC path following' simulation.
iii. Code listing for the 'Path feasibility analysis' routine.
iv. Source code for the DMM and DPP simulation routines.

Source code reference table

S. No	Simulation	Parent file	Child files
1	Velocity control	simtest.m	None
2	Single MBC path following	lapath.m	None
3	Path feasibility analysis	ellipse.m	None
4	DMM & DPP routines	dynamicpath.m	i. dynamic_clear.m ii. dynamic_interp.m iii. path_param.m

Source code listing begins in the following page.

i. Velocity control simulation – simtest.m

```
%Aish and Mike - LAVT CHS Research - 1998

function mbc_sim()

clear all

%global handle

global timestep; timestep = 0.1; %second
global updaterate; updaterate = 0;
global Mx;              % X Dimension
global My;              % Y Dimension
global Mc;              % Color
global Mh;              % Handle
global vr;              % Right track velocity (fps)
global vl;              % left track velocity (fps)
global va               % average/resultant velocity (vr+vl)/2
global MBC1;

%Generate window for simulation

h = gcf;    %get current figure handle, return the handle of that figure and
            %create one
%same thing as get(0,'CurrentFigure')

set(h,'name','Long-Airdox Continuous Haulage System','numbertitle','off');
clf;
colordef(h,'white');
whitebg('w');
axes('XLim',[-50 400],'YLim',[-50 200])
cla;
axis off;
%axes('XLim',[175 300],'YLim',[-10 115])
set(1,'MenuBar','none')

%create a 'frame' to group the uicontrols

frame=uicontrol('Style', 'frame','Position', [20 35 640
100],'BackgroundColor',[0.2 0.2 0.3]);

%create a on/off plot button for the new path

pltoff=uicontrol('Style', 'PushButton','String','Plot OFF','Position', [500
100 70 25],'callback','');
plton=uicontrol('Style', 'PushButton','String','Plot ON','Position', [580 100
60 25],'callback','');

% create a slider control for track velocities

sliderR=uicontrol('Style', 'slider','Min',0,'Max',5,'SliderStep',[0.1
0.5],'String', 'Track R','Position', [100 50 200 25],'callback','');
text1=uicontrol('Style', 'text','String', 'Right Track =','Position', [310 50
100 25],'BackgroundColor',[0.2 0.4 0.8]);
```

```
sliderL=uicontrol('Style', 'slider', 'Min',0,'Max',5,'SliderStep',[0.1
0.5],'String', 'Track L','Position',[100 100 200 25], 'callback','');
text2=uicontrol('Style', 'text','String', 'Left Track =','Position', [310 100
100 25],'BackgroundColor',[0.2 0.4 0.8]);
set(sliderR,'value',2);
set(sliderL,'value',2);

%get the velocities in a continuous loop while running simulation

vr=get(sliderR,'value')
vl=get(sliderL,'value')

%Declare the dimension of MBC

Mx = [ -12.5;12.5;12.5;-12.5];
My = [ -3; -2;2;3];
Mc = [ 0 0 1];

% Given initial data for each MBC
% each one is row matrix containing
% [x,y,th,d,x0,y0,Speed,Radius,V,B,movebit]
% when movebit = 0, all data are the same except V and B

MBC1 = [248,0,0,12,236,0,0,inf,0,0,1];
MBC2 = [238,0,0,12,236,0,0,inf,0,0,1];

%Create 1 MBC at the origin, all handle of the objects are stored in Mh matrix

Mh(1,1) = patch(Mx,My,Mc);
Mh(2,1) = patch(Mx,My,Mc);

updatescreen;

%Generate Path for the 1st MBC - stline - Arc R1 - stline - Arc R2

R1 = 50;
for t = 1:1:151
   Arc(t,1) = 248+R1*sin((t-1)*pi/200);
   Arc(t,2) = R1*(1-cos((t-1)*pi/200))+50;
end


R2=-30
for t=1:1:100
  Arc(t,3) = 248+R2*sin((t-1)*pi/200);
  Arc(t,4) = R2*(1-cos((t-1)*pi/200))+50;
end

hold on;
plot(Arc(:,1),Arc(:,2));
plot(Arc(:,3),Arc(:,4));

%Move 1st MBC along that path

MBC1 = [1,0,0,12,236,0,2,50,2,0,1];
MBC2 = [1,0,0,12,236,0,0,inf,0,0,1];
```

```
%opening a data file and writing data to the file

fid=fopen('aish.txt','w');
fprintf(fid,'Time\tX Coord\tY Coord\tVelo R\tVelo L\tD\n');

 for i = 1:1:100 %simulate for 50 sec
   % at t = t
   %compute following speeds based on the front MBC movement


   vr=get(sliderR,'value');
   velor=uicontrol('Style', 'text','String',vr,'Position', [400 50 50
25],'BackgroundColor','y','callback','');
   vl=get(sliderL,'value');
   velol=uicontrol('Style', 'text','String',vl,'Position', [400 100 50
25],'BackgroundColor','y','callback','');
   va=sqrt((vr^2)+(vl^2));
   MBC1(1,7)=va;        %incorporate data in slider control
   if vr == vl
      MBC1(1,7) = va;
   else
   MBC1(1,8)= ((vr+vl)/(vr-vl))*5;
   end
   MBC1 = MBC(MBC1);

   %plotting the new course

   x=MBC1(1,1);
   y=MBC1(1,2);
   theta=MBC1(1,3);
   dolly=MBC1(1,4);
   %hold on;

   %check plt button for plot OFF
   %off=get(pltoff,'value');
   %on=get(plton,'value');
   %if on==1
   %   plot(x,y,'k');
   %else if off==0
   %      plot(x,y,'k');
   %   end
   %end

   %displaying the coordinate values

   xcoord=uicontrol('Style', 'text','String',x,'Position', [500 50 50
25],'BackgroundColor','c','callback','');
   ycoord=uicontrol('Style', 'text','String',y,'Position', [600 50 50
25],'BackgroundColor','c','callback','');
   theta=uicontrol('Style', 'text','String',theta,'Position', [700 50 50
25],'BackgroundColor','c','callback','');

   %setting the 'enable' to 'inactive' increases the refresh rate!!

   set(xcoord,'Enable','inactive');
   set(ycoord,'Enable','inactive');
   set(theta,'Enable','inactive');
```

```
   set(velor,'Enable','inactive');
   set(velol,'Enable','inactive');

   updatescreen;

fprintf(fid,'%6.2f\t%6.2f\t%6.2f\t%6.2f\t%6.2f\t%6.2f\n',i*timestep,x,y,vr,vl,
dolly);

   if i==1000
      fclose(fid);
      close all;
   end

end

%close the file
%fclose(fid);

return % end of main

function Handle = Moveto(X,Y,c,h,x,y,theta)

%This function generates the specified MBC or PIG at the desire location,and
%delete the object at old position
%Input arguments : X = Column Matrix contains the x-coordinate of vertices of
%a rectangle
%                  Y = Column Matrix contains the y-coordinate of vertices of
%a rectangle
%                  c = row matrix contains color spec.
%                  h = handle of an object
%                  position ( x,y) of the center of MBC or PIG
%                  theta = Rotation angle(rad) measured from x-axis
%global handle

global updaterate

%Forming Trasformatino Matrix

T = [ cos(theta) -sin(theta)    x;
      sin(theta)  cos(theta)    y;
      0              0          1];

%Forming the vertices matrix
% [ x1 x2 ... ]
% [ y1 y2 ... ]
% [  1  1 ... ]

V = [ X';Y';ones(1,4)];

%Locating new vertices

for i = 1:4
    V(:,i) = T*V(:,i);
end

%Plot
```

```
delete(h);
Handle =patch(V(1,:)',V(2,:)',c);

%pause(updaterate);

return


function newdata = MBC(data)

%This function receives the MBC old data( MBC1,2,...,N), calculates,
%and updates the data for the next timestep based on Speed and Radius.
%Both speed and radius have their sign conventions(refer to the note).
%When R = 0, Speed has to be signed angular velocity ( + for CCW)
%data = [x,y,th,d,x0,y0,Speed,Radius,V,B]

global timestep
global L

%updates position and orientation

if data(1,8) == 0.0  % in case of rotating with R=0
   data(1,3) = data(1,3)+(data(1,7)*data(1,11)*timestep); % theta
   data(1,3) = checktheta(data(1,3));
   data(1,5) = data(1,1)-(L*cos(data(1,3))); %x0
   data(1,6) = data(1,2)-(L*sin(data(1,3))); %y0
   data(1,9) = abs(data(1,7))*L; %V
   data(1,10) = atan2(-data(1,7)*cos(data(1,3)),data(1,7)*sin(data(1,3))); %B
   newdata = data;
   return
elseif data(1,8) == inf % in case of moving straight
   data(1,1) = data(1,1)+(cos(data(1,3))*data(1,7)*data(1,11)*timestep);%x
   data(1,2) = data(1,2)+(sin(data(1,3))*data(1,7)*data(1,11)*timestep);%y
   data(1,9) = abs(data(1,7)); %V
   data(1,10)= atan2(data(1,7)*sin(data(1,3)),data(1,7)*cos(data(1,3))); %B
   newdata = data;
   return
else
%data = [x,y,th,d,x0,y0,Speed,Radius,V,B]
    w = data(1,7)/data(1,8); %compute angular velocity
    data(1,1) = data(1,1)+
data(1,8)*((cos(data(1,3))*sin(w*data(1,11)*timestep))-(sin(data(1,3))*(1-
cos(w*data(1,11)*timestep))));
    data(1,2) = data(1,2)+
data(1,8)*((sin(data(1,3))*sin(w*data(1,11)*timestep))-(cos(data(1,3))*(1-
cos(w*data(1,11)*timestep))));
    data(1,3) = data(1,3)+ (w*data(1,11)*timestep);
    data(1,3) = checktheta(data(1,3));
    newdata = data;
    return
end

function updatescreen()
global MBC1;
global Mx;
global My;
global Mc;
```

```
global Mh;
global updaterate

%update position and orientation of each MBC

Mh(1,1) = Moveto(Mx,My,Mc,Mh(1,1),MBC1(1,1),MBC1(1,2),MBC1(1,3));
pause(updaterate);
return

function d = cal_d(f,r)

% Input arguments: front MBC data(f) and rear MBC data(r)
%                : f and r = [x,y,th,d,x0,y0,Speed,Radius,V,B]
% Return argument: Location of the pivot on dolly(x0,y0) and distance(d)
%                : d = [x0,y0,d];

global l;
global dmax;
global dmin;

m = tan(r(1,3));
a = 1+(m^2); %compute the solution of the quadratic equation
b = -2*(f(1,5)+(m*(f(1,6)-r(1,2)))+((m^2)*r(1,1)));
c = ((f(1,6)-r(1,2)+(m*r(1,1)))^2)+ (f(1,5)^2) - (l^2);
check = (b^2)-(4*a*c);
if check < 0 % no solution and return d = 0
   d = [0,0,0];
   disp('no solution');
   return
else
   equ = [a b c];
   x = roots(equ);
   x1 = x(1,1);
   x2 = x(2,1);
   y1 = m*(x1-r(1,1))+r(1,2);
   y2 = m*(x2-r(1,1))+r(1,2);

   d1 = sqrt((x1-r(1,1))^2+(y1-r(1,2))^2);
   d2 = sqrt((x2-r(1,1))^2+(y2-r(1,2))^2);
   dif1 = abs(r(1,4)-d1);
   dif2 = abs(r(1,4)-d2);

   % choose d

   if ((d1>=dmin)&(d1<=dmax))&((d2>=dmin)&(d2<=dmax))
      if dif1 < dif2
         d = [x1,y1,d1];
         return
      else
         d = [x2,y2,d2];
         return
      end
   elseif ((d1>=dmin)&(d1<=dmax))
      d = [x1,y1,d1];
      return
   elseif ((d2>=dmin)&(d2<=dmax))
      d = [x2,y2,d2];
```

```
      return
   else
      d = [0,0,0]; % have solution, but not in the limited range
      disp('exceed range');
      return
   end
end

function u = followspeed(f,r)

% Input arg. : The data of both front and rear MBC
%            : f and r = [x,y,th,d,x0,y0,Speed,Radius,V,B]
% Return arg.: The following speed of the rear MBC

alpha = atan2((f(1,6)-r(1,2)),(f(1,5)-r(1,1)));
if alpha < 0
   alpha = (2*pi)+alpha;
end
if r(1,3) < 0
   r(1,3) = (2*pi)+r(1,3);
end
if f(1,10) < 0
   f(1,10) = (2*pi)+f(1,10);
end
speed = (cos(alpha+r(1,3)-f(1,10))/cos(alpha))*f(1,9);
u = speed;
return


function th = checktheta(theta)

% input: theta in radian
% output: theta in range [0,2pi);

if theta < 0
   theta = (2*pi)+theta;
end

if theta > (2*pi)
   theta = theta-(2*pi);
end
th = theta;
return
```

ii. Single MBC path following – lapath.m

```
%LA Path planning Algorithm
%Aish - Spring 1999 - March 19, 1999.

%modified to include c-space solution - Aish, July 5, 1999.


%The path created is an "optimal plan" for a 90 degree mine plan
%This plan is made of only arcs and lines and the radius of the arcs the only
param required
%The arcs are stored as a list with an index number and the list is aggregated
to get the path

%Program structure

%1. Define the data points for the 90 deg mine pillars with a 5 entry cut
%2. Use "Patch" to patch the pillars
%3. Draw a 40 feet pig & 28 feet MBC at an arbitrary point
%4. Move this connfiguration around a corner to get the maximal radius around
corners
%5. Validate this path so as to check for 2 MBC's and a pig configuration.

%AIM OF THE PROGRAM

%1. Pick the knot points - representing the direction of travel.
%2. Using the computed arc, lines, plan a path and move along that path
%3. This can be accomplished by defining a function that takes accepts the
data points as arg

%PROGRAM CHARACTERISTICS

%1. The program has the flexibility of changing the arcs defined and observe
the positioning
%   of the pig w.r.t the mine walls.

%PROGRAM EXTENSION

%1. Extend this program to plot various mine patterns
%2. Use a switch case statement to choose between various cases.

%DEFINE THE MINE MAP

global m                %Roadway along X
global n                %Roadway along Y
global i;
global j;
global refreshrate;     %Screen update rate
global xcr;             %Critical distance - X coordinate - X center of arc
global ycr;             %Critical distance - Y coordinate - Y center of arc
global sang;            %Start angle of the turn
global eang;            %End angle of the turn
global pig_len;         %length of the pig
global rad;             %Radius of turn
```

```
refreshrate=0.8;
xcr=140;
ycr=145;
sang=90;
eang=180;
xcr1=80;
ycr1=135;
sang1=0;
eang1=-90;
pig_len=28;
rad=30;

%parameters for path # 2 - depicting the outer boundary

xcr2=137.5;
ycr2=147.5;
sang2=90;
eang2=180;
xcr3=82.5;
ycr3=132.5;
sang3=0;
eang3=-90;


m=0;
n=0;
a=1;
b=1;

zoom on;

for i=0:40:160

   %x(a,:)=[i+m,i+40+m,i+40+m,i+m];

   j=0;
   n=0;

   for j=0:40:160

      x(b,:)=[i+m,i+40+m,i+40+m,i+m];
      y(b,:)=[j+n,j+n,j+40+n,j+40+n];

      n=n+20;
      b=b+1;

   end;

   m=m+20;
   a=a+1;

end;

%PATCH THE MINE PLAN!

for i=1:1:25
   patch(x(i,:),y(i,:),[1,1,0]);
```

```
end


%PIG LENGTH = 40' and MBC LENGTH = 28'

%Draw a pig and MBC....

hold on;

%Plot the initial positions of the MBC and the PIG in St-line Config
%MBC Coordinates - assume as miner!

mx1=[50,50];
my1=[80,108];
mh1=line(mx1,my1);
set(mh1,'LineWidth',[8.0],'Color',[1,0,0]);

px1=[50,50];
py1=[40,80];
ph1=line(px1,py1);
set(ph1,'LineWidth',[4.0],'Color',[0,0,1]);

mx2=[50,50];
my2=[12,40];
mh2=line(mx2,my2);
set(mh2,'LineWidth',[8.0],'Color',[1,0,0]);

hold on;

%The dolly length is constant and the whole config is treated as a rigid link

%for i=5:1:100
%
%   delete(mh1);
%   delete(ph1);
%   delete(mh2);
%
%   mx1=[50,50];
%   my1=[80+i,108+i];
%   mh1=line(mx1,my1);
%   set(mh1,'LineWidth',[8.0],'Color',[1,0,0]);
%
%     px1=[50,50];
%     py1=[40+i,80+i];
%   ph1=line(px1,py1);
%     set(ph1,'LineWidth',[4.0],'Color',[0,0,1]);
%
%   mx2=[50,50];
%     my2=[12+i,40+i];
%     mh2=line(mx2,my2);
%     set(mh2,'LineWidth',[8.0],'Color',[1,0,0]);
%   pause(refreshrate);
%
%end;


%plotting the same configuration in a different area
```

```
mx3=[110,110];
my3=[80,108];
mh3=line(mx3,my3);
set(mh3,'LineWidth',[8.0],'Color',[1,0,0]);

px2=[110,110];
py2=[40,80];
ph2=line(px2,py2);
set(ph2,'LineWidth',[4.0],'Color',[0,0,1]);

mx4=[110,110];
my4=[12,40];
mh4=line(mx4,my4);
set(mh4,'LineWidth',[8.0],'Color',[1,0,0]);


%plot an arc and position the pivot points along  the arc
%the arc is a 90 deg with c (xcr,ycr) and a left turn and rad = 30'

j=1;

for i=sang:1:eang
   xc(j,:)=rad*cos(i*pi/180)+xcr;
   yc(j,:)=rad*sin(i*pi/180)+ycr;
   path(j,:)=[xc(j,:),yc(j,:)];
   hold on;
   j=j+1;
end;

%path1=round(path);

%generate a line from (110,145) to (110,135) in increments if 1

j2=1;
for i=0:1:10
   xl(j2,:)=110;
   yl(j2,:)=145-i;
   li(j2,:)=[xl(j2,:),yl(j2,:)];
   %plot(xl(j2,:),yl(j2,:),'r :');
   hold on;
   j2=j2+1;
end;


%plot another turn;

j1=1;

for i=sang1:-1:eang1
   xcl(j1,:)=rad*cos(i*pi/180)+xcr1;
   ycl(j1,:)=rad*sin(i*pi/180)+ycr1;
   pathl(j1,:)=[xcl(j1,:),ycl(j1,:)];
   %plot(xcl(j1,:),ycl(j1,:),' r :');
   hold on;
   j1=j1+1;
end
```

```
%generate another path to depict the boundary

j2=1;

for i=sang2:1:eang2
   xc2(j2,:)=rad*cos(i*pi/180)+xcr2;
   yc2(j2,:)=rad*sin(i*pi/180)+ycr2;
   circle3(j2,:)=[xc2(j2,:),yc2(j2,:)];
   hold on;
   j2=j2+1;
end;


%generate a line from (107.5,147.5) to (112.5,132.5) in increments if 1

j4=1;
for i=0:1:15
   xl2(j4,:)=107.5+(i/3);
   yl2(j4,:)=147.5-i;
   li2(j4,:)=[xl2(j4,:),yl2(j4,:)];
   %plot(xl(j2,:),yl(j2,:),'r :');
   hold on;
   j4=j4+1;
end;

j3=1;
for i=sang3:-1:eang3
   xc3(j3,:)=rad*cos(i*pi/180)+xcr3;
   yc3(j3,:)=rad*sin(i*pi/180)+ycr3;
   circle4(j3,:)=[xc3(j3,:),yc3(j3,:)];
   hold on;
   j3=j3+1;
end;

hold on;

%union both the paths to get a path

%temp_path=union(path,pathl,'rows');
temp_path=[path;li;pathl];
path1=round(temp_path);

s=size(path1);
s1=s(1,1);

%tempxc=union(xc,xcl,'rows');
%tempyc=union(yc,ycl,'rows');

tempxc=path1(:,1);
tempyc=path1(:,2);



%to move the pig along the path, position one end along the path. To calculate
%the other end, draw a circle(not necessarily full) with this point as the
%center and find the intersection point with the arc (the path). Circle radius
```

```
%is 40'. The point of intersection is stored and then use this to position the
pivot.

   m=1; %index for intersection points

   for k=1:1:s1 %j-1
      ind=1;            %index for circle;
      circle=[0,0];

      for ang=0:1:270
         xcir=pig_len*cos(ang*pi/180)+tempxc(k,:);
         ycir=pig_len*sin(ang*pi/180)+tempyc(k,:);
         circle(ind,:)=[xcir,ycir];
         ind=ind+1;
      end;

      circle1=round(circle);

      int=intersect(circle1,path1,'rows');

      s2=size(int);
      s3=s2(1,1);

      if(s3>0)
         line_end(m,:)=int(1,:);    %filter the first value and use this as
                                    %the other pt
         m=m+1;
      end;

      ind=1;

  end;

  %these are the coordinates of the path and the other end of the pig
  %the first end of the pig is the first coordinate of the path
  %a very small no of points are only obtained. because of 40' length
  %extend the program to position the MBC's too!

  %plotting the continuous path

  plot(xc,yc,'- r');
  hold on;
  plot(xcl,ycl,'- r');
  hold on;
  plot(xl,yl,'- r');

  %plotting the deviate path

  %hold on;
  %plot(xc2,yc2,'-b');
  %hold on;
  %plot(xl2,yl2,'-b');
  %hold on;
  %plot(xc3,yc3,'-b');

  %extract the coordinates of the line  - in this case the pig.
```

```
  pig1x=path1(:,1);
  pig2x=line_end(:,1);

  pig1y=path1(:,2);
  pig2y=line_end(:,2);

  %based on the size of the pig, simulate the motion of the locus of the pig

  pig_s=size(line_end);
  pig_size=pig_s(1,1);

  %plotting the initial pig position.

            lx=[pig1x(1,:),pig2x(1,:)];
      ly=[pig1y(1,:),pig2y(1,:)];

      hnd=line(lx,ly);



%the animation loop - draw and delete the MBC/PIG position

for in=1:1:m-1

     delete(hnd);

     lx=[pig1x(in,:),pig2x(in,:)];
     ly=[pig1y(in,:),pig2y(in,:)];

     hnd=line(lx,ly);
     %set(hnd,'LineWidth',[10.0],'Color',[0.3,0.3,0.2],'marker','o');
     set(hnd,'LineWidth',[4.0],'Color',[0.3,0.3,0.2]);

     pause(refreshrate);
     %drawnow;

end;


  %Function to position the MBC & PIG pivots along the 'optimal path'.

hold on;
zoom on;


%extend the program to position the center of the MBC along the path
%to find out the path deviation, start at a random point,
%then sweep the pig about that point to avoid collision with walls

%solve as a C-Space problem
%position the pivot along one end and sweep 360 deg...
%find the collision angle with any of the walls
%store that angle and plot it w.r.t every point on the path.

%create the lines representing the mine walls - The OBSTACLES

%generate first two lines
```

```
index=1;

for i=0:1:40
   line_x1(index,:)=120+i;
   line_y1(index,:)=180;
   line_1(index,:)=[line_x1(index,:),line_y1(index,:)];

   line_x2(index,:)=120+i;
   line_y2(index,:)=160;
   line_2(index,:)=[line_x2(index,:),line_y2(index,:)];

   index=index+1;
end;

%generate the 2 vertical lines

index1=1;

for i=0:1:40
   line_x3(index1,:)=120;
   line_y3(index1,:)=160-i;
   line_3(index1,:)=[line_x3(index1,:),line_y3(index1,:)];

   line_x4(index1,:)=100;
   line_y4(index1,:)=160-i;
   line_4(index1,:)=[line_x4(index1,:),line_y4(index1,:)];

   index1=index1+1;
end;

%generate the other two horizontal lines

index2=1;

for i=0:1:40
   line_x5(index2,:)=100-i;
   line_y5(index2,:)=120;
   line_5(index2,:)=[line_x5(index2,:),line_y5(index2,:)];

   line_x6(index2,:)=100-i;
   line_y6(index2,:)=100;
   line_6(index2,:)=[line_x6(index2,:),line_y6(index2,:)];

   index2=index2+1;
end;


%union all obstacles into one single obstacle

temp_line=[line_1;line_2;line_3;line_4;line_5;line_6];
line_obs=round(temp_line);

line_s=size(line_obs);
line_size = line_s(1,1);

line_x=line_obs(:,1);
```

```
line_y=line_obs(:,2);              %not required for c-space generation
line_size;


%obstacle avoidance routine

%1. create a 28' link on the first point on the path
%2. sweep over a full circle and store colliding points
%3. this represents the max angle of pivot at any point

 no=1;                          %index for intersection points

for k=1:10:s1 %j-1              %the pig pivot positioning

    ent1=0;                     %index to initiate break....
                                %restricts to only one direction

     for ang=0:36:360

        ind=1;                  %index for pig/mbc;

         for len=0:7:pig_len
            pig_x=len*cos(ang*pi/180)+tempxc(k,:);       %add the x coordinate
                                                         %- tempxc
            pig_y=len*sin(ang*pi/180)+tempyc(k,:);       %add the y coordinate
                                                         %- tempyc
            pig(ind,:)=[pig_x,pig_y];
            ind=ind+1;

            %check for collision

            pig1=round(pig);

            int_obs=intersect(pig1,line_obs,'rows');
            siz=size(int_obs);
            act_siz=siz(1,1);

            if(act_siz>0)
             %  if (ent1==0)                    %only the very first time for
                                                %that pig length
                  theta(no,:)=ang;              %store the theta that collides
                  index_path(no,:)=k;           %gives the index of the point
                                                %on the path.
                  no=no+1;
                  ent1=1;
              % end;                          %end of inner if
            end;                              %end of act_siz if

         end;                                 %end of pig length for loop

      end;                                    %end of for (angle sweep)

 end;                                         %end of for (path for)

 %PROBLEM!!!

 %even though no = 153...size(theta)=1912!!!..
```

```
%SOLUTION- use only until no = 153..

%rewrite into the data array of theta and index_path
%also extract the indices into a data file....

for i=1:1:(no-1)
   theta_new(i,:)=theta(i,:);              %the critical angle
   index_path_new(i,:)=index_path(i,:);    %the index of the path
   act_ind = index_path_new(i,:);          %extract the collision index
   obs_path(i,:) = path1(act_ind,:);       %the critical points on the path
   %also extract the (x,y) pair on the path - write to a data file.....
end;


%things to do...

%1. store the path index where theta collides
%2. plot theta vs the path index or coordinate axis

%after havng got the theta and index_path, on the main plot,
%plot * where theta is present at that index
%this represents the most critical region to the MBC
%can plot an * on the mine wall - showing the critical region

%Method 2 - Ref Mohan Sainani - Accuracy Ellipsoids for positioning error

%THE TECHNIQUE

%Assuming the positioning errors are N(mean, sig), and assuming a
%probability of success, compute the accuracy ellipses at every point
%on the path for the front most pivot of link 1..the boundary of all the
%major axes represents the max boundary constraint along that path.
%can consider just one or two links....use IK to find the joint angles
%assuming that the pivots are exactly positioned on the path.

%THE ALGORITHM

%1. Consider for ex, a 2 link manipulator...
%2. X = L1*C1 + L2*C12   ; Y = L1*S1 + L2*S12
%3. Find the partial of this end eff pos w.r.t jt angles - The JACOBIAN
%4. Assume a SD for the angle errors, and probably the dolly distance..
%5. Assume a p(1) for positioning - say a 3 sig limit
%6. Calculate II order moments of J
%7. Compute corr coeff from step 6.
%8. Compute the major/minor axis of the ellispe
%9. Plot the lines (major axis) on the path.... - RESULT!!!


%THE ACTUAL CODE

%INVERSE KINEMATICS - Assuming a single MBC - length of 28', perform IK to
%find theta the ref axis is at the rear end of sweep - pivot point (x,y) is
%usual - x is always neg for this case....
```

iii. Path feasibility analysis – ellipse.m

```
                    %LA Path planning Algorithm - Path deviation

%ACCURACY ELLIPSE METHOD - Ref : Mohan's  research on Dexterity
%Tech Pubs    : On two methods of determining the ellipses and ellipsoids of
%               positioning accuracy of robot manipulators - Archive of Appl
%               mechanics, Springer, 1991
%AUTHOR       : Aish - Spring 1999 - July 9, 1999.
%SUPPORT      : LAVT CHS Automation Team - Mike & Bruce
%PI's         : Dr. Mike Deisenroth & Dr. Bob Sturges
%MODIFIED     : August 5, 1999 to include the Inverse Kinematics of two link
               %system

%EXTEND FOR A TWO LINK SYSTEM..IMPORTANT!!! - EXTSNSIONS TO WORK

%1. Extend for a single link - theoretical proof
%2. Extend work for error in slider positioning - dolly distance
%3. For slider, can consider error as N(0.5, Sigma) - dolly center is mean

%THE TECHNIQUE

 %Assuming the positioning errors are N(mean, sig), and assuming a
 %probability of success, compute the accuracy ellipses at every point
 %on the path for the front most pivot of link 1..the boundary of all the
 %major axes represents the max boundary constraint along that path.
 %can consider just one or two links....use IK to find the joint angles
 %assuming that the pivots are exactly positioned on the path.

 %THE ALGORITHM

 %1. Consider for ex, a 2 link manipulator...
 %2. X = L1*C1 + L2*C12   ; Y = L1*S1 + L2*S12
 %3. Find the partial of this end eff pos w.r.t jt angles - The JACOBIAN
 %4. Assume a SD for the angle errors, and probably the dolly distance..
 %5. Assume a p(1) for positioning - say a 3 sig limit
 %6. Calculate II order moments of J
 %7. Compute corr coeff from step 6.
 %8. Compute the major/minor axis of the ellispe
 %9. Plot the lines (major axis) on the path.... - RESULT!!!


%PROGRAM EXTENSION

%1. Extend this program to plot various mine patterns
%2. Use a switch case statement to choose between various cases.

%Conclusions from this techniques...

%1. Does not work for a mobile robot - needs reshifting of coord frame at
%every time step.
%2. Not efficient for a one-link manipulator - as IK solution is just Cos(th)
%and Sin(th).
```

```
global m                %Roadway along X
global n                %Roadway along Y
global i;
global j;
global refreshrate;     %Screen update rate
global xcr;             %Critical distance - X coordinate - X center of arc
global ycr;             %Critical distance - Y coordinate - Y center of arc
global sang;            %Start angle of the turn
global eang;            %End angle of the turn
global pig_len;         %length of the pig
global rad;             %Radius of turn
global prob;            %the probability of positioning - p of success
global sig_th;          %the standard deviation - error in angle positioning -
                        %in RADIANS!!!
global mine_st;         %mine start coordinate
global mine_en;         %mine end coordinate

refreshrate=0.3;
xcr=140;
ycr=145;
sang=90;
eang=180;
xcr1=80;
ycr1=135;
sang1=0;
eang1=-90;
pig_len=28;
rad=30;
prob = 0.85;          %with a 3 sig limit - cannot be one - as (1-prob) = 0
sig_th = 0.0872;      %assuming a 5 deg theta error
mine_st = 0;
mine_en = 160;

m=0;
n=0;
a=1;
b=1;

zoom on;

%DEFINE THE MINE MAP

for i=mine_st:40:mine_en
   j=0;
   n=0;
   for j=mine_st:40:mine_en
      x(b,:)=[i+m,i+40+m,i+40+m,i+m];
      y(b,:)=[j+n,j+n,j+40+n,j+40+n];
      n=n+20;
      b=b+1;
   end;
   m=m+20;
   a=a+1;
end;

%PATCH THE MINE PLAN!
```

```
for i=1:1:25
   patch(x(i,:),y(i,:),[0.1,0.3,0.5]);
end


%PIG LENGTH = 40' and MBC LENGTH = 28'

%Draw a pig and MBC....

hold on;


%plot an arc and position the pivot points along  the arc
%the arc is a 90 deg with c (xcr,ycr) and a left turn and rad = 30'

j=1;

for i=sang:1:eang
   xc(j,:)=rad*cos(i*pi/180)+xcr;
   yc(j,:)=rad*sin(i*pi/180)+ycr;
   path(j,:)=[xc(j,:),yc(j,:)];
   hold on;
   j=j+1;
end;

%path1=round(path);

%generate a line from (110,145) to (110,135) in increments if 1

j2=1;

for i=0:1:10
   xl(j2,:)=110;
   yl(j2,:)=145-i;
   li(j2,:)=[xl(j2,:),yl(j2,:)];
   %plot(xl(j2,:),yl(j2,:),'r :');
   hold on;
   j2=j2+1;
end;


%plot another turn;

j1=1;

for i=sang1:-1:eang1
   xcl(j1,:)=rad*cos(i*pi/180)+xcr1;
   ycl(j1,:)=rad*sin(i*pi/180)+ycr1;
   pathl(j1,:)=[xcl(j1,:),ycl(j1,:)];
   %plot(xcl(j1,:),ycl(j1,:),' r :');
   hold on;
   j1=j1+1;
end
%union both the paths to get a path

%temp_path=union(path,pathl,'rows');
```

```
temp_path=[path;li;pathl];
path1=round(temp_path);

s=size(path1);
s1=s(1,1);

%tempxc=union(xc,xcl,'rows');
%tempyc=union(yc,ycl,'rows');

tempxc=path1(:,1);
tempyc=path1(:,2);



%to move the pig along the path, position one end along the path. To calculate
%the other end, draw a circle(not necessarily full) with this point as the %
%center and find the intersection point with the arc (the path). Circle radius
%is 40'. The point of intersection is stored and then use this to position the
%pivot.

   m=1;                %index for intersection points

   for k=1:1:s1 %j-1
      ind=1;           %index for circle;
      circle=[0,0];

      for ang=0:1:270
         xcir=pig_len*cos(ang*pi/180)+tempxc(k,:);
         ycir=pig_len*sin(ang*pi/180)+tempyc(k,:);
         circle(ind,:)=[xcir,ycir];
         ind=ind+1;
      end;

      circle1=round(circle);

      int=intersect(circle1,path1,'rows');

      s2=size(int);
      s3=s2(1,1);

      if(s3>0)
         line_end(m,:)=int(1,:);    %filter the first value and use this as
                                    %the other pt
         m=m+1;
      end;

      ind=1;

  end;

  %these are the coordinates of the path and the other end of the pig
  %the first end of the pig is the first coordinate of the path
  %a very small no of points are only obtained. because of 40' length
  %extend the program to position the MBC's too!

  %plotting the continuous path
```

```
  plot(xc,yc,'- r');
  hold on;
  plot(xcl,ycl,'- r');
  hold on;
  plot(xl,yl,'- r');

  %extract the coordinates of the line  - in this case the pig.

  pig1x=path1(:,1);
  pig2x=line_end(:,1);

  pig1y=path1(:,2);
  pig2y=line_end(:,2);

  %based on the size of the pig, simulate the motion of the locus of the pig

  pig_s=size(line_end);
  pig_size=pig_s(1,1);

  %plotting the initial pig position.

      lx=[pig1x(1,:),pig2x(1,:)];
      ly=[pig1y(1,:),pig2y(1,:)];

      hnd=line(lx,ly);



%the animation loop - draw and delete the MBC/PIG position

for in=1:1:m-1

     %delete(hnd);

     %lx=[pig1x(in,:),pig2x(in,:)];
     %ly=[pig1y(in,:),pig2y(in,:)];

     %hnd=line(lx,ly);
     %set(hnd,'LineWidth',[1.0],'Color',[0.3,0.3,0.2],'marker','o');

     %pause(refreshrate);
     %drawnow;

      %Position the Axis Origin at the rear end of the pig
      %though both local (x,y) axis are neg w.r.t the global figure ref, no
      %problem
      %just need the abs value to find the angle at that point

     x_st = pig1x(in,:);
     y_st = pig1y(in,:);

     x_en = pig2x(in,:);
     y_en = pig2y(in,:);

     delta_x = x_st - x_en;
     delta_y = y_st - y_en;
```

```
      %delta_x and delta_y are the (x,y) position of the end effector w.r.t
      %base

     %change the sign of delta_x and delta_y

     delta_x = 0-delta_x;
     delta_y = 0-delta_y;

    %find the link angle.....

    %lnk_th = atan2(delta_y,delta_x);

     lnk_th = atan(delta_y/delta_x);

     lnk_th1(in,:) = atan(delta_y/delta_x)*180/pi;

     lnk_th_D = lnk_th*180/pi;        %in degrees

     %the length of the link is the same - length of the pig...

     x_pos = pig_len*cos(lnk_th);       %the end effector position
     y_pos = pig_len*sin(lnk_th);

     %get the partial of the end eff pos w.r.t the joint angles - JACOBIAN

     par_x = -pig_len*sin(lnk_th);
     par_y = pig_len*cos(lnk_th);

     %compute the elements of the matrix of second order moments...

     lambda_xx = (par_x*par_x)*(sig_th*sig_th);
     lambda_yy = (par_y*par_y)*(sig_th*sig_th);
     lambda_xy = (par_x*par_y)*(sig_th*sig_th);

     %compute the correlation coefficient

     corr_coeff(in,:) = lambda_xy/(sqrt(lambda_xx*lambda_yy));


     %compute the chi^2 goodness of fit test...

     %compute the major and minor axis of the ellipse..

     ellipse_D = lambda_xx + lambda_yy;
     mod_k = (lambda_xx*lambda_yy) - (lambda_xy*lambda_xy);

     ellipse_a_xsq = -log(1-prob)*(ellipse_D + sqrt((ellipse_D*ellipse_D)-
4*mod_k));
     ellipse_b_xsq = -log(1-prob)*(ellipse_D - sqrt((ellipse_D*ellipse_D)-
4*mod_k));

     ellipse_a(in,:) = sqrt(ellipse_a_xsq);
     ellipse_b(in,:) = sqrt(ellipse_b_xsq);
     ellipse_th(in,:) = atan(b/a)*180/pi;

end;
```

```
%function to plot the ellipse's.....

%use tempxc, tempyc, ellipse_a, ellipse_b to plot a line as b is ~0
%use the same axis coordinates.....plot a line (x,y) at angle ellipse_th

%MAIN PATH DEVIATION PLOTTING ROUTINE...

for la = 1:1:m-1

   %the center of the line...
   x_pos_dev = tempxc(la,:);
   y_pos_dev = tempyc(la,:);

   dev_ang = ellipse_th(la,:);     %the angle of the line

   line_len = ellipse_a(la,:);     %the length of the deviation - semi major
                                   %axis

   x_dev_end = -line_len*cos(dev_ang*pi/180) + x_pos_dev;
   %x coord at both ends
   x_dev_str = line_len*cos(dev_ang*pi/180) + x_pos_dev;

   y_dev_end = -line_len*sin(dev_ang*pi/180)+ y_pos_dev;
   %y coord at both ends
   y_dev_str = line_len*sin(dev_ang*pi/180) + y_pos_dev;

   line_x = [x_dev_end, x_dev_str];
   line_y = [y_dev_end, y_dev_str];

   handle=line(line_x, line_y);

   %control the line properties

   set(handle,'LineWidth',[1.0],'Color',[0,1,0]);


end;

%plot the path again

plot(xc,yc,'- r');
hold on;
plot(xcl,ycl,'- r');
hold on;
plot(xl,yl,'- r');

hold on;
zoom on;

%Label the graph with the deviation path values...

text(120,110,'Prob = ');
text(130,110,num2str(prob));
text(120,106,'a     = ');
text(130,106,num2str(ellipse_a(1,:)));
text(120,102,'b     = ');
text(130,102,num2str(ellipse_b(1,:)));
```

```
%send the plot to printer..

opt = input('Send plot to printer ?(1/0):');

if opt==1
   orient landscape
   print;
end;


%THE ACTUAL CODE - for IK

%INVERSE KINEMATICS - Assuming a single MBC - length of 28', perform IK to
%find theta The ref axis is at the rear end of sweep - pivot point (x,y) is
%usual - x is always neg for this case....
```

iv. DMM and DPP simulations - dynamicpath.m
- **dynamic_clear.m**
- **dynamic_interp.m**
- **path_param.m**

dynamicpath.m

```
% Project: LA Path planning Algorithm
% Author : Aish - Spring 00, Feb 13, 2000, NEB 197
% Theory : LA Path planning using CAGD and mine geometry...
% This is the parent file....callback fn calls the path_param file...

% Program structure....

% 1. Input mine data - pillar widths, roadways and angle...(iterate later)..
% 2. Choose an arbitrary point and fit a parametric cubic..(or) a cubic
bezier...
% 3. For a param cubic - need 2 points (X,Y) and two tangents
%    For a cubic bezier - need 4 points - Start, End and two control points
% 4. Plot this curve and check for feasibility using the PIG and C-Space
analysis
% 5. If feasible, store the params, else iterate for various pts until
feasibility is achieved
% 6. Change mine params and find a new curve or modify curve dynamically!
% 7. As a result, establish diff curves (that do not change) for a mine param
range..


% C-SPACE Analysis..

% 1. Use a 40' PIG and allow for changing the PIG lengths...
% 2. Include the wall fillets...say a line tgt to the fillet will simulate a
fillet...


% This program simulates the LA Dynamic Path planning approach developed by
% Aish & Mike - PPP = Perfect Path Planning

% The program starts off with a simple 90 deg mine plan and plots a 'S' curve
- the worst case
% The simulation runs thro various mine plan widths and mine angles and
depicts the
% generation of a curve dynamically.
% The program assumes that the trail MBC has a local map of a 3 MBC system
% from the VTMMB algorithm - VT Mine Map Building!


% Variable declaration..

global PXW;     % Pillar width in the X Direction
global PYW;     % Pillar width in the Y Direction
global XW;      % Roadway in the X Direction
global YW;      % Roadway in the Y Direction
global M_ang;   % Mine angle - the included angle between the pillars..
global RAD;     % Conversion from deg to rad = angle*pi/180
```

```
RAD = pi/180;

PXW = 40;
PYW = 40;
M_ang = 90;
XW  = 20;
YW  = 20;


%DEFINE THE MINE MAP

h=gcf;
hold on;
set(h,'name','LAVT Curve Planning - Aish','numbertitle','off');
set(h,'Position',[0 28 800 534]);
clf;

% Screen widgets..

pillar_width = uicontrol('Style', 'popup'
,'String','Width|30|40|50|60|70','Position',[125,500,60,25],'callback','path_p
aram');
x_width      = uicontrol('Style', 'popup' ,'String','X
Dist|10|15|16|17|18|19|20|22|25','Position',[200,500,60,25],'callback','path_p
aram');
y_width      = uicontrol('Style', 'popup'
,'String','Y_Dist|10|15|16|17|18|19|20|22|25','Position',[275,500,60,25],'call
back','path_param');
Ang          = uicontrol('Style', 'popup'
,'String','Angle|30|60|75|90|120|150','Position',[350,500,60,25],'callback','p
ath_param');
gridbtn   = uicontrol('Style', 'PushButton','String','GRID','Position', [500
503 60 25],'callback','grid');
interpbtn = uicontrol('Style', 'PushButton','String','INTERP','Position', [575
503 60 25],'callback','dynamic_interp');
clearbtn  = uicontrol('Style', 'PushButton','String','CLEAR','Position', [650
503 60 25],'callback','dynamic_clear');
dirbtn    = uicontrol('Style', 'popup'   ,'String','Dir|Right|Left|straight|R-
L|L-R','Position',[425,500,60,25],'callback','path_param');
%nature    = uicontrol('Style', 'popup'   ,'String','Curve|Cen-Off|Off-
Cen','Position',[725,500,60,25],'callback','zoom');

% Default dir of curve is towards the right....

% define the line parameters--
% LL - Lower Left, LR - Lower Right and similair for UL and UR..

% The Lower Left Section...

LLX1 = [0,0+PXW];        % The lower left corner - the two X coord of the
                         % bottom line
LLY1 = [0,0];
LL1  = line(LLX1, LLY1);
LLX2 = [0,PXW*cos(M_ang*RAD)];
LLY2 = [0,PXW*sin(M_ang*RAD)];
LL2  = line(LLX2, LLY2);
```

```
LLX3 = [PXW*cos(M_ang*RAD),PXW*cos(M_ang*RAD)+PXW];
LLY3 = [PXW*sin(M_ang*RAD),PXW*sin(M_ang*RAD)];
LL3  = line(LLX3, LLY3);
LLX4 = [0+PXW,PXW*cos(M_ang*RAD)+PXW];
LLY4 = [0,PXW*sin(M_ang*RAD)];
LL4  = line(LLX4, LLY4);

% The lower right section.. = LL + PXW + XW

LRX1 = [0+XW+PXW,0+PXW+XW+PXW];     % Add the roadway width + PXW to the
                                    % lower left corner....
LRY1 = [0,0];
LR1  = line(LRX1, LRY1);
LRX2 = [0+XW+PXW,PXW*cos(M_ang*RAD)+PXW+XW];
LRY2 = [0,PXW*sin(M_ang*RAD)];
LR2  = line(LRX2, LRY2);
LRX3 = [PXW*cos(M_ang*RAD)+XW+PXW,PXW*cos(M_ang*RAD)+PXW+XW+PXW];
LRY3 = [PXW*sin(M_ang*RAD),PXW*sin(M_ang*RAD)];
LR3  = line(LRX3, LRY3);
LRX4 = [0+PXW+XW+PXW,PXW*cos(M_ang*RAD)+PXW+XW+PXW];
LRY4 = [0,PXW*sin(M_ang*RAD)];
LR4  = line(LRX4, LRY4);

% The upper left corner - LL + PYW + YW

ULX1 = [PXW*cos(M_ang*RAD),PXW*cos(M_ang*RAD)+PXW];        % The lower left
corner - the two X coord of the bottom line
ULY1 = [PXW*sin(M_ang*RAD)+YW,PXW*sin(M_ang*RAD)+YW];
UL1  = line(ULX1, ULY1);
ULX2 = [PXW*cos(M_ang*RAD),PXW*cos(M_ang*RAD)+PXW*cos(M_ang*RAD)];
ULY2 = [PXW*sin(M_ang*RAD)+YW,PXW*sin(M_ang*RAD)+PXW*sin(M_ang*RAD)+YW];
UL2  = line(ULX2, ULY2);
ULX3 =
[PXW*cos(M_ang*RAD)+PXW*cos(M_ang*RAD),PXW*cos(M_ang*RAD)+PXW*cos(M_ang*RAD)+P
XW];
ULY3 =
[PXW*sin(M_ang*RAD)+PXW*sin(M_ang*RAD)+YW,PXW*sin(M_ang*RAD)+PXW*sin(M_ang*RAD
)+YW];
UL3  = line(ULX3, ULY3);
ULX4 = [PXW*cos(M_ang*RAD)+PXW*cos(M_ang*RAD)+PXW,PXW*cos(M_ang*RAD)+PXW];
ULY4 = [PXW*sin(M_ang*RAD)+PXW*sin(M_ang*RAD)+YW,PXW*sin(M_ang*RAD)+YW];
UL4  = line(ULX4, ULY4);

% The upper right corner - UL + PXW + XW

URX1 = [PXW*cos(M_ang*RAD)+XW+PXW,PXW*cos(M_ang*RAD)+PXW+XW+PXW];        % The
lower left corner - the two X coord of the bottom line
URY1 = [PXW*sin(M_ang*RAD)+YW,PXW*sin(M_ang*RAD)+YW];
UR1  = line(URX1, URY1);
URX2 =
[PXW*cos(M_ang*RAD)+XW+PXW,PXW*cos(M_ang*RAD)+PXW*cos(M_ang*RAD)+XW+PXW];
URY2 = [PXW*sin(M_ang*RAD)+YW,PXW*sin(M_ang*RAD)+PXW*sin(M_ang*RAD)+YW];
UR2  = line(URX2, URY2);
URX3 =
[PXW*cos(M_ang*RAD)+PXW*cos(M_ang*RAD)+XW+PXW,PXW*cos(M_ang*RAD)+PXW*cos(M_ang
*RAD)+PXW+XW+PXW];
```

```
URY3 =
[PXW*sin(M_ang*RAD)+PXW*sin(M_ang*RAD)+YW,PXW*sin(M_ang*RAD)+PXW*sin(M_ang*RAD
)+YW];
UR3  = line(URX3, URY3);
URX4 =
[PXW*cos(M_ang*RAD)+PXW*cos(M_ang*RAD)+PXW+XW+PXW,PXW*cos(M_ang*RAD)+PXW+XW+PX
W];
URY4 = [PXW*sin(M_ang*RAD)+PXW*sin(M_ang*RAD)+YW,PXW*sin(M_ang*RAD)+YW];
UR4  = line(URX4, URY4);

% Creating two more mine blocks - to facilitate S turns
% Say call that L2 and U2

% Creating the next left block...LR + PXW + XW

L2X1 = [0+XW+PXW+XW+PXW,0+PXW+XW+PXW+XW+PXW];         % Add the roadway width
+ PXW to the lower left corner....
L2Y1 = [0,0];
L21  = line(L2X1, L2Y1);
L2X2 = [0+XW+PXW+XW+PXW,PXW*cos(M_ang*RAD)+PXW+XW+XW+PXW];
L2Y2 = [0,PXW*sin(M_ang*RAD)];
L22  = line(L2X2, L2Y2);
L2X3 =
[PXW*cos(M_ang*RAD)+XW+PXW+XW+PXW,PXW*cos(M_ang*RAD)+PXW+XW+PXW+XW+PXW];
L2Y3 = [PXW*sin(M_ang*RAD),PXW*sin(M_ang*RAD)];
L23  = line(L2X3, L2Y3);
L2X4 = [0+PXW+XW+PXW+XW+PXW,PXW*cos(M_ang*RAD)+PXW+XW+PXW+XW+PXW];
L2Y4 = [0,PXW*sin(M_ang*RAD)];
L24  = line(L2X4, L2Y4);

% Creating the previous block...LL - XW - PXW

L0X1 = [0-XW-PXW,0+PXW-XW-PXW];         % The lower left corner - the two X
coord of the bottom line
L0Y1 = [0,0];
L01  = line(L0X1, L0Y1);
L0X2 = [0-XW-PXW,PXW*cos(M_ang*RAD)-XW-PXW];
L0Y2 = [0,PXW*sin(M_ang*RAD)];
L02  = line(L0X2, L0Y2);
L0X3 = [PXW*cos(M_ang*RAD)-XW-PXW,PXW*cos(M_ang*RAD)+PXW-XW-PXW];
L0Y3 = [PXW*sin(M_ang*RAD),PXW*sin(M_ang*RAD)];
L03  = line(L0X3, L0Y3);
L0X4 = [0+PXW-XW-PXW,PXW*cos(M_ang*RAD)+PXW-XW-PXW];
L0Y4 = [0,PXW*sin(M_ang*RAD)];
L04  = line(L0X4, L0Y4);

% The upper right corner (2) - UR + PXW + XW

U2X1 =
[PXW*cos(M_ang*RAD)+XW+PXW+PXW+XW,PXW*cos(M_ang*RAD)+PXW+XW+PXW+PXW+XW];
% The lower left corner - the two X coord of the bottom line
U2Y1 = [PXW*sin(M_ang*RAD)+YW,PXW*sin(M_ang*RAD)+YW];
U21  = line(U2X1, U2Y1);
U2X2 =
[PXW*cos(M_ang*RAD)+XW+PXW+PXW+XW,PXW*cos(M_ang*RAD)+PXW*cos(M_ang*RAD)+XW+PXW
+PXW+XW];
U2Y2 = [PXW*sin(M_ang*RAD)+YW,PXW*sin(M_ang*RAD)+PXW*sin(M_ang*RAD)+YW];
```

```
U22  = line(U2X2, U2Y2);
U2X3 =
[PXW*cos(M_ang*RAD)+PXW*cos(M_ang*RAD)+XW+PXW+PXW+XW,PXW*cos(M_ang*RAD)+PXW*co
s(M_ang*RAD)+PXW+XW+PXW+PXW+XW];
U2Y3 =
[PXW*sin(M_ang*RAD)+PXW*sin(M_ang*RAD)+YW,PXW*sin(M_ang*RAD)+PXW*sin(M_ang*RAD
)+YW];
U23  = line(U2X3, U2Y3);
U2X4 =
[PXW*cos(M_ang*RAD)+PXW*cos(M_ang*RAD)+PXW+XW+PXW+PXW+XW,PXW*cos(M_ang*RAD)+PX
W+XW+PXW+PXW+XW];
U2Y4 = [PXW*sin(M_ang*RAD)+PXW*sin(M_ang*RAD)+YW,PXW*sin(M_ang*RAD)+YW];
U24  = line(U2X4, U2Y4);

% The upper left corner (0) - UL - PYW - YW

U0X1 = [PXW*cos(M_ang*RAD)-PXW-XW,PXW*cos(M_ang*RAD)+PXW-PXW-XW];        % The
lower left corner - the two X coord of the bottom line
U0Y1 = [PXW*sin(M_ang*RAD)+YW,PXW*sin(M_ang*RAD)+YW];
U01  = line(U0X1, U0Y1);
U0X2 = [PXW*cos(M_ang*RAD)-PXW-XW,PXW*cos(M_ang*RAD)+PXW*cos(M_ang*RAD)-PXW-
XW];
U0Y2 = [PXW*sin(M_ang*RAD)+YW,PXW*sin(M_ang*RAD)+PXW*sin(M_ang*RAD)+YW];
U02  = line(U0X2, U0Y2);
U0X3 = [PXW*cos(M_ang*RAD)+PXW*cos(M_ang*RAD)-PXW-
XW,PXW*cos(M_ang*RAD)+PXW*cos(M_ang*RAD)+PXW-PXW-XW];
U0Y3 =
[PXW*sin(M_ang*RAD)+PXW*sin(M_ang*RAD)+YW,PXW*sin(M_ang*RAD)+PXW*sin(M_ang*RAD
)+YW];
U03  = line(U0X3, U0Y3);
U0X4 = [PXW*cos(M_ang*RAD)+PXW*cos(M_ang*RAD)+PXW-PXW-
XW,PXW*cos(M_ang*RAD)+PXW-PXW-XW];
U0Y4 = [PXW*sin(M_ang*RAD)+PXW*sin(M_ang*RAD)+YW,PXW*sin(M_ang*RAD)+YW];
U04  = line(U0X4, U0Y4);



% set thee figure axis..

xmins = [L0X1(1,1), L0X2(1,2), U0X1(1,1),U0X2(1,2)];
ymins = L0Y1(1,1);
xmaxs = [L2X1(1,2), L2X3(1,2), U2X1(1,2),U0X3(1,2)];
ymaxs = U0Y2(1,2);

% axis([min(xmins),max(xmaxs),ymins,ymaxs]);
axis([min(xmins),max(xmaxs),min(xmins),max(xmaxs)]);



% Patching the mine plan..

col=[1,1,0];
lx0 = [L0X1(1,1), L0X1(1,2), L0X4(1,2), L0X3(1,1)];
ly0 = [L0Y1(1,1), L0Y1(1,2), L0Y4(1,2), L0Y3(1,2)];
patch(lx0,ly0,col);
lx1 = [LLX1(1,1), LLX1(1,2), LLX4(1,2), LLX3(1,1)];
ly1 = [LLY1(1,1), LLY1(1,2), LLY4(1,2), LLY3(1,2)];
```

```
patch(lx1,ly1,col);
lx2 = [LRX1(1,1), LRX1(1,2), LRX4(1,2), LRX3(1,1)];
ly2 = [LRY1(1,1), LRY1(1,2), LRY4(1,2), LRY3(1,2)];
patch(lx2,ly1,col);
lx3 = [L2X1(1,1), L2X1(1,2), L2X4(1,2), L2X3(1,1)];
ly3 = [L2Y1(1,1), L2Y1(1,2), L2Y4(1,2), L2Y3(1,2)];
patch(lx3,ly3,col);
ux3 = [ULX1(1,1), ULX1(1,2), ULX4(1,1), ULX3(1,1)];
uy3 = [ULY1(1,1), ULY1(1,2), ULY4(1,1), ULY3(1,2)];
patch(ux3,uy3,col);
ux4 = [URX1(1,1), URX1(1,2), URX4(1,1), URX3(1,1)];
uy4 = [ULY1(1,1), URY1(1,2), URY4(1,1), URY3(1,2)];
patch(ux4,uy4,col);
ux0 = [U0X1(1,1), U0X1(1,2), U0X4(1,1), U0X3(1,1)];
uy0 = [U0Y1(1,1), U0Y1(1,2), U0Y4(1,1), U0Y3(1,2)];
patch(ux0,uy0,col);
ux2 = [U2X1(1,1), U2X1(1,2), U2X4(1,1), U2X3(1,1)];
uy2 = [U2Y1(1,1), U2Y1(1,2), U2Y4(1,1), U2Y3(1,2)];
patch(ux2,uy2,col);


% start off with a set of control points.....
% Considering just one turn only...either L - R or R - L...
% Follow the Start-End philosophy or thr end-start philosophy..
% Consider a quadratic, a cubic as well as a cubic bezier (needs 4 points!!)..

% The first and last points are exactly 50% in the width and 75% of width
respectively
% if the principle of start-end is followed...(center at start, offset at
end)..

% A Bezier curve with the start-end points and the two corner points as two
other ctrl pts..

% A CUBIC BEZIER PATH..
% General form of De'Casteljau algo for bezier curves..
% P(i,r) = P(i,r-1) + P(i+1,r-1), r = 1,2,....n
%                                 i = 0,1,...n-r, where n = degree of curve

% General matrix form of a cubic bezier curve...

% P(0,0)
% P(1,0) P(0,1)
% P(2,0) P(1,1) P(0,2)
% P(3,0) P(2,1) P(1,2) P(0,3) -> The last level of interpolation - Point on
the curve!

% The values in the first col are the given points - ctrl polygon points

hold on;

global degree;          % No of data points..
global t;               % Parameter t is of very high order - need 10 + for a
                        %smooth curve
global r;
global i;               % Same param as in Bezier.m is followed for
                        %simplicity..
```

```
global res;

degree = 4;
res = 0.1;
bx=zeros(degree,degree);
by=zeros(4,4);

% The points defining the ctrl polygon.....

bx(1,1) = (LLX1(1,2) + LRX2(1,2))/2;      % The point centered in the X road
                                          % way
by(1,1) = (LLY4(1,1) + LLY4(1,2))/2;
bx(2,1) = (LLX4(1,2) + LRX2(1,2))/2;      % The point centered in the X way,
                                          % but inline with end
by(2,1) = LLY4(1,2);
%bx(3,1) = URX1(1,1);
%by(3,1) = URY1(1,1);
bx(3,1) = URX1(1,1);
by(3,1) = (((LRY2(1,2) + URY1(1,1))/2) + URY1(1,1))/2;      % Point is inline
                                                            % with the point -
                                                            % 75% distance
bx(4,1) = (LRX3(1,1) + LRX3(1,2))/2;
by(4,1) = (((LRY3(1,1) + URY1(1,1))/2) + URY1(1,2))/2;    % 75% off the
corner....

%plot(bx,by,'ro');

interp = 1;

for t=0:res:1
   interp = interp+1;
   for r=2:1:degree+1
      for i=1:1:degree+1-r
         bx(i,r) = bx(i,r-1)*(1-t) + bx(i+1,r-1)*t;
         by(i,r) = by(i,r-1)*(1-t) + by(i+1,r-1)*t;
      end;
   end;

   % Store the last level of interpolation - the last point on the curve

   ptx(interp)=bx(1,degree);
   pty(interp)=by(1,degree);

end;             % End of t loop ->


% Curve plotting routine.....

% size of ptx and pty = interp

for j = 2:1:interp-1        % Ignore the last point
   x1 = ptx(1,j);
   y1 = pty(1,j);
   x2 = ptx(1,j+1);
   y2 = pty(1,j+1);
   c1 = [x1,x2];
   c2 = [y1,y2];
```

```
   crv = line(c1,c2);
   set(crv,'LineWidth',[1.0]);
   plot(x1,y1,'b');
   plot(x2,y2,'b');
end;

plot(bx,by,'ro');

zoom on;

% Repeat the PIG Animation process....
% To find the intersection of the PIG with the curve, draw a circle and find
intersection

% when solving for a param cubic, -> easy as it has a closed form
expression....

% CAN ALSO BE A PIECEWISE CUBIC BEZIER CURVES...

% Write a routine to plot the second set of curves in case of a S turn..
% Use the same as the first curve and just add code to calculate the new
points

% The points on the curves are ptx and pty (X,Y) respectively...there will be
12 points...
% Position the PIG at the starting point..

% finding the other intersection of the PIG with the curve...use the line
segments
% There are 10 lines bet the 11 points...find the equation of each line and
% substitute the PIG sweep coords in the line eqn, if on  = 0 (need to round
off)

% THIS IS THE ANIMATION ROUTINE.....NEED TO COMPUTE THE END COORDS FIRST...
% The other end of the PIG as of now is positioned 2 points ahead....

   pig_x1 = ptx(1,2);     % Skip the first point as it is the origin....
   pig_y1 = pty(1,2);
   pig_x2 = ptx(1,4);
   pig_y2 = pty(1,4);
   pig_x = [pig_x1,pig_x2];
   pig_y = [pig_y1,pig_y2];

   pig = line(pig_x,pig_y);
   set(pig,'LineWidth',[3.0],'Color',[1.0,0.0,0.0]);


   for i=2:1:interp-2          % should be interp-1 if just one point is
considered..
      delete(pig);
      pig_x1 = ptx(1,i);     % Skip the first point as it is the origin....
      pig_y1 = pty(1,i);
      pig_x2 = ptx(1,i+2);
      pig_y2 = pty(1,i+2);

      pig_x = [pig_x1,pig_x2];
      pig_y = [pig_y1,pig_y2];
```

```
      pig = line(pig_x,pig_y);
      set(pig,'LineWidth',[3.0],'Color',[1.0,0.0,0.0]);
      pause(0.1);
   end;
```

dynamic_clear.m

```
% Aish, LA Dynamic path planning..
% Clears the interpolation pts..
% USe with parent dynamicpath.m

% Patching the mine plan..

hold on;
cla;

% Patching the mine plan..

col=[1,1,0];
lx0 = [L0X1(1,1), L0X1(1,2), L0X4(1,2), L0X3(1,1)];
ly0 = [L0Y1(1,1), L0Y1(1,2), L0Y4(1,2), L0Y3(1,2)];
patch(lx0,ly0,col);
lx1 = [LLX1(1,1), LLX1(1,2), LLX4(1,2), LLX3(1,1)];
ly1 = [LLY1(1,1), LLY1(1,2), LLY4(1,2), LLY3(1,2)];
patch(lx1,ly1,col);
lx2 = [LRX1(1,1), LRX1(1,2), LRX4(1,2), LRX3(1,1)];
ly2 = [LRY1(1,1), LRY1(1,2), LRY4(1,2), LRY3(1,2)];
patch(lx2,ly1,col);
lx3 = [L2X1(1,1), L2X1(1,2), L2X4(1,2), L2X3(1,1)];
ly3 = [L2Y1(1,1), L2Y1(1,2), L2Y4(1,2), L2Y3(1,2)];
patch(lx3,ly3,col);
ux3 = [ULX1(1,1), ULX1(1,2), ULX4(1,1), ULX3(1,1)];
uy3 = [ULY1(1,1), ULY1(1,2), ULY4(1,1), ULY3(1,2)];
patch(ux3,uy3,col);
ux4 = [URX1(1,1), URX1(1,2), URX4(1,1), URX3(1,1)];
uy4 = [ULY1(1,1), URY1(1,2), URY4(1,1), URY3(1,2)];
patch(ux4,uy4,col);
ux0 = [U0X1(1,1), U0X1(1,2), U0X4(1,1), U0X3(1,1)];
uy0 = [U0Y1(1,1), U0Y1(1,2), U0Y4(1,1), U0Y3(1,2)];
patch(ux0,uy0,col);
ux2 = [U2X1(1,1), U2X1(1,2), U2X4(1,1), U2X3(1,1)];
uy2 = [U2Y1(1,1), U2Y1(1,2), U2Y4(1,1), U2Y3(1,2)];
patch(ux2,uy2,col);


interp = 1;

for t=0:res:1
   interp = interp+1;
   for r=2:1:degree+1
      for i=1:1:degree+1-r
         bx(i,r) = bx(i,r-1)*(1-t) + bx(i+1,r-1)*t;
         by(i,r) = by(i,r-1)*(1-t) + by(i+1,r-1)*t;
      end;
```

```
   end;

   % Store the last level of interpolation - the last point on the curve

   ptx(interp)=bx(1,degree);
   pty(interp)=by(1,degree);

end;             % End of t loop ->


% Curve plotting routine.....

% size of ptx and pty = interp

for j = 2:1:interp-1        % Ignore the last point
   x1 = ptx(1,j);
   y1 = pty(1,j);
   x2 = ptx(1,j+1);
   y2 = pty(1,j+1);
   c1 = [x1,x2];
   c2 = [y1,y2];
   crv = line(c1,c2);
   set(crv,'LineWidth',[1.5]);
   plot(x1,y1,'b*');
   plot(x2,y2,'b*');
end;

plot(bx,by,'ro');
```

dynamic_interp.m

```
% Aish, LA Dynamic path planning
% Use with parent dynamicpath.m

% Just plots the interpolation points...

hold on;

interp = 1;

for t=0:res:1
   interp = interp+1;
   for r=2:1:degree+1
      for i=1:1:degree+1-r
         bx(i,r) = bx(i,r-1)*(1-t) + bx(i+1,r-1)*t;
         by(i,r) = by(i,r-1)*(1-t) + by(i+1,r-1)*t;
      end;
   end;

   % Plotting Routine...

   rc=0;
   gc=0;
   bc=0;
```

```
   for r=1:1:degree
      rc=r*(1/degree);
      gc=r*(1/degree);
      bc=r*(1/degree);
       for i=1:1:degree-r
         x1 = bx(i,r);
         y1 = by(i,r);
         x2 = bx(i+1,r);
         y2 = by(i+1,r);
         l1=[x1,x2];
         l2=[y1,y2];
         l=line(l1,l2);
         set(l,'Color',[rc,gc,bc]);
         hold on;
         plot(x1,y1,'ro');
         plot(x2,y2,'ro');
      end;
   end;

   % Store the last level of interpolation - the last point on the curve

   ptx(interp)=bx(1,degree);
   pty(interp)=by(1,degree);

end;             % End of t loop ->


% Curve plotting routine.....

% size of ptx and pty = interp

for j = 2:1:interp-1        % Ignore the last point
   x1 = ptx(1,j);
   y1 = pty(1,j);
   x2 = ptx(1,j+1);
   y2 = pty(1,j+1);
   c1 = [x1,x2];
   c2 = [y1,y2];
   crv = line(c1,c2);
   set(crv,'LineWidth',[1.5]);
   plot(x1,y1,'b*');
   plot(x2,y2,'b*');
end;

plot(bx,by,'ro');
```

path_param.m

```
% Aish, LA Dynamic path planning
% Use with parent dynamicpath.m

% Just plots the interpolation points...

hold on;
```

```
interp = 1;

for t=0:res:1
   interp = interp+1;
   for r=2:1:degree+1
      for i=1:1:degree+1-r
         bx(i,r) = bx(i,r-1)*(1-t) + bx(i+1,r-1)*t;
         by(i,r) = by(i,r-1)*(1-t) + by(i+1,r-1)*t;
      end;
   end;

   % Plotting Routine...

   rc=0;
   gc=0;
   bc=0;

   for r=1:1:degree
      rc=r*(1/degree);
      gc=r*(1/degree);
      bc=r*(1/degree);
       for i=1:1:degree-r
         x1 = bx(i,r);
         y1 = by(i,r);
         x2 = bx(i+1,r);
         y2 = by(i+1,r);
         l1=[x1,x2];
         l2=[y1,y2];
         l=line(l1,l2);
         set(l,'Color',[rc,gc,bc]);
         hold on;
         plot(x1,y1,'ro');
         plot(x2,y2,'ro');
      end;
   end;

   % Store the last level of interpolation - the last point on the curve

   ptx(interp)=bx(1,degree);
   pty(interp)=by(1,degree);

end;             % End of t loop ->


% Curve plotting routine.....

% size of ptx and pty = interp

for j = 2:1:interp-1        % Ignore the last point
   x1 = ptx(1,j);
   y1 = pty(1,j);
   x2 = ptx(1,j+1);
   y2 = pty(1,j+1);
   c1 = [x1,x2];
   c2 = [y1,y2];
   crv = line(c1,c2);
```

```
   set(crv,'LineWidth',[1.5]);
   plot(x1,y1,'b*');
   plot(x2,y2,'b*');
end;

plot(bx,by,'ro');
```

APPENDIX II

This section describes the developmental efforts contributed by the author towards the development of the interface software using LabVIEW®. Detailed descriptions of the most recent Virtual Instrument (VI) developed for a multi-unit system is the main focus. This section concentrates on this VI as its structure is based off the developments of the earlier interface versions, and this is currently employed by the team for experimentation.

As described in the Volume I of this documentation, it is anticipated that there will be no further requirements for extensive development of a new interface engine. The VI under discussion encompasses all the possible requirements for future developments. The structure of the VI is hierarchical and the functionality of this VI is expected to satisfy all of the current needs of the automation effort and is anticipated to cater to most future requirements too.

This sections also lists the C source code implemented by the author for the generation of the CRC 16 Checksum for the SICK® LMS 200 Laser scanner. This code is essentially a direct implementation of the C routine found in the user's manual supplied by the manufacturers of the scanner. The chapter is organized as follows…

i. Listing of the CRC 16 Checkcum code – C Source file
ii. Detailed frame description of the multi-unit VI
iii. SUB VI's used in the multi-unit VI.
iv. RS422 PCMCIA NULL MODEM pin-out

Source code reference table

S.No	Function	Parent file	Child files
1	CRC 16 Checksum code	Checksum.cpp	None
2	Multi-unit control interface	PMS_MULTI_UNIT.VI	Rec_data.vi Send_data.vi

i. Source code listing for computing the CRC16 Checksum for the SICK® scanner

```
#include "stdafx.h"
#include<stdio.h>
#include<conio.h>

void main()
{

	unsigned char buff[100];

	int len;			//length of the string - # of bytes
	int i=0;			//loop counter
	int j;				//input loop counter

	unsigned short crc16=0, crc_data=0;

	printf("\nEnter the length :");
	scanf("%d",&len);		//input length

	printf("\nEnter the string byte by byte preceeded by 0x :\n");

	for(j=0;j<len;j++)		//input hex values byte by byte
	{
		printf("\nEnter byte # %d :",j+1);
		scanf("%x",&buff[j]);
	}

	//loop to compute checksum...

	while(len)
	{
		crc_data = crc_data<<8;		//shift left by 8 bits
		crc_data = crc_data & 0xff00; //perform AND with 0xff00
		crc_data = crc_data | buff[I];//perform OR with input data byte
		i=i+1;				//increment to next data byte

		if(crc16 & 0x8000)		//higher order bit == 1
		{
			crc16 = crc16 <<1;		//left shift 1 bit
			crc16 = crc16 ^ 0x8005;		//XOR with 0x8005
			crc16 = crc16 ^ crc_data;	//XOR with crc_data
		}

		else				//higher order bit == 0
		{
			crc16 = crc16 << 1;		//shift left by 1 bit
			crc16 = crc16 ^ crc_data;	//XOR with crc_data
		}

		--len;				//pre-decrement while count
	}

	printf("\nThe CHECKSUM is %x\n\n",crc16); //flush to display
	getch();
}
```

ii. The Multi-unit control interface – LabVIEW® VI

This section concentrates on the description of the Multi-unit control interface developed for use on the 1/10th scale prototype using the PMS scanners for guidance and navigation. The discussion is mainly focused towards elaborating on the functional aspects of the VI as compared to its creation. The VI is broken down into frames, and each frame is described with a motive towards introducing the user to the front panel controls, and their functions. The discussion begins by highlighting all the front panel controls, and its functions, and then continues to the frame-wise detailing of the VI block diagram.

The Front Panel

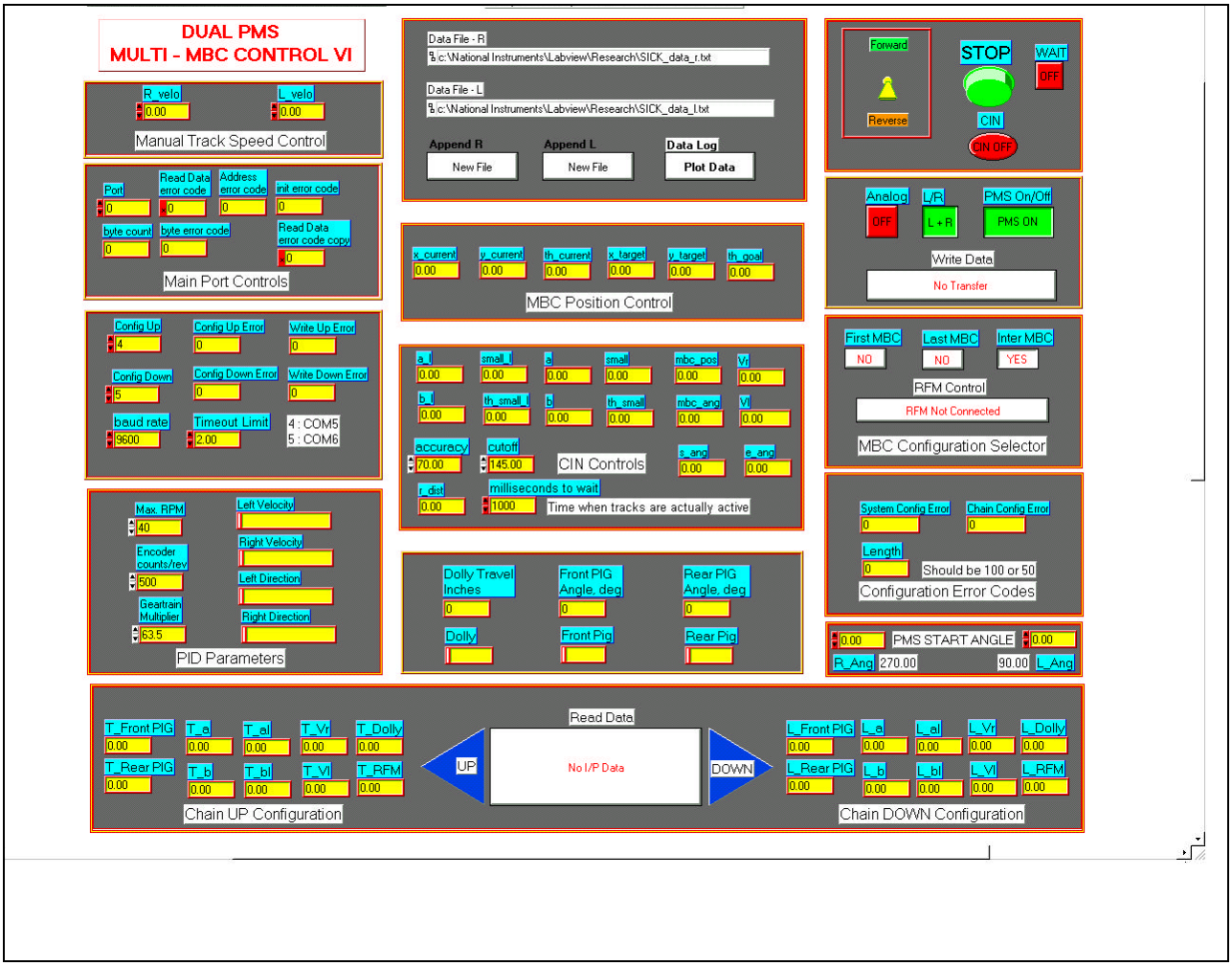


Fig. 1. Front Panel of the Multi-unit control VI

Figure 1, is a screen shot, of the front panel of the multi-unit control interface. The discussion below lists all the controls in this panel, with a short description of each of them with their functions. The widget labels as found in the front panel are used to maintain consistency in the description.

Boolean Controls (Switch and Button controls)

1. STOP button – Activation of the STOP button will completely halt both the left and right tracks from responding to any commands. This button will halt the tracks, irrespective of whether operating the unit in automatic or manual mode. Hence, this button should be used only in cases of an emergency stop procedure. Default state is OFF – VI is functional, with the tracks responding to commands.
2. WAIT button – This button alternates between two modes of motion control viz., The continuous mode, where the tracks continue to be active while the PMS's are scanning, and the 'Wait and Go' strategy, wherein the tracks are active only for a specified period of time specified by the 'milliseconds to wait' control in the 'CIN controls' console. Hence, while operating in the latter mode, the tracks are completely stopped while the scanners perform the scanning routine. Default value for this button is FALSE – corresponding to the continuous mode.
3. CIN button – This button allows the user to change between 'Automatic' and 'Manual' mode of control. While this button is active (ON), it allows the automatic control of the system, by processing the data via the CIN Object codes. In the default OFF state, the mode is manual, wherein, the MBC tracks can be controlled by supplying the required velocity in the 'Manual Track Speed Control' console.
4. Mode switch – This button enables the selection of the mode of driving, when in automatic mode, or when the CIN button is on. Automatic mode corresponds to the control of the unit via the object codes, loaded in the Code Interface Nodes. The two modes allowed for operation are FORWARD and REVERSE. It must be noted that this button acts as a dummy, when in manual mode of operation.
5. Analog button – This screen widget controls the analog sensor inputs. The button specifies whether the VI should receive and display the three analog measurements, viz.,

the rear PIG angle, out by PIG angle and the dolly distance. Default state is OFF – corresponding to null display of these values.

6. L/R button – This button allows the selection of either the LEFT scanner, or both the left and right scanners. This button should be used only while de-budding hardware, and normally should be in ON state for proper functioning of the VI.
7. PMS On/Off button – This controls the functioning of the PMS. When in OFF state (the default state), the PMS's scanners are inactive. It must be noted that this button should always be in ON state for the VI to function without errors.
8. Write data button – Activation of this button will enable the data sharing routine for PC-PC communication between two host controllers through the RS422 PCMCIA ports. Default state for this control is OFF.
9. MBC Configuration Selector console – This console consists of three MBC configuration selection buttons and one button for the RFM control. The three configuration selectors, viz., the First MBC, Inter MBC, and Last MBC allows the same VI to be used on different systems. It is the responsibility of the user to choose the operating unit's configuration prior to operation. Default state is OFF for the First MBC, and Last MBC buttons, while is ON for the Inter MBC button. It must also noted that these buttons do not act as radio buttons, and the uses must make sure that only one of them is in ON state at any time. Simultaneous ON states of any two will result in data loss.
10. RFM Control button – This button is no longer being used in the current stage of operation. The button was originally designed to choose between a tethered and un-tethered configuration of the last PIG with reference to riding on the RFM.
11. Read Data button – This button will activate the 'Read' portion of the PC-PC data sharing routine. When this button is ON, the host controller will receive inter-MBC data from the leading and trailing MBC controllers. Default state for this control is OFF.
12. Data Log button – This control activates the data logging for storing the scanned data from both the scanners. Default state is OFF – corresponding to plotting data on the screen. When in ON state, the VI will generate an ASCII text file, at the location specified by the 'Data File –R' and 'Data File – L' controls for the file path.
13. Append L and Append R buttons – These two controls switch between the modes of 'appending data to an existing file' or 'overwriting data'. This is used for the storage of

the scanned data from the PMS scanners. These buttons are to be used in combination with the 'Data Log' button.

Variable Controls

1. The 'R_ang' and 'L_ang' controls found in the 'PMS START ANGLE' control console allow the start angle configuration of the PMS laser scanners. Default values are 0.00, for both these controls. It must be noted that the changing of the start angle values will affect the data plotting, and will lead to the 're-setting' of the VI if no corresponding changes are made to the CIN object code processing the scanner data. These controls are for developmental purposes only, and they command no usage while operating the unit.
2. The 'accuracy' and 'cutoff' controls in the 'CIN Controls' console are used when the simple line-finding routine is used in the CIN object code. These two values are used for the creation of a usable window for the line-finding by restricting the scanning range. Default values are 70 and 145 respectively, and these controls act as dummies while using the 'iterative end point line splitting' technique for localization. During these period, it is left to the developer to manipulate these controls as desired.
3. The 'milliseconds to wait' control, also in the 'CIN Controls' console is used with the 'Wait and Go' strategy of driving. As mentioned while discussing the 'WAIT' button, this control provides the user with the flexibility to change the time period for which the track will be active while in this mode of driving. The default value is 1000 ms, with the control allowing changes in increments of 500 ms. Using a value of 0 ms corresponds to null delay, and hence, the track will never be active.
4. The 'R_velo' and 'L_velo' controls in the 'Manual Track Speed Control' console are used with the manual mode of control. These controls allow the user to input the right and left track speeds, when the CIN button is in OFF state. These controls become inactive while in automatic mode – or while the CIN control is in ON state.
5. The 'Port', and 'Config Up and Config Down' controls in the 'Main Ports Control' and the 'Transfer Ports Controls' console respectively allow the selection of the main port, and PC-PC data sharing port for the VI. It should be noted that the values for these controls are integers, with COM 1 = 0, COM 2 = 1, and so on.

6. The ‘baud rate’ control in the ‘Transfer Port controls’ console allows the selection of the transfer baud rate for the sharing of inter-MBC data through the ‘Config Up’ and ‘Config Down’ ports. Default value is set to 9600 baud while the control allows the operation at 19200 baud rate.
7. The ‘Timeout Limit’ control was included to remove the data clogging of serial port data between the host controllers. (Section 6.2.3, pg 62, Vol I). The value is in seconds, and the default value is 2 seconds. It is important to set these controls at different settings (different timeout limits) when operating with data sharing between two host controllers.
8. The three controls for ‘Max RPM’, ‘Encoder counts/rev’ and the ‘Geartrain Multiplier’ in the ‘PID parameters’ console allow change in the closed loop feedback operation of the track control system. These controls are for developmental purposes only, and they should never be altered while the VI is in operation.

Indicators and Error handlers

1. The set of six indicators in the ‘MBC Position Control’ console are the outputs from the line-finding Code Interface Node. These values are a direct reflection of the output values computed by the line-finding object code. It must be noted that these values directly act as inputs for the SSS algorithm. The values displayed in this indicator corresponds to the algorithm’s inputs - current MBC location $S(X_s, Y_s, \theta_s)$ and the desired or target location $F(X_f, Y_f, \theta_f)$ (Section 7.3.2, pg 80, Vol I).
2. The set of 15 indicators in the ‘CIN Controls’ console display the result of the line-finding algorithm. These indicators are programmed to display results from both the simple line-finding, and the iterative end point line splitting techniques of self-localization. A display of a null string or ‘NaN’ corresponds to an error in the CIN object code, and hence, these indicators act as error identifiers for the CIN object code. Moreover, these indicators are active only when the ‘CIN’ button is in ON state, else, all the indicators display ‘0.00’.
3. The three analog measurement controllers, directly below the ‘CIN Controls’ console display the analog sensor data from the rear PIG potentiometer, out by PIG potentiometer, and the dolly distance. The angles are displayed in degrees (0 +/- 90),

while the dolly distance is indicated in inches, with 0 representing a fully retracted slider, and 6” corresponding to a fully extended slider.

4. The two sets of 10 indicators, on either side of the ‘Read data’ button, with the labels ‘Chain UP Configuration’ and ‘Chain DOWN Configuration’ display the inter-MBC data read from the immediate leading and trailing units. These indicators are active only when data sharing is enabled on both the communicating host controllers.
5. The ‘Config Up Error’, and ‘Config Down Error’ indicators in the ‘Transfer Port Controls’ console display a null value, when the VI is in successful operation. These indicators display different error codes when a serial port initialization error occurs. Error codes are listed in the LabVIEW® User manual. The most commonly observed error is ‘37’ – this corresponds to absence of port or Invalid port address.
6. The two error handling controls, ‘Write Up Error’ and ‘Write Down Error’ correspond to the error code displays of the data write error. Whenever a write error occurs in the data sharing ports, these indicators display different error codes.
7. The error indicators in the ‘Configuration Error Codes’ console are the most important error handlers. These display error values, when the user has overlooked the instructions listed in this documentation. The ‘System Config Error’ and the ‘Chain Config Error’ displays a value of ‘1’ when none of the MBC Configuration selector buttons are in TRUE state, and when the VI is in Read or Write mode respectively. These indicators usually display a null value when the VI is in normal operation.
8. The ‘Length’ indicator displays the size of the incoming inter-MBC data. The three possible values for this indicator are 100, 50 or 4. If the indicator displays other values than the ones specified, the VI is in abnormal operation.

Start Up and Shut Down sequence

The following procedure is recommended for a normal start up and shut down of the VI. As the VI involves a hierarchical frame-wise execution, the start up and shut down sequence is vital to the successful operation of the interface.

START UP

The VI can be set into continuous mode of running by pressing the ‘run continuous’ icon on the screen menu, or by using ‘Ctrl+R’ on the keyboard. It is important to switch the PMS button, L/R button to the ON state before running the VI. It is the user’s responsibility to enter the correct values for the port numbers in the respective controls.

SHUT DOWN

The VI can be shutdown in two ways. The first involves the shutting down of the tracks, and the PMS by switching the ‘STOP’ button and the PMS button to ON and OFF state respectively, in the same order as described. This will stop the tracks from motion, and the scanners from scanning. Once this is performed, the VI can be physically halted using the screen menu stop button. The second method is to halt the VI by using the ‘stop’ icon without changing the front panel controls. This methods is generally not recommended for normal shutdown, but can be used for an emergency mode. Shutting down the VI using the latter method will not ensure halting of the tracks, and might require re-setting the controller boards, or even recycling the power.

Notes

1. It is important to retain the variable names, or the labels on the front panel controls. Changing these labels will directly affect the block diagram, and in-turn the internal variable names used in the CIN object code. Hence, if the front panel labels are altered, it is vital to make corresponding changes to the object codes.
2. Secondly, the data type of the indicator, or control should not be changed as preferred as this again required corresponding changes to be done in the CIN object codes.
3. It is very important to load the appropriate resource file in the Code Interface Nodes to prevent the VI from abnormal abortion. Loading a faulty, or a different object code will cause abnormal execution, and might even require re-starting of the PC.

Internal functioning of the VI

This section provides a brief frame-wise description of each of the frames constituting the VI. Extensive screen shots are included for additional clarity.

Frame 0 – Serial port initialization

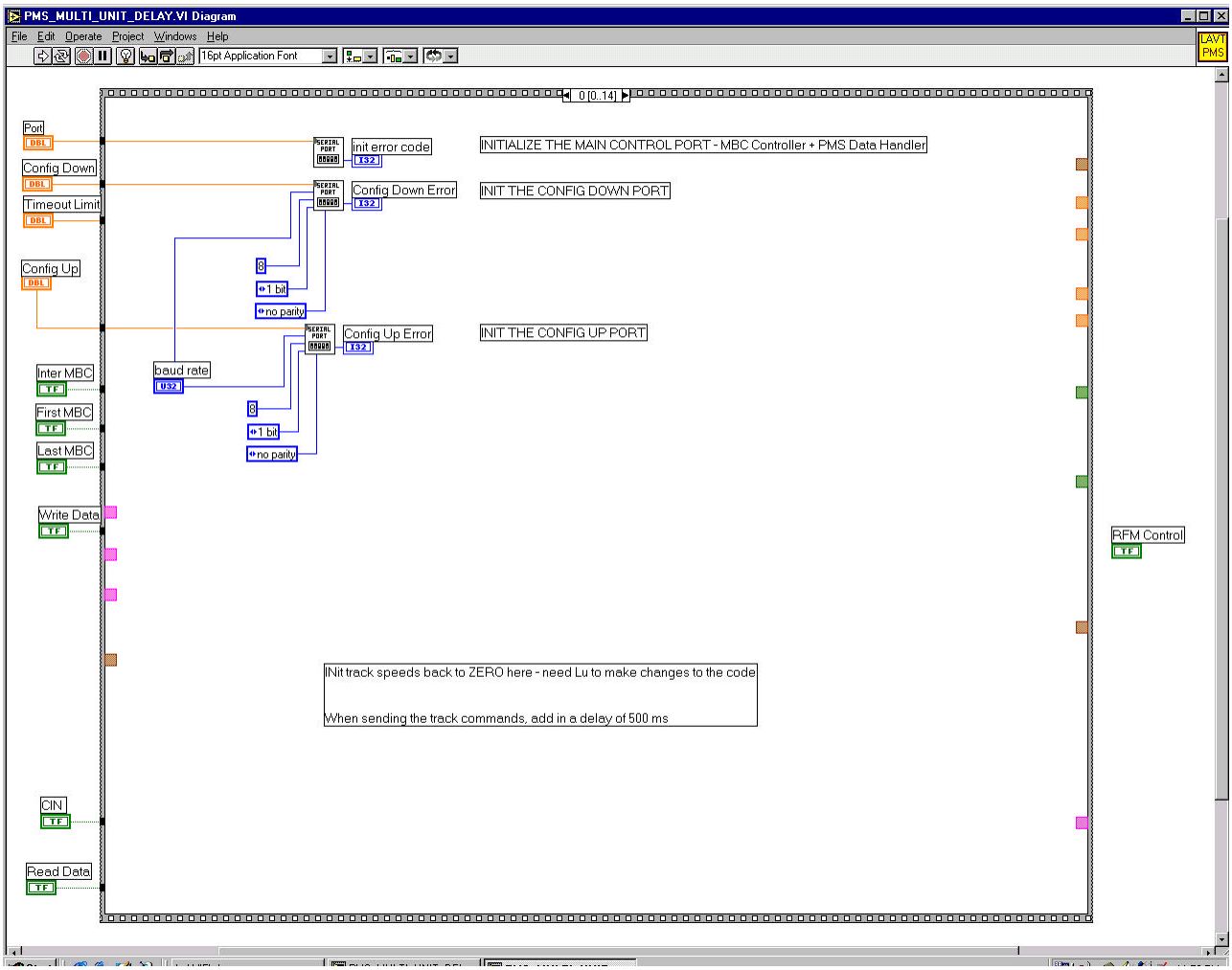


Fig. 2. Frame 0 of 14 – PMS_MULTI_UNIT_DELAY.VI

This is the ‘Serial Port Initialization’ routine. This frame uses the ‘Serial Port Init.vi’ Sub-VI for performing the port initialization. The transfer baud rate is controlled by the ‘baud rate’ control found in the ‘Main Port Controls’ console in the front panel. It must be noted that the port initialization routine does not employ a parity bit for error checking. The error code, if any due to initialization error is displayed in the front panel.

Frame 1 – Select driving mode - ‘Wait and Go’ & ‘Continuous’.

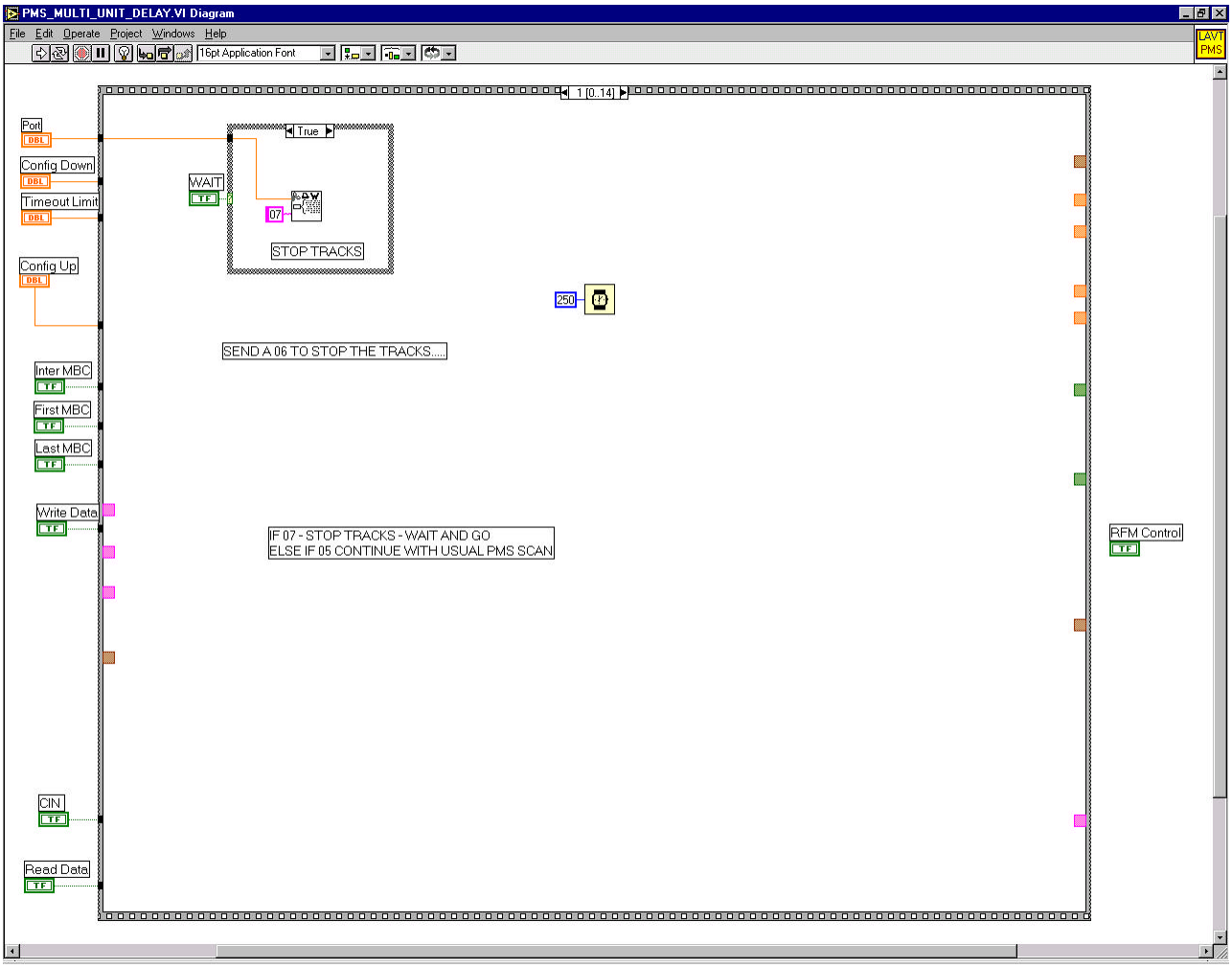


Fig. 3. Frame 1 of 14 – PMS_MULTI_UNIT_DELAY.VI

This frame allows the selection between the two modes of driving, viz., the ‘Continuous’ mode and the ‘Wait and Go’ mode of motion control. The Boolean variable controlling this selection is the ‘Wait’ button on the front panel. If ‘Wait’ is in ON state, this frame sends a ‘07h’ init string to the MBC controller, instructing it to stop the tracks while the PMS’s are scanning. On the other hand, while the ‘Wait’ button is in OFF state, the frame sends a ‘05h’ init string instructing the MBC controller to choose the ‘continuous’ mode, wherein the tracks continue to be active whole the scanners perform the scanning.

Frame 2 – Request for PMS Scan sequence

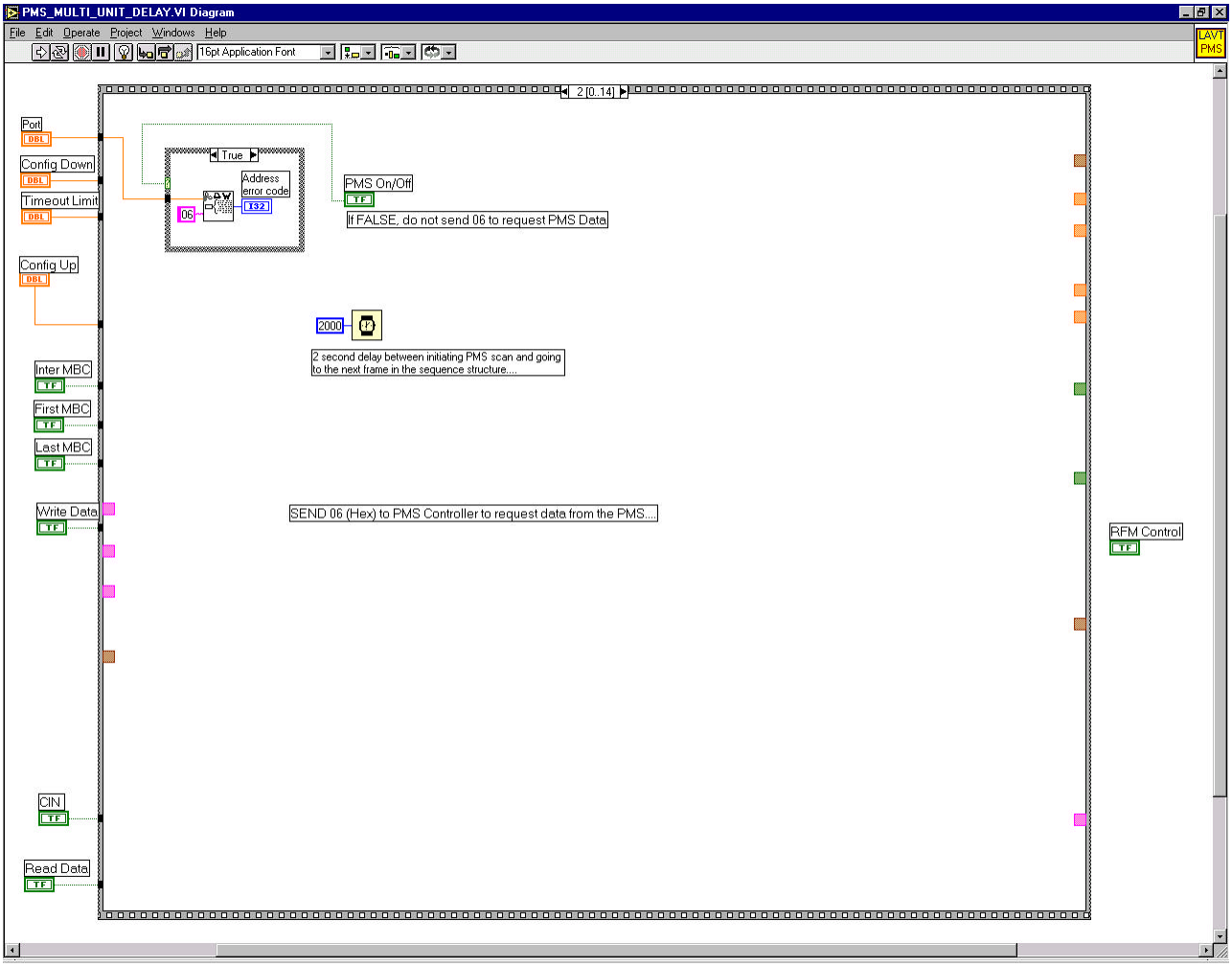


Fig. 4. Frame 2 of 14 – PMS_MULTI_UNIT_DELAY.VI

This is the 'Request for scan' frame, wherein if the 'PMS On/Off' button is in ON state, LabVIEW sends a init string '06h' to the MBC controller to initiate a scan sequence. The presence of the 'millisecond' delay icon is to ensure that the init string has been flushed out to the MBC controller before continuing with the next frame.

Frame 3 – Receive PMS data

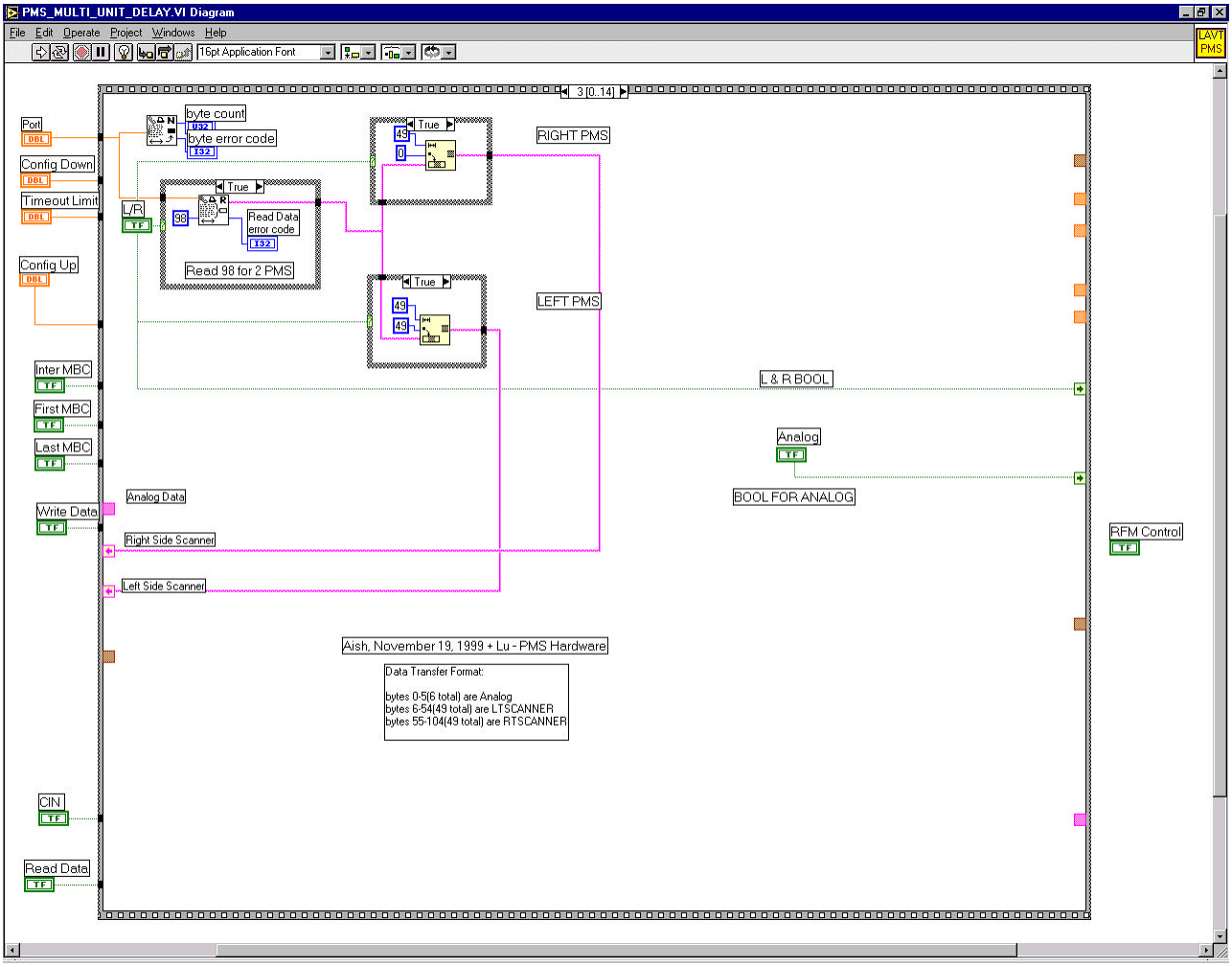


Fig. 5. Frame 3 of 14 – PMS_MULTI_UNIT_DELAY.VI

This frame is the ‘Receive PMS data’ frame. The frame reads the bytes at the serial port, sent by the ob-board controller on the ‘Request to scan’ commands initiated in the previous frame. The selection of choosing between the Left scanner, and Left & Right scanner is performed in this frame. If the L/R button is in ON state, this frame reads 98 bytes from the serial port, else the frame reads 49 bytes for the left scanner.

Frame 4 – Plot PMS data to front panel – Right side

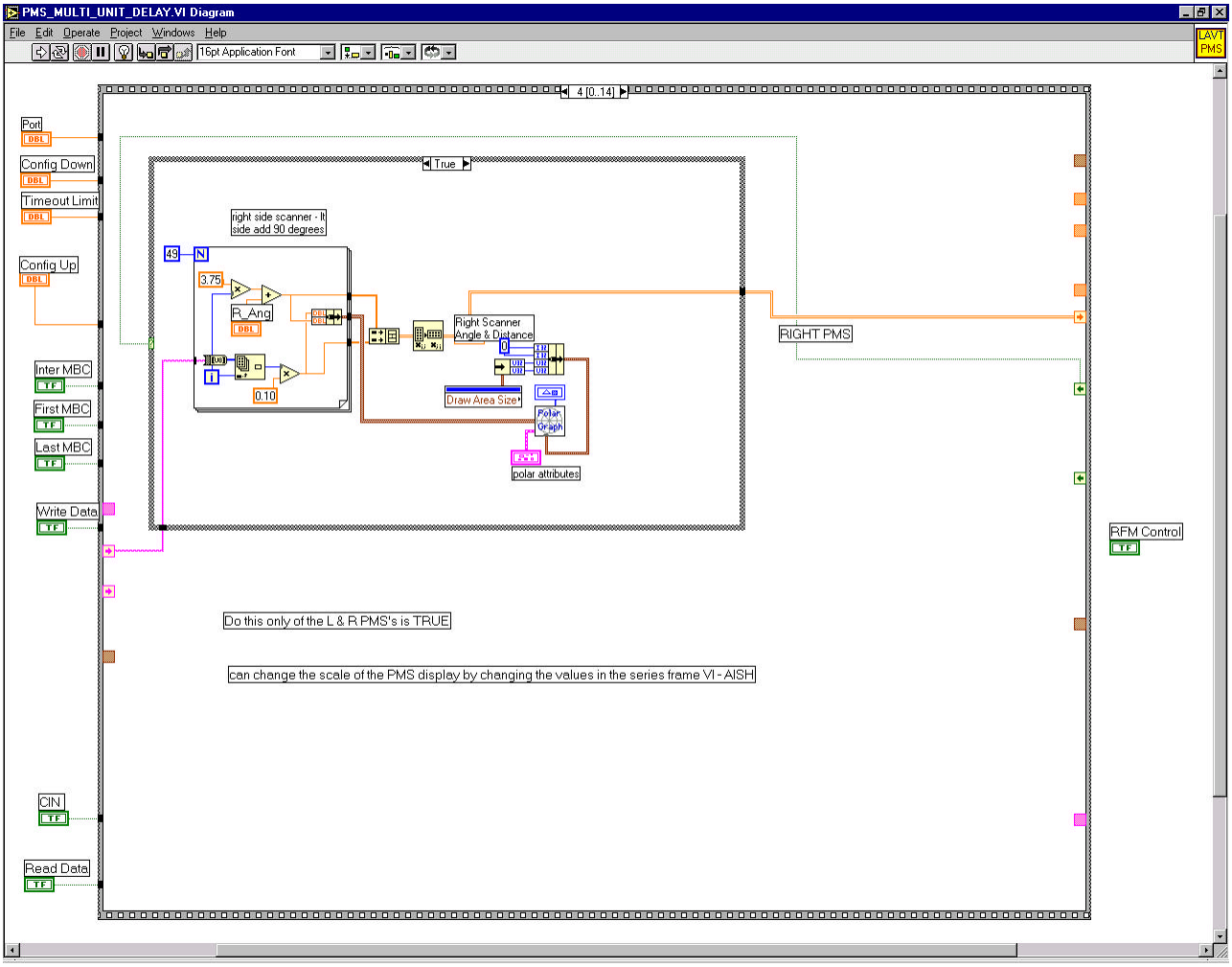


Fig. 6. Frame 4 of 14 – PMS_MULTI_UNIT_DELAY.VI

The plotting of the polar data from the Right PMS scanner is performed in this frame. The streaming data from the serial port is sent to the front panel using the 'PolarGraph.vi' Sub-VI found under the 'Picture' library.

Frame 5 – Plot PMS data to front panel – Left side

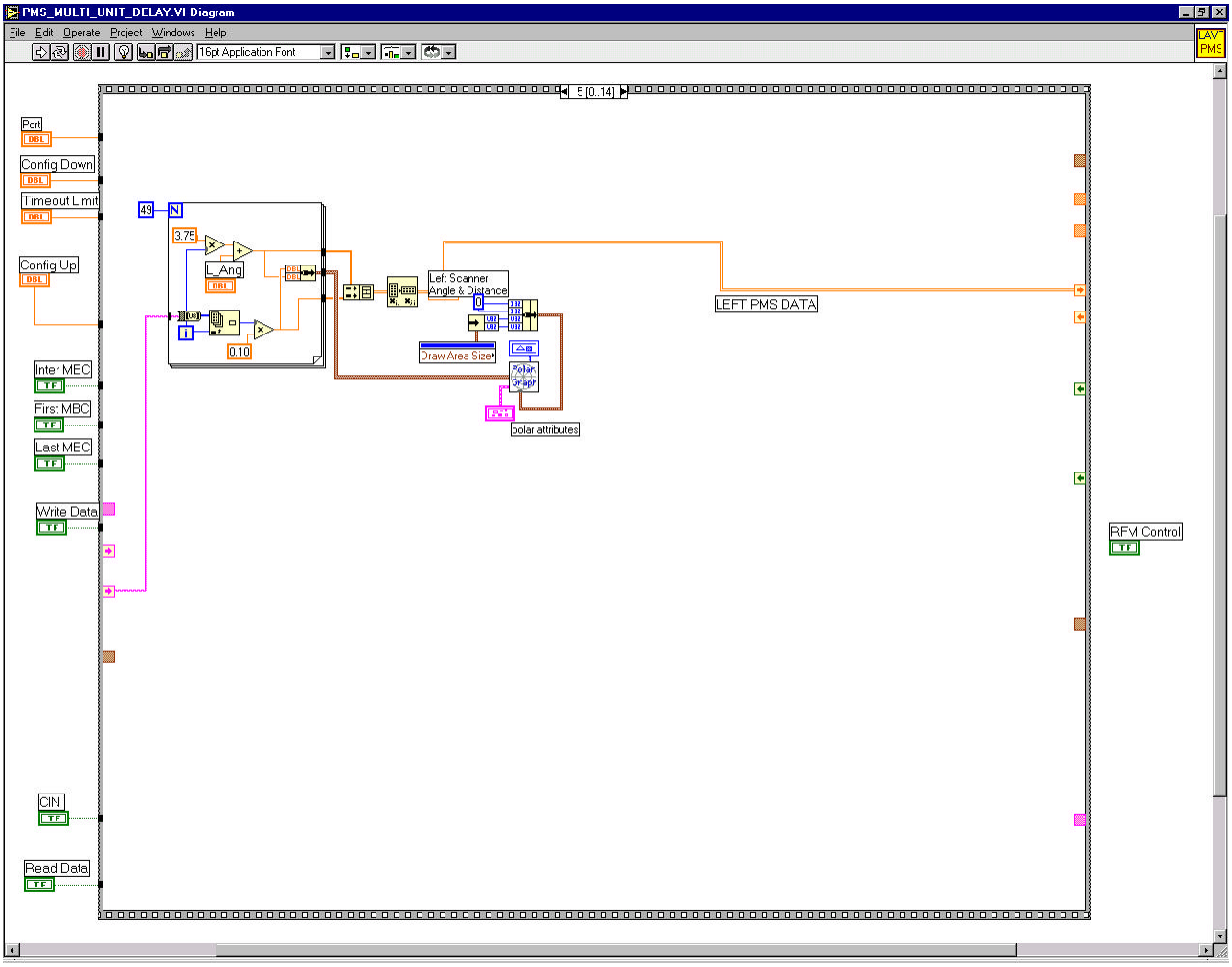


Fig. 7. Frame 5 of 14 – PMS_MULTI_UNIT_DELAY.VI

This frame performs the data plotting routine for the Left side PMS laser scanner. It uses the same methodology as the previous frame.

Frame 6 – Request for 'Analog' data transfer

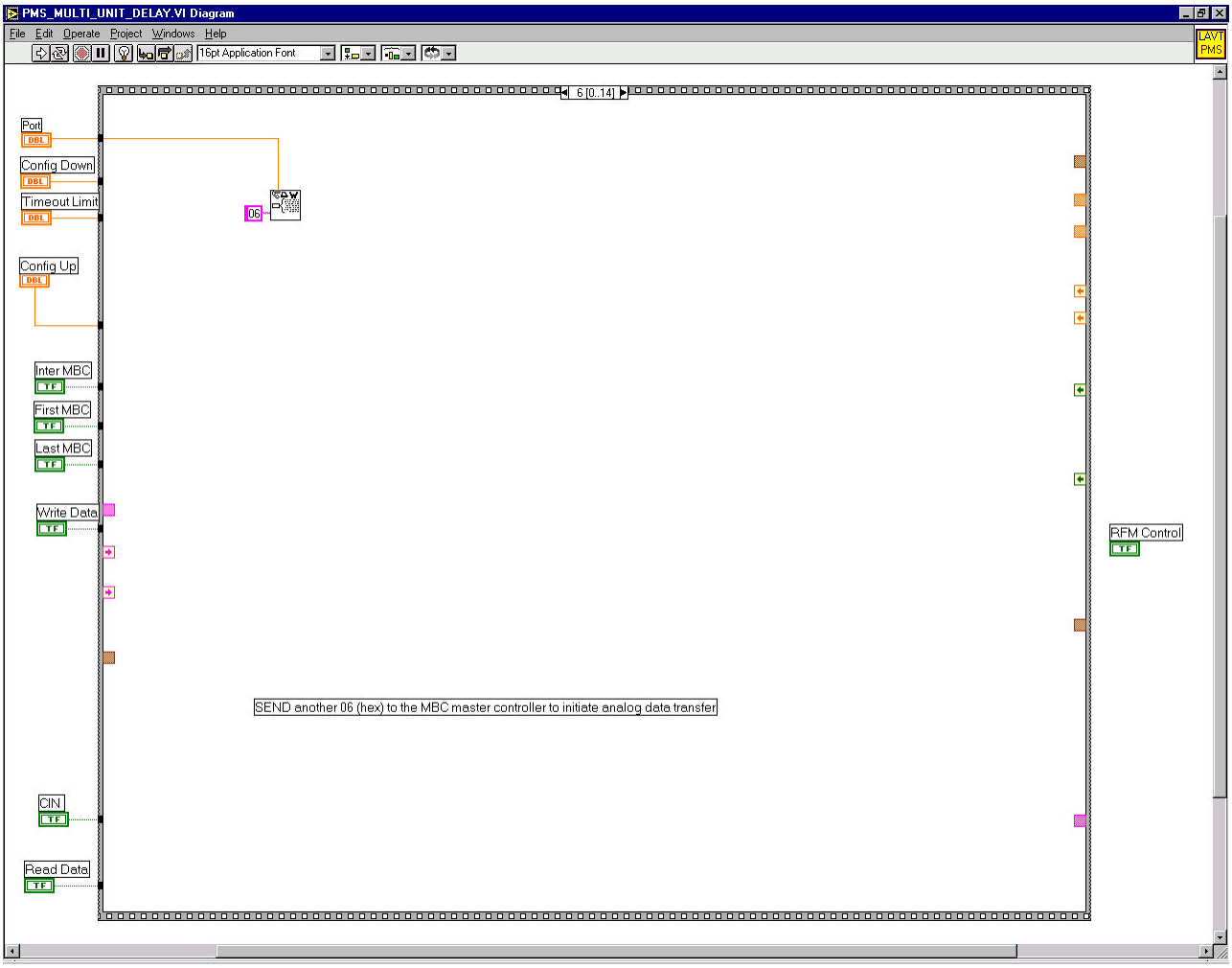


Fig. 8. Frame 6 of 14 – PMS_MULTI_UNIT_DELAY.VI

The request for analog data – The two angles and dolly slider distance is performed in this frame. The frame sends an initialization string of '06h' to the on-board controller intimating the controller to send analog data back to the host PC. The analog values received are then displayed.

Frame 7 – Receive analog data

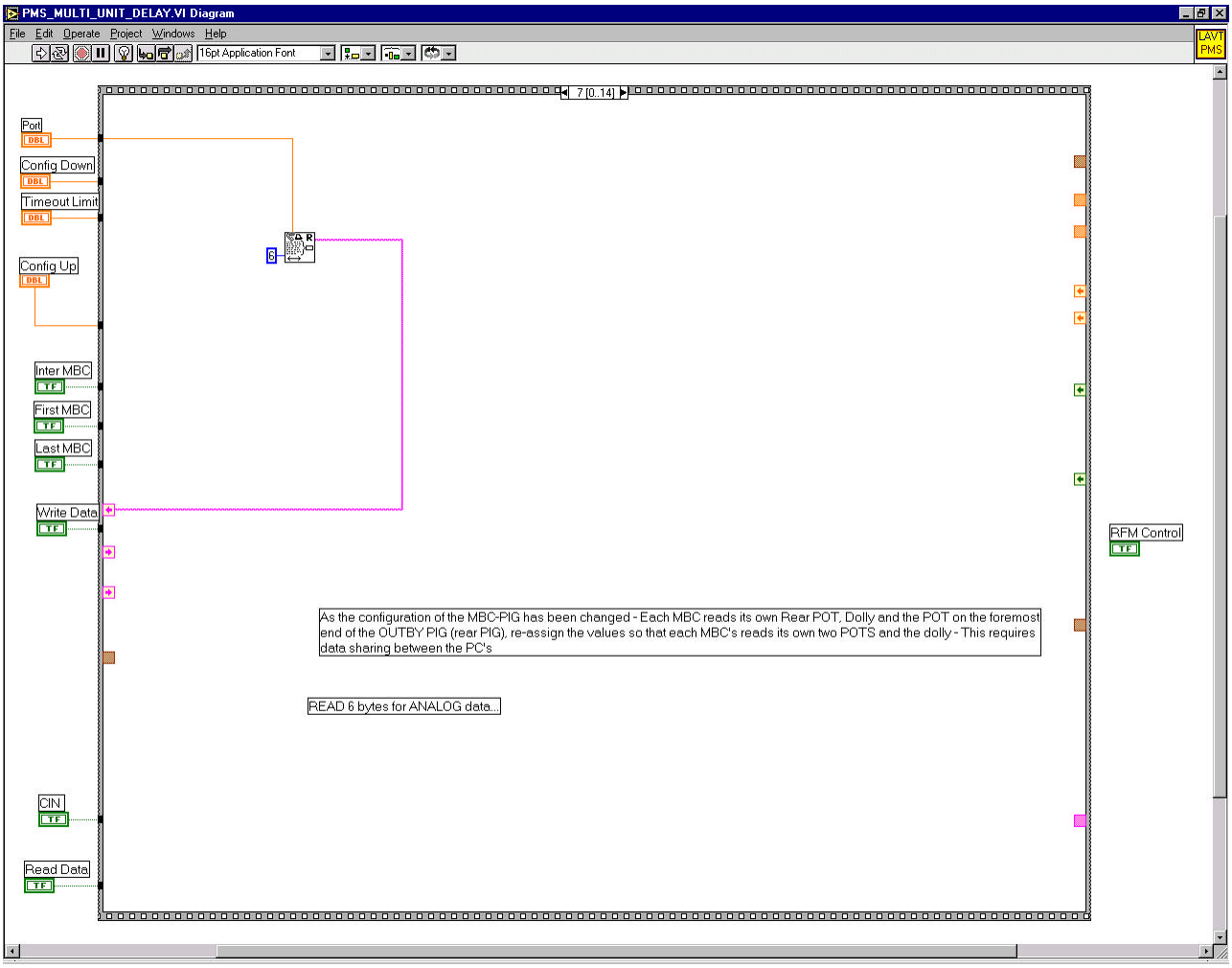


Fig. 9. Frame 7 of 14 – PMS_MULTI_UNIT_DELAY.VI

This frame reads the analog data requested in the previous frame. The frame reads 6 bytes, 2 for each of the analog measurements.

Frame 8 – Convert analog data and display

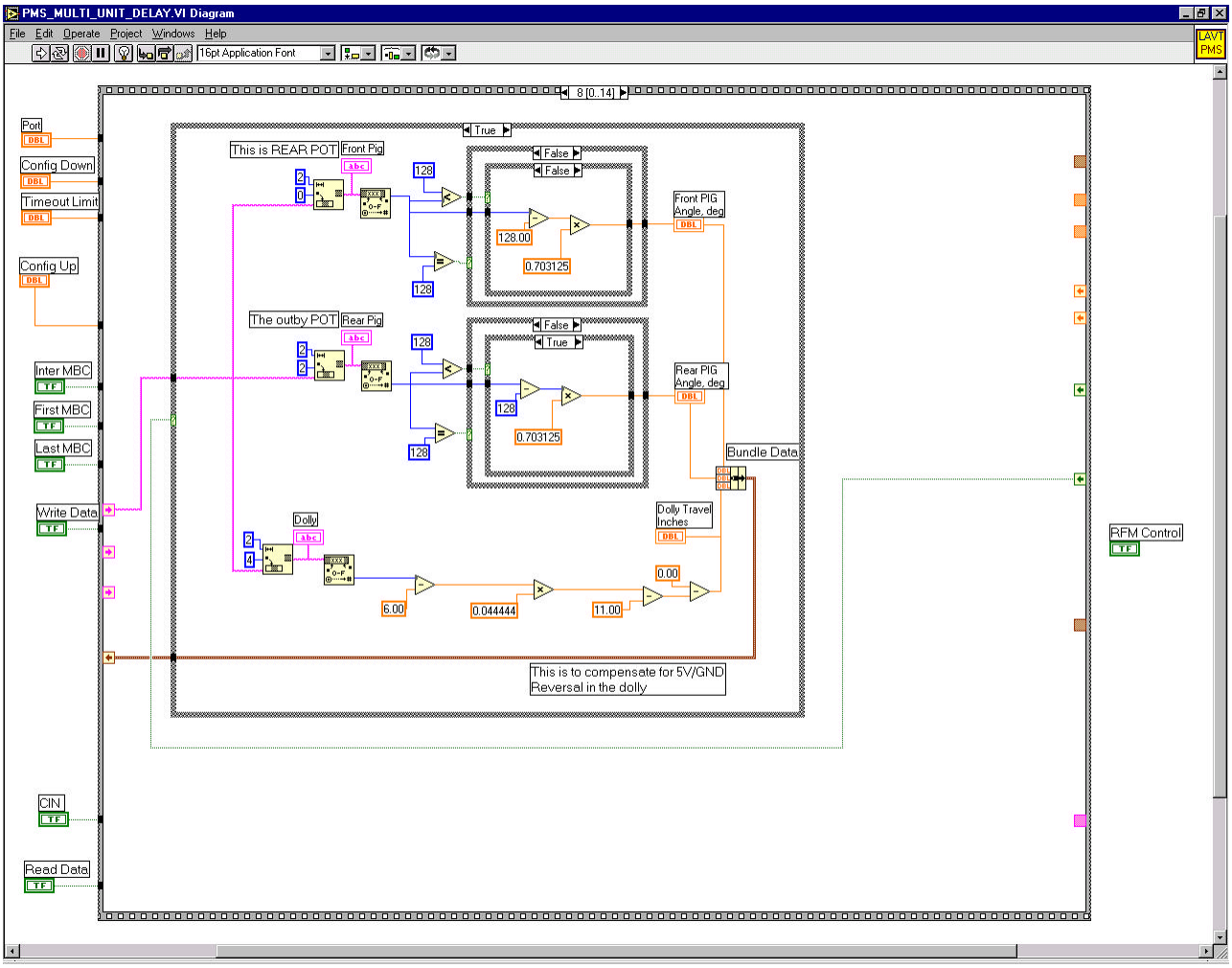


Fig. 10. Frame 8 of 14 – PMS_MULTI_UNIT_DELAY.VI

This frame illustrated above converts the input hexadecimal string into a number, and displays the values on the indicators in the front panel. Details on data conversions can be found in the final document created by Bruce Wells [3 – References, Vol I].

Frame 9 – Read serial data – Inter MBC data

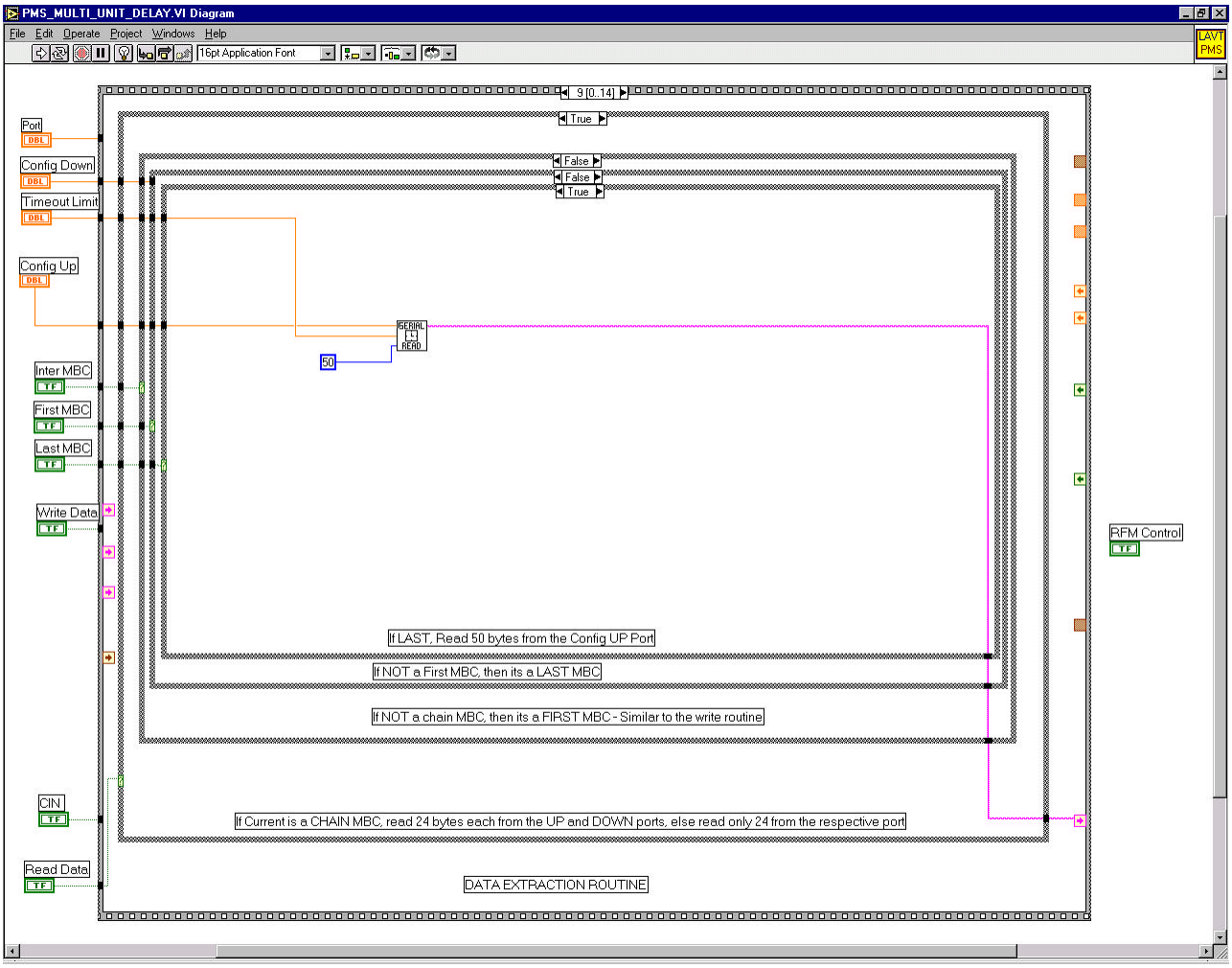


Fig. 11. Frame 9 of 14 – PMS_MULTI_UNIT_DELAY.VI

If the ‘Read Data’ button is in ON state on the front panel, this frame reads the inter-MBC data sent by the other host controller. Based on the current condition of the MBC configuration selector variables, the frame either reads 100 bytes for an intermediate MBC, and 50 bytes if the operating unit is a First MBC or the Last MBC. The serial data is read from the two transfer ports as controlled by the port numbers in the ‘Config Up’ and ‘Config Down’ port controls on the front panel.

Frame 10 – Convert and display inter-MBC data

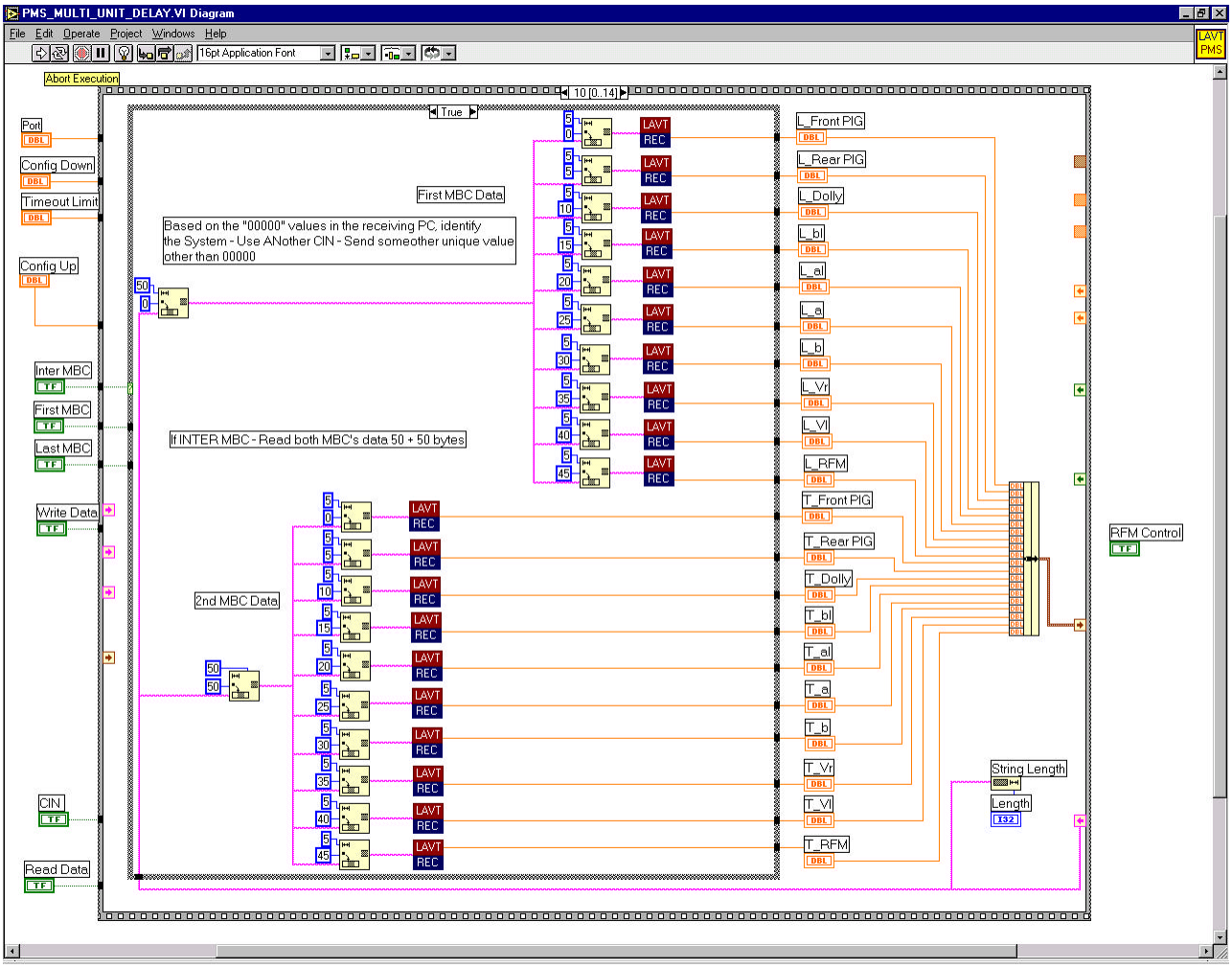


Fig. 12. Frame 10 of 14 – PMS_MULTI_UNIT_DELAY.VI

This frame employs the 'LAVT Rec' Sub-VI for converting the input hexadecimal string into a floating point number. This frame displays the data from the peer units onto the front panel indicators.

Frame 11 – CIN frame

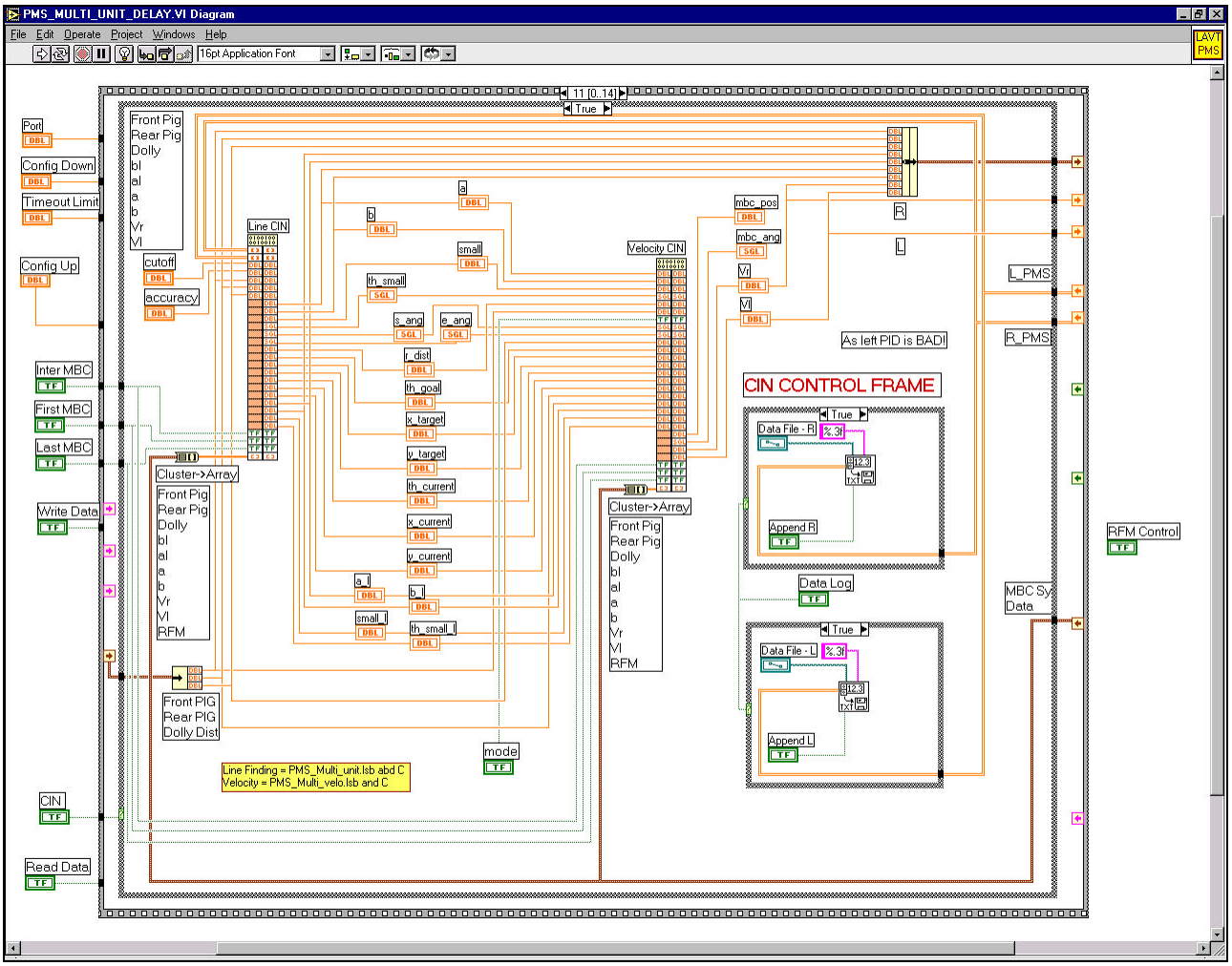


Fig. 13. Frame 11 of 14 – PMS_MULTI_UNIT_DELAY.VI

This is the frame that implements the ‘Line-finding’ and the ‘Motion control’ algorithms via the use of Code Interface Nodes. The left CIN implements the line-finding algorithm, using the scanner data for self-localization whereas the right CIN implements the SSS algorithm, or the simple motion control algorithm for controlling the track speeds. It can also be observed that the scanner data storage to a data file is performed in this frame. The object codes for the line-finding and motion control CIN are ‘pms_multi_unit.lsb’ and ‘pms_multi_velo.lsb’ respectively.

Frame 12 – PID controls frame

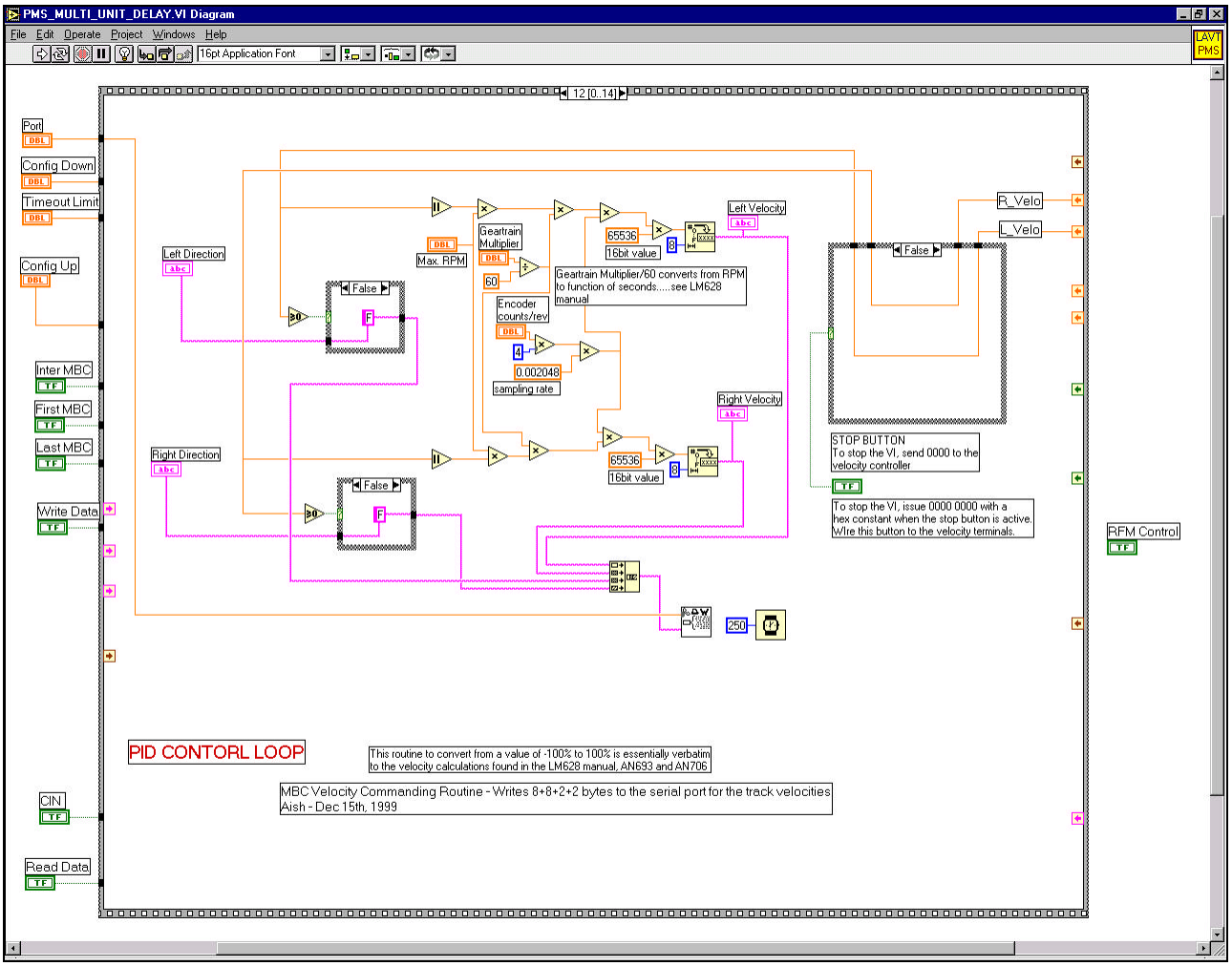


Fig. 14. Frame 12 of 14 – PMS_MULTI_UNIT_DELAY.VI

This frame is a direct implementation of the PID controls frame developed by Bruce Wells. More details regarding the creation of this frame can be found in the manual created by Wells [3 – References, Vol I).

Frame 13 – Implement 'Wait and Go' driving strategy

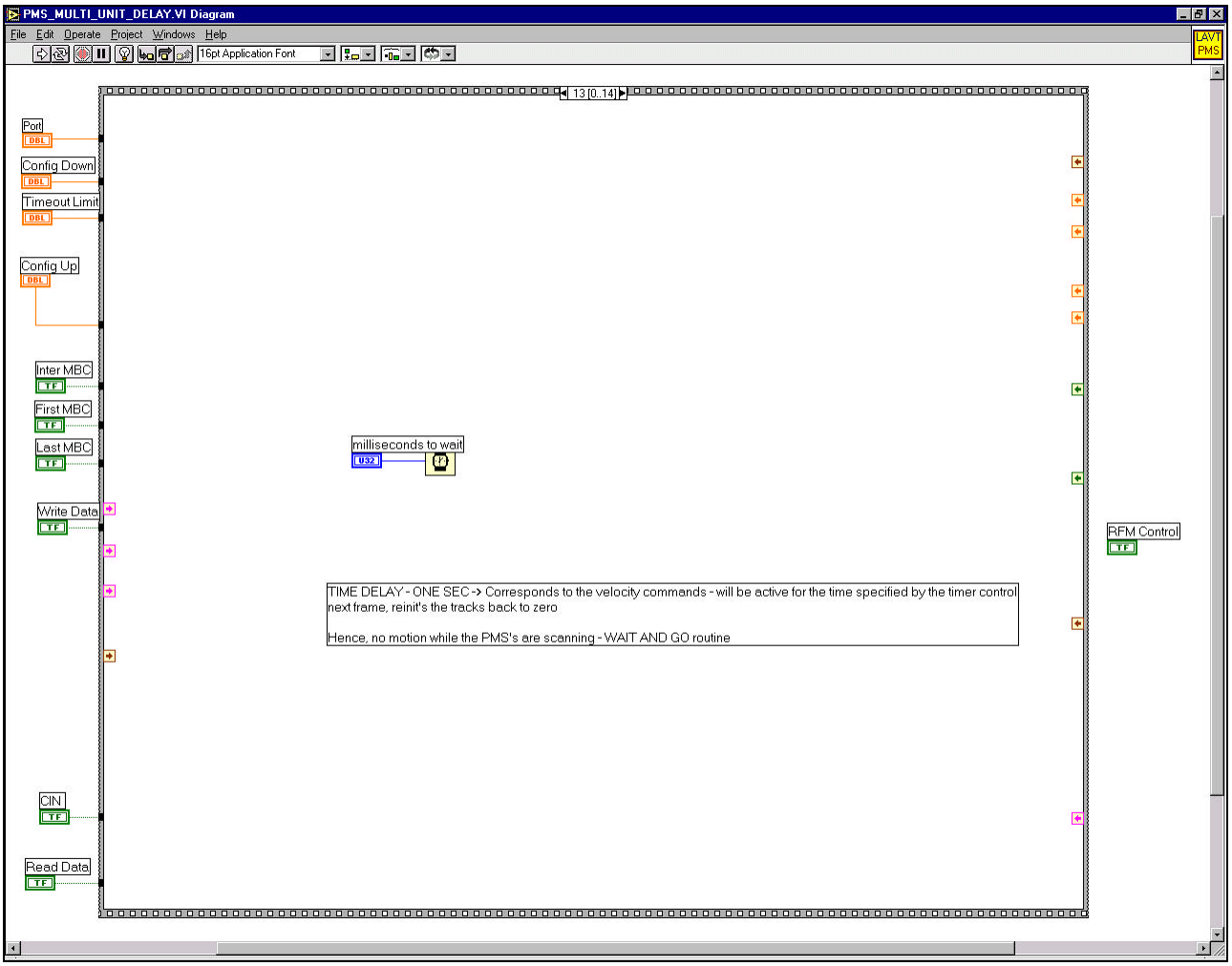


Fig. 15. Frame 13 of 14 – PMS_MULTI_UNIT_DELAY.VI

This frame implements the 'Wait and Go' strategy of motion control. The frame supplies delay to the on-board controller to keep the tracks active for the amount of time specified by the 'milliseconds to wait' control on the front panel.

Frame 14 – Send MBC data to other MBC host controllers

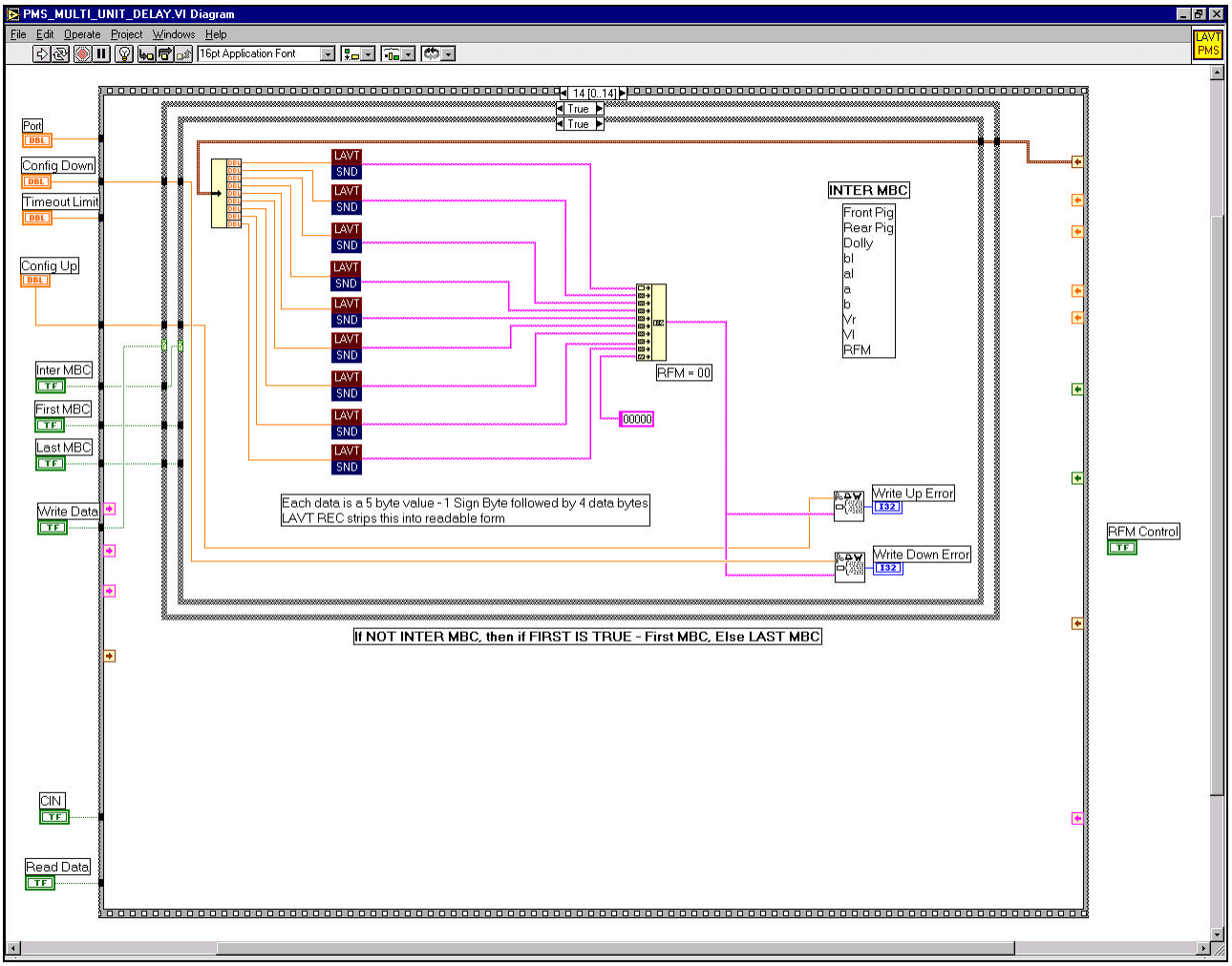


Fig. 16. Frame 14 of 14 – PMS_MULTI_UNIT_DELAY.VI

This final frame of the VI initiates the PC-PC data transfer, by writing the operating unit's data to the other PC controllers. The frame employs the Sub-VI 'LAVT SND" VI for creating the transfer packet (section 6.2.3, pg 58, Vol I).

iii. The data sharing SUB-VI's

This section illustrate the block diagram of the two SUB-VI's created to enable PC-PC data sharing between the MBC host controllers. These VI's are called in the main VI and they interpret and create the transfer byte packet for the MBC data. Only the block diagrams are illustrated as they are self explanatory in their function.

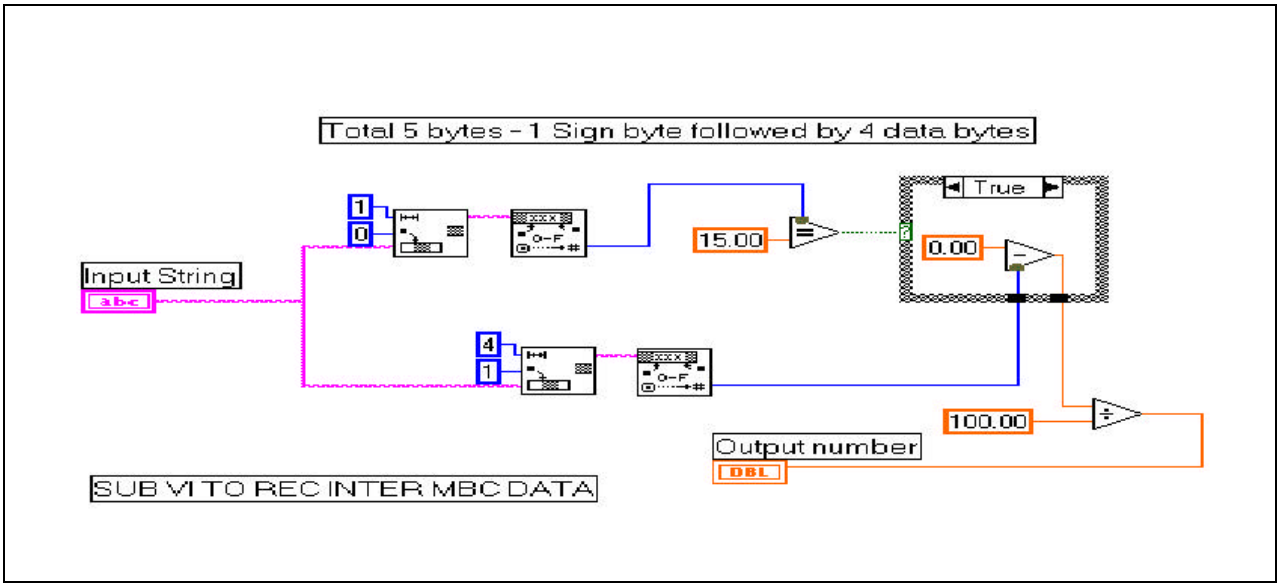


Fig. 17. LAVT REC Sub-VI Block Diagram

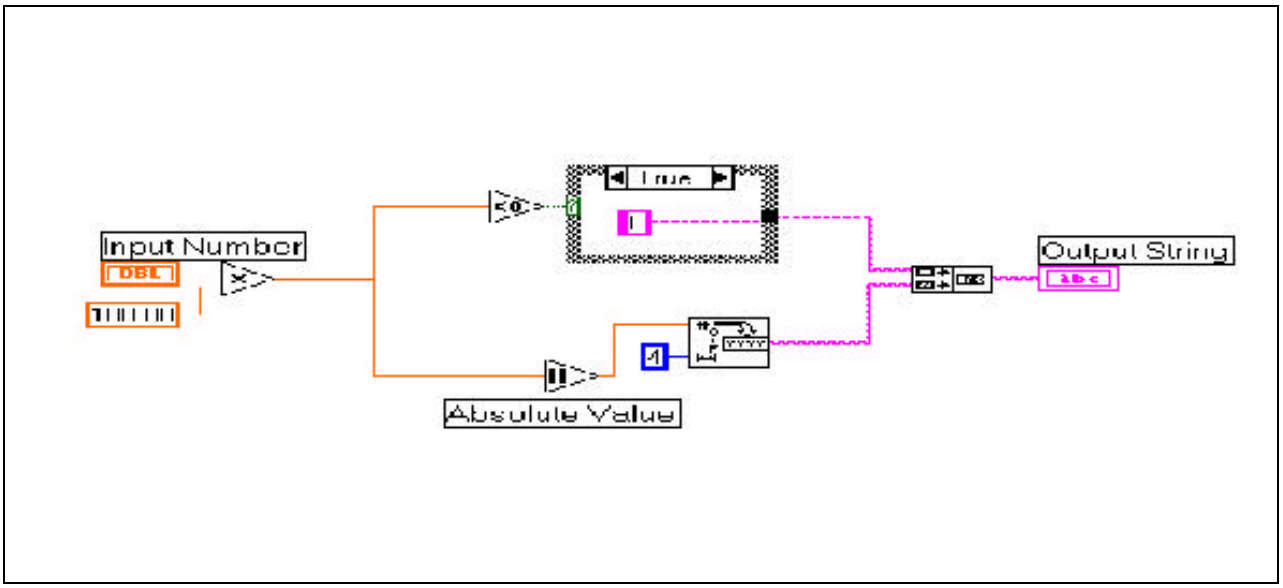


Fig. 18. LAVT SND Sub-VI Block Diagram

iv. Pin-out diagram – RS422 NI PCMCIA Card, NULL MODEM Connection

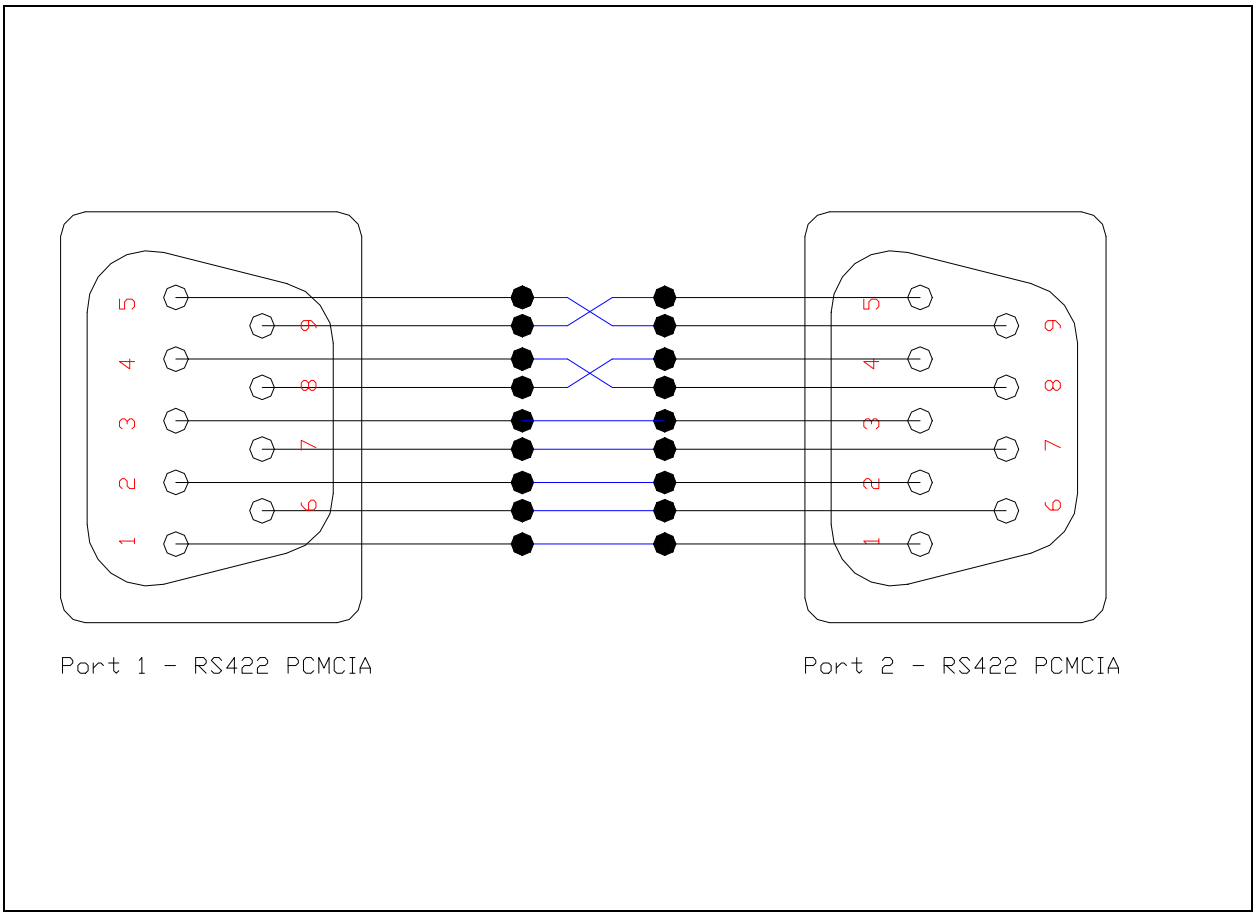


Fig 19. NULL MODEM Pin-out for the RS422 PCMCIA Serial card

APPENDIX III

This section focuses on the CIN object codes developed for the automation of the LACHS. The source code listing is provided, accompanied by a brief description of the functional and operational aspects of the code.

This section discusses the object codes developed for both the SICK and the PMS laser scanners, along with the two motion control strategies. Hence, four code listings are provided. The codes discussed in this section are…

i. SICK line-finding – 'Simple line-finding'
ii. Code modifications to accommodate the PMS laser scanner
iii. Code listing of the 'Iterative end point line splitting technique'
iv. Simple motion control & SSS Algorithm

The discussion for all these codes follows the same format as described below…

i. Brief Description of the code with the algorithm being implemented
ii. Code Structure
iii. Variable list
iv. Function list
v. Complete code listing

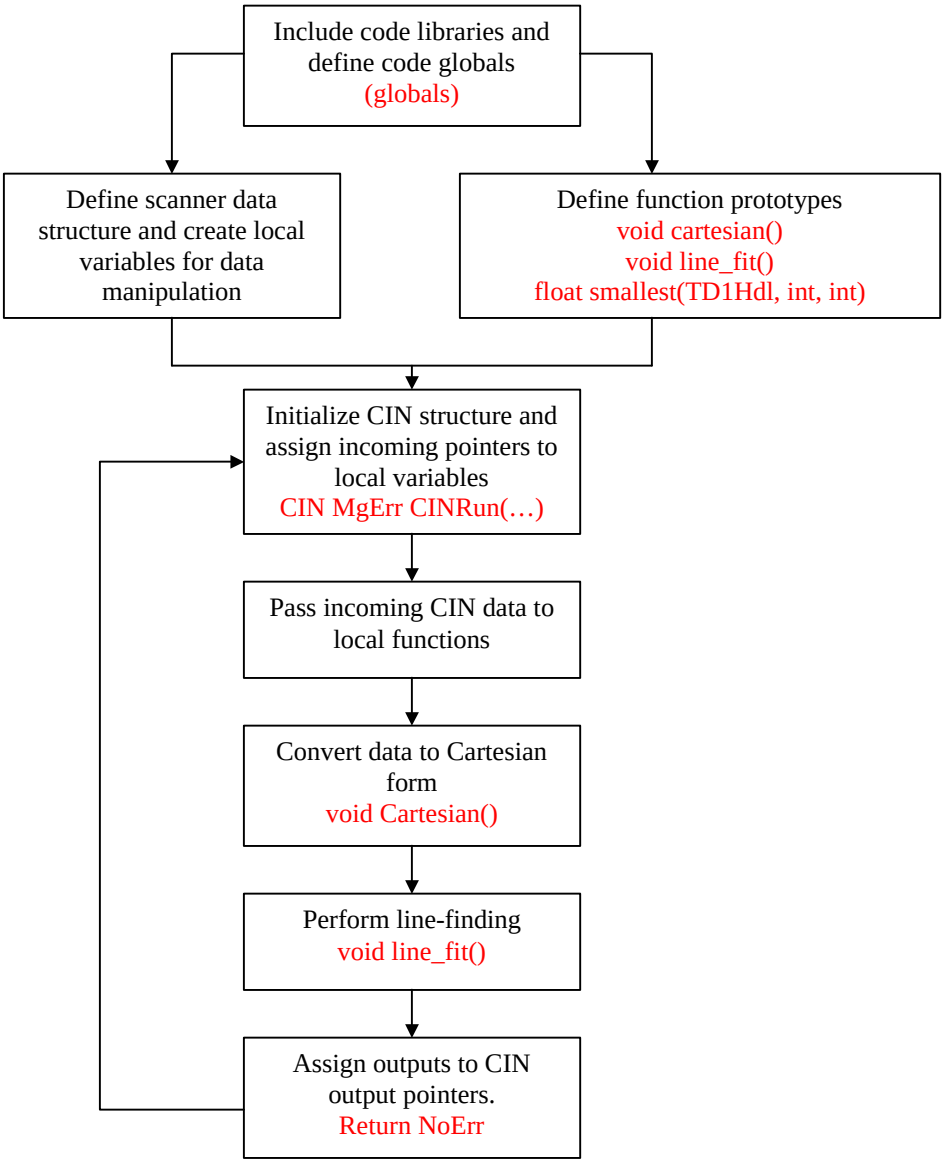
The above described format is followed for all the source code, except for the listing of the 'iterative end point line splitting technique'. Only THE source code for this implementation is listed as the code is still under development.

The following sections provide the documentation for the CIN object codes.

i. Simple line-finding implemented for self localization using the SICK® LMS200 data

This object code performs the implementation of the simple line-finding developed for the self-localization of the MBC using the SICK laser scanner. The code implements the initial line-finding developed termed as 'Simple line-finding'. The algorithm employs two range values for creating a usable window, from which the line finding computes the equations of the line representing the mine walls (Section 7.1.3. pg 70, Vol I).

Code structure



Variable list

Variable name	Data type	Declaration	Comments
i	Int	Global	Index variable
x[200]	Float	Global	Scanner X data
y[200]	Float	Global	Scanner Y data
len	Int	Global	Length of data array
k	Int	Global	Index variable
acc	Int	Global	Start index
cut	Int	Global	End index
r_small	float	Global	Smallest range
a_small	float	Global	Corresponding angle
sum_x	float	Global	Line parameter
sum_y	float	Global	Line parameter
sum_xy	float	Global	Line parameter
sum_xsq	float	Global	Line parameter
al	float	Global	Slope of line
bl	float	Global	Intercept of line
TD1	struct	Global	Scanner data struct
n_rows	int	Global	No of data rows
n_cols	int	Global	No of data columns
cin_data	TD1Hdl	Global	Scanner data variable
prop	int32	CINProperties(…)	--
data	void *	CINProperties(…)	--
output	float	Smallest(…)	Smallest range

Note: The LabVIEW CIN library compilation requires the use of the standard header file supplied by National Instruments. This file has to be included with the code headers for successful compilation of the code resource file. The file is ‘extcode.h’ and can be located in the path to ~/LabView.

Function list

Function name	Argument	Return value & type	Comments
cartesian	Global Scanner data	void, Global assignment to x[] & y[]	Converts the raw scanner data to Cartesian form
line_fit	Global scanner data	void, global assignments to al & bl	Performs line finding on the converted data
smallest	TD1Hdl in, int rows, int cols	float, r_small, a_small	Searches the scanner data array and returns the smallest range and corresponding angle
CINProperties	int32 prop, void *data	CIN MgErr	Converts the CIN code to an re-entrant code
CINRun	TD1Hdl var1, float32 *cutoff, float64 *accuracy, float64 *a, float64 *b, float64 *small, float32 *th_small, float32 *s_ang, float32 *e_ang	CIN MgErr	The main input-output function. This is the CIN call back routine that processes all the I/O operations.

Note: The main CIN processing function, the CINRun function will always return a LabVIEW NoErr message confirming the proper execution of the CIN object code. This value will force the CIN object code to return to the block diagram. Hence, if the error trap is not included, the object code will not return to the calling block diagram and will result in the ‘re-setting’ of the VI.

Complete Code listing

```
//-------------------------------------------------------------------------
//          DATA FILTERING & LINE FINDING FOR THE SICK LASER SCANNER
//
//AUTHOR          : Aish - Spring 1999
//DEPT            : LA - VT - Line Finding Algorithm for SICK Laser Scanner
//                : May 19, 1999 - NEB 122.
//
//SUPPORT         : Mike's Corner location Algorithm
//                : Bruce's Labview Data Acquisition code
//-------------------------------------------------------------------------

//ALGORITHM

//1. Use the distance to angle difference ratio to look at the data
//2. Then, perform grouping and line fitting
//3. OR - Restrict the range of the sick

#include "extcode.h"
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<iostream.h>
#include<conio.h>


#define pi 3.14159


//global variable declarations

int i;
float x[200];                       //global x
float y[200];                       //global y
int len;                            //data length (every 1 deg)

//GLOBAL DATA FOR THE ALIAS METHOD..

int k;
int no_data;                        //no of data points forming a line
int acc;                            //start angle index
int cut;                            //end index
float r_small;                      //smallest range
float a_small;                      //corresponding angle


//GLOBAL DATA FOR THE LINE PARAMETERS

int start;                          //start index of line
int end;                            //end index of line
float sum_x;                        //total of x
float sum_y;                        //total of y
float sum_xsq;                      //total of x squares
float sum_xy;                       //total of xy
```

```
float al;                           //slope of line
float bl;                           //intercept of line;

//CIN data passing structure

typedef struct
{
      int32 dimSizes[2];            //rows X cols --> rows = len = data size
      float64 arg1[1];              //first col = angle, second col = range
} TD1;

typedef TD1 **TD1Hdl;

//CIN global variables

int n_rows;                         //no of rows = length of data set
int n_cols;                         //is always 2
TD1Hdl cin_data;                    //data handle to SICK data set

//function prototypes

void cartesian();                   //convert data to cartesian coord


//function to perform regression fit

void line_fit();

//function to return the smallest range and coresponding angle

float smallest(TD1Hdl in, int rows, int cols);

//CIN function prototype - call all functions in this routine...

CIN MgErr CINProperties (int32 prop, void *data);


CIN MgErr CINRun( TD1Hdl var1,
                  float32 *cutoff,
                  float64 *accuracy,
                  float64 *a,
                  float64 *b,
                  float64 *small,
                  float32 *th_small,
                  float32 *s_ang,
                  float32 *e_ang);

//beginning of main routine

CIN MgErr CINProperties(int32 prop, void *data)
{
      switch(prop)
      {
      case kCINIsReentrant:
            *(Bool32*)data = TRUE;
            return noErr;
      }
```

```
	return mgNotSupported;

}


CIN MgErr CINRun( TD1Hdl var1,
                  float32 *cutoff,
                  float64 *accuracy,
                  float64 *a,
                  float64 *b,
                  float64 *small,
                  float32 *th_small,
                  float32 *s_ang,
                  float32 *e_ang)
{



	//Assign the incoming data to cin_data

	cin_data=var1;

	n_rows = (*cin_data)->dimSizes[0];
	n_cols = (*cin_data)->dimSizes[1];

	//Assign n_rows to data length = len

	len=n_rows;

//	acc=*accuracy;
//	cut=*cutoff;

	start=*accuracy;
	end=*cutoff;

	//call the other functions

	cartesian();                                //converts data to x / y

	//call line finding routine...

	line_fit();

	//call the smallest range/angle routine

	r_small= smallest(cin_data, n_rows, n_cols);

	//Assign the function return values to CIN output variables...

	*a = al;
	*b = bl;
	*small = r_small;
	*th_small = a_small;
	*s_ang = start;//start;
	*e_ang = end;//end;
```

```
	return noErr;
}


//function to convert data to cartesian coordinates

void cartesian()
{
	for(i=0;i<len;i=i++)
	{
		x[i]=(float)(((*cin_data)->arg1[i*n_cols+1])*(cos(((*cin_data)-
>arg1[i*n_cols+0])*pi/180)));
		y[i]=(float)(((*cin_data)->arg1[i*n_cols+1])*(sin(((*cin_data)-
>arg1[i*n_cols+0])*pi/180)));
	}
}



//REGRESSION FIT TO DATA

void line_fit()
{
	sum_x=0;
	sum_y=0;
	sum_xsq=0;
	sum_xy=0;

	no_data=(end-start)+1;

	for(k=start;k<=end;k++)
	{
		sum_x = sum_x + x[k];
		sum_y = sum_y + y[k];
		sum_xsq = sum_xsq + (x[k]*x[k]);
		sum_xy = sum_xy + (x[k]*y[k]);
	}

	//compute the line parameters

	al=((no_data*sum_xy)-(sum_x*sum_y))/((no_data*sum_xsq)-(sum_x*sum_x));
	bl=(sum_y - (al*sum_x))/(no_data);

}

//function to determine the smallest range and corresponding angle

float smallest(TD1Hdl in, int rows, int cols)
{
	float output;

	output=(*in)->arg1[1];  //assign to the first range - arg1[1]

	for(i=0;i<rows;i++)
	{
```

```
            if(output>((*in)->arg1[i*cols + 1]))
            {
                  output=(*in)->arg1[i*cols + 1];
                  a_small = i;
            }

      }

      return output;
}
```

ii. PMS Line-finding

As the PMS is very similar in function to the SICK LMS 200 scanner, the data handling for the PMS does not require any additional handling apart from the addition of a function to convert the data set from a custom angle index to 0-180 index. The other main difference is the reduced data set of the PMS data which is automatically handled by the CIN object code. Adding the function listed below to the CIN code for the SICK (as discussed in the previous section) will prepare the code for use on the PMS.

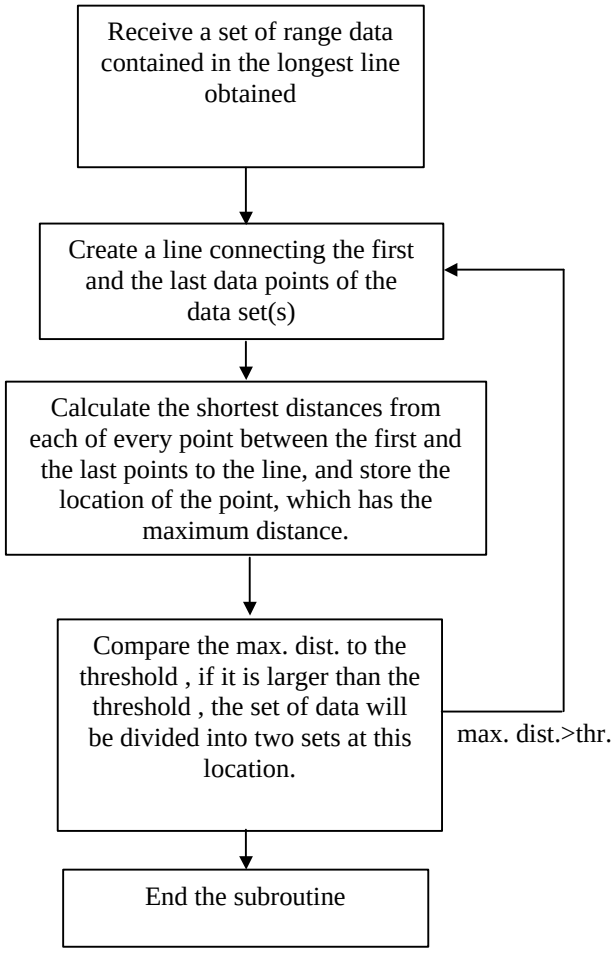
```
void angle_convert()
{
	for(i=0;i<len;i++)
	{
		angle[i]   = angle[i]-a; // To convert the angles back to 0->180
		angle_l[i] = angle[i]-b; // For the left scanner
	}
}
```

Note : ‘a’ and ‘b’ are the values specified in the controls ‘R-ang’ and ‘L-ang’ in the ‘PMS Start angle’ controls console in the front panel of the VI.

iii. Iterative end point line-splitting technique

This code implements the iterative end point line-splitting technique. As this code is currently under development, only the code structure and the code listing is provided. The code structure illustrated below is a direct representation of the recursive line-splitting technique as described in Section 7.1.4 of Vol I.

Code Structure



Complete Code listing

```
//-------------------------------------------------------------------------
//    MULTI UNIT MBC CONTROL ALGORITHM USING THE PMS FOR NAVIGATION
//
//
//AUTHOR          : Aish & Mike - Fall 1999
//DEPT            : LA - VT - Line Finding Algorithm for SICK Laser Scanner
//                : January 12, 2000, NEB 122.
//
//SUPPORT         : The iterative end point line splitting technique
//                : Bruce's Labview Data Acquisition code
//
//MODIFIED        : From the original file THE SICK LINE FINDING
//                : To include the PMS
//                : STRUGGLED with the PMS on March 28th, 2000. - Aish ☺
//
//-------------------------------------------------------------------------

#include "extcode.h"
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<iostream.h>
#include<conio.h>


#define pi 3.14159
#define max_radius 20              //8.1 meters scaled down 1:10
#define max_no_line 41

//global variable declarations

int i;
float x[60];                       //global x
float y[60];                       //global y
float x_l[60];                     //globals for the left sick
float y_l[60];                     //the globals for the left sick
int len;                           //data length
float angle[60];                   //the angle from the sick
float radius[60];                  //the range from the sick
float angle_l[60];                 //for the left sick
float radius_l[60];                //for the left sick
int temp_index;                    //temp index for the segment row #
float ref_dist;                    //current position w.r.t refernce point
float temp_rdist;                  //r_dist value from the other CIN
float lin_angle;

//GLOBAL DATA FOR THE ALIAS METHOD..

//float d_ratio[200];              //the ratio of (i+1)/i data point
//float a_ratio[200];              //the angle ratio
//int st_index[100];               //start coordinates
//int tmpr;
int k;
```

```
//int m=0;
//int l;
int no_data;                        //# of data pt in the group forming a line
int acc;                            //wall surface roughness for data
int cut;                            //alias data range cutoff
float r_small_r;                    //smallest range
float a_small;                      //corresponding angle - a global argument
float r_small_l;                    //for the left SICK
float a_small_l;                    //for the left SICK
float a_small_r;                    //for the right SICK


//GLOBAL DATA FOR THE LINE PARAMETERS

int start;                          //start index of line
int end;                            //end index of line
float sum_x;                        //total of x
float sum_y;                        //total of y
float sum_xsq;                      //total of x squares
float sum_ysq;
float sum_xy;                       //total of xy
float al;                           //slope of line
float bl;                           //intercept of line;
float al2;
float bl2;
float al_l;                         //for the left sick
float bl_l;                         //for the left sick
float theta_pig;
int segment[5][5];
int occ_corner;                     //this parameter distinguishes between
                                    //zone#1 and #2
float x_ref;
float y_ref;
float x_pos;
float y_pos;
float th_go;                        //the angle to go next
float dist_go;                      //the next target
float xc;                           //the center coordinates of the arc
float yc;
float xd;                           //the arc center distance - a standard
                                    //from the big white chart
float yd;
float R_mbc;                        //the distance of the MBC fron the center
                                    //(xc,yc)
float th_cen_mbc;                   //the angle of the MBC w.r.t center
                                    //(xc,yc)

//Global data for the line_fit

float x_cur;
float y_cur;
float th_cur;
float x_tar;
float y_tar;
float th_tar;
float slider;
```

```
//parameters for fit two lines loop

int start1;
int end1;
int start2;
int end2;
float angle_btw_line;
double case_no;
float max_length;                  // the max. length from start to end angle
                                   //points.


// GLOBAL DATA FOR INTER MBC DATA..

float UP_front_pig;                // Front PIG angle of CHAIN UP MBC
float UP_rear_pig;                 // Rear PIG angle of CHAIN UP MBC
float UP_dolly;                    // Dolly distance of the CHAIN UP
float UP_bl;                       // Line data of CHAIN UP MBC - Left Side
float UP_al;
float UP_ar;                       // Line data of CHAIN UP MBC - Right Side
float UP_br;
float UP_vr;                       // Right track velocity of CHAIN UP MBC
float UP_vl;                       // Left track velocity of CHAIN UP MBC
float UP_RFM;                      // CHAIN UP RFM DATA - sounds crazy, but
                                   //to maintain simplicity in programming:-)

float DOWN_front_pig;              // Front PIG angle of CHAIN DOWN MBC
float DOWN_rear_pig;               // Rear PIG angle of CHAIN DOWN MBC
float DOWN_dolly;                  // Dolly distance of the CHAIN DOWN
float DOWN_bl;                     // Line data of CHAIN DOWN MBC - Left Side
float DOWN_al;
float DOWN_ar;                     // Line data of CHAIN DOWN MBC - Right
float DOWN_br;
float DOWN_vr;                     // Right track velocity of CHAIN DOWN MBC
float DOWN_vl;                     // Left track velocity of CHAIN DOWN MBC
float DOWN_RFM;                    // CHAIN DOWN RFM DATA

// Boolean control variables...

LVBoolean Inter;                   // Intermediate MBC Boolean data - use
                                   //these to control the above variable
                                   //values
LVBoolean First;                   // Lead MBC Boolean
LVBoolean Last;                    // Rear MBC Boolean



float theta;

//CIN data passing structure

typedef struct
{
      int32 dimSizes[2];           // rows X cols --> rows = len = data size
      float64 arg1[1];             // first col = angle, second col = range
} TD1;
```

```
typedef TD1 **TD1Hdl;               // Creata a 'handle' to the sructrure...
                                    // Pointer to a pointer is handle.


// MBC Data Array struct

typedef struct {
      int32 dimSize;
      float64 Output_number[1];
      } TD2;

typedef TD2 **TD2Hdl;



//CIN global variables

int n_rows;                         //no of rows = length of data set
int n_cols;                         //is always 2
TD1Hdl cin_data_r;                  //data handle to SICK data - right sick
TD1Hdl cin_data_l;                  //data handle to SICK data - left sick

//MBC Data global variables..

TD2Hdl MBC_data;                    // Data Handle for the input array
int ip_size;                        // size of the input data - always be 20
int data_rows;
int data_cols;


//function prototypes

void read_file();                   //read SICK data file
void sort();                        //overwrites the sick data range
void angle_convert();               // To convert PMS angles into 0->180 aray
void cartesian();                   //convert data to cartesian coord
void MBC();                         //reads the incoming data and performs
                                    //global variable assignment
void grouping(int SIDE);
float check_theta(float th);

//function to perform regression fit

void line_fit();

void line_fit_left();               //line fitting for the left sick

void test_r();                      // Simple line finding – Right Scanner
void test_l();                      // Simple line finding – Left scanner

//function to return the smallest range and coresponding angle

float smallest(TD1Hdl in, int rows, int cols, int side);

//CIN function prototype - call all functions in this routine...

CIN MgErr CINProperties (int32 prop, void *data);
```

```
CIN MgErr CINRun(TD1Hdl var1,
                         TD1Hdl var2,
                         float64 *cutoff,
                         float64 *accuracy,
                         float64 *var5,
                         float64 *var6,
                         float64 *var7,
                         float64 *a,
                         float64 *b,
                         float64 *small,
                         float32 *th_small,
                         float32 *s_ang,
                         float32 *e_ang,
                         float64 *r_dist,
                         float64 *th_goal,
                         float64 *x_target,
                         float64 *y_target,
                         float64 *th_current,
                         float64 *x_current,
                         float64 *y_current,
                         float64 *a_l,
                         float64 *b_l,
                         float64 *small_l,
                         float64 *th_small_l,
                         LVBoolean *Inter_MBC,
                         LVBoolean *First_MBC,
                         LVBoolean *Last_MBC,
                         TD2Hdl var28);

//beginning of main routine

CIN MgErr CINProperties(int32 prop, void *data)
{
      switch(prop)
      {
      case kCINIsReentrant:
            *(Bool32*)data = TRUE;
            return noErr;
      }

      return mgNotSupported;

}

CIN MgErr CINRun(TD1Hdl var1,
                         TD1Hdl var2,
                         float64 *cutoff,
                         float64 *accuracy,
                         float64 *var5,
                         float64 *var6,
                         float64 *var7,
                         float64 *a,
                         float64 *b,
                         float64 *small,
                         float32 *th_small,
                         float32 *s_ang,
```

```
                        float32 *e_ang,
                        float64 *r_dist,
                        float64 *th_goal,
                        float64 *x_target,
                        float64 *y_target,
                        float64 *th_current,
                        float64 *x_current,
                        float64 *y_current,
                        float64 *a_l,
                        float64 *b_l,
                        float64 *small_l,
                        float64 *th_small_l,
                        LVBoolean *Inter_MBC,
                        LVBoolean *First_MBC,
                        LVBoolean *Last_MBC,
                        TD2Hdl var28)
{


      //Assign the incoming data to cin_data

      cin_data_r = var1;            // Right SICK data assignment
      cin_data_l = var2;            // Left SICK data assignment

      // Assign the MBC Data to MBC_Data..

      MBC_data   = var28;                 // Input MNC Data assignment
      ip_size    = (*MBC_data)->dimSize;  // the depth of the array ALWAYS 20
      Inter    = *Inter_MBC;
      First    = *First_MBC;
      Last     = *Last_MBC;



      n_rows = (*cin_data_r)->dimSizes[0];
      n_cols = (*cin_data_r)->dimSizes[1];

      //Assign n_rows to data length = len

      len=n_rows;                   //the length of the SICK dat array...

      acc=*accuracy;
      cut=*cutoff;

      theta_pig=*var5;
      slider=*var7;

      //NOTE : *var5 is front pig angle and *var6 is rear pig angle of MBC # 2
      //*var7 is the dolly distance

      //call the other functions - performs computation for both the sick's
      //together.....

      read_file();                  //copy incoming data to angle[i]&range[i]
      //sort();
```

```
      angle_convert();              //Call fn to convert angle to 0->180
      cartesian();                  //Converts data to x / y
      MBC();                        //Call to the data handling function....

      //call the smallest range/angle routine for the right SICK

      r_small_r = smallest(cin_data_r, n_rows, n_cols, 1);
      a_small_r = a_small;

      //Call function for the LEFT SICK

      r_small_l = smallest(cin_data_l, n_rows, n_cols, 2);
      a_small_l = a_small;

      //call line finding routine... for the right SICK - create a global
      //access function for both left and right SICK's
      //perform data filtering for each side of SICK ( 1 = Right Side; 2 =
      //Left Side) and fit the line

      grouping(2);
      line_fit_left();
      grouping(1);
      line_fit();

      test_r();                     // call the test line finding fn......
      test_l();


      //Assign the function return values to CIN output variables...

      *a = al;
      *b = bl;
      *a_l = al_l;
      *b_l = bl_l;
      *small_l = r_small_l;
      *th_small_l = a_small_l;
      *small = r_small_r;
      *th_small = a_small_r;
      *s_ang = start1;
      *e_ang = end1;
      *r_dist= ref_dist;
      *th_goal = th_tar;
      *x_target = x_tar;
      *y_target = y_tar;
      *th_current = th_cur;
      *x_current = x_cur;
      *y_current = y_cur;

      return noErr;
}


//define the member functions
```

```
//Function read_file reads data from the SICK data file into R,THETA

void read_file()
{

      for(i=0;i<len;i++)
      {
      // To access the element in the ith row and jth column in an array with
      //'c' columns
      // element = array[i*c + j]

            angle[i]=(*cin_data_r)->arg1[i*n_cols + 0];
            radius[i]=(*cin_data_r)->arg1[i*n_cols + 1];

            //DO THE SAME COMPUTATION FOR THE OTHER SICK

            angle_l[i]=(*cin_data_l)->arg1[i*n_cols + 0];
            radius_l[i]=(*cin_data_l)->arg1[i*n_cols + 1];
      }
}


//function to sort the sick data based on the max_radius

void sort()
{
      //restrict the scope of the sick to maximum radius".
      for(i=0;i<len;i++)
      {
            if(radius[i]>max_radius)      //if > 25 = 25, else, retain as such
                  radius[i]=max_radius;

            //do the same for the other sick

            if(radius_l[i]>max_radius)
                  radius_l[i]=max_radius;

      }
}

void angle_convert()
{
      for(i=0;i<len;i++)
      {
            angle[i]   = angle[i];  // convert the angles to 0->180 array form
            angle_l[i] = angle[i];
      }
}



//function to convert data to cartesian coordinates

void cartesian()
{
      for(i=0;i<len;i++)
      {
```

```
            x[i]=(float)(((*cin_data_r)-
>arg1[i*n_cols+1])*(cos(((*cin_data_r)->arg1[i*n_cols+0])*pi/180)));
            y[i]=(float)(((*cin_data_r)-
>arg1[i*n_cols+1])*(sin(((*cin_data_r)->arg1[i*n_cols+0])*pi/180)));

            //perform the same computation for the left sick..

            x_l[i]=(float)(((*cin_data_l)-
>arg1[i*n_cols+1])*(cos(((*cin_data_l)->arg1[i*n_cols+0])*pi/180)));
            y_l[i]=(float)(((*cin_data_l)-
>arg1[i*n_cols+1])*(sin(((*cin_data_l)->arg1[i*n_cols+0])*pi/180)));

      }


}

void MBC()
{
      // assign the MBC data to the global variables...

      UP_front_pig      = (*MBC_data)->Output_number[0];
      UP_rear_pig       = (*MBC_data)->Output_number[1];
      UP_dolly          = (*MBC_data)->Output_number[2];
      UP_bl             = (*MBC_data)->Output_number[3];
      UP_al             = (*MBC_data)->Output_number[4];
      UP_ar             = (*MBC_data)->Output_number[5];
      UP_br             = (*MBC_data)->Output_number[6];
      UP_vr             = (*MBC_data)->Output_number[7];
      UP_vl             = (*MBC_data)->Output_number[8];
      UP_RFM            = (*MBC_data)->Output_number[9];

      // Trail MBC Data

      DOWN_front_pig    = (*MBC_data)->Output_number[10];
      DOWN_rear_pig     = (*MBC_data)->Output_number[11];
      DOWN_dolly        = (*MBC_data)->Output_number[12];
      DOWN_bl           = (*MBC_data)->Output_number[13];
      DOWN_al           = (*MBC_data)->Output_number[14];
      DOWN_ar           = (*MBC_data)->Output_number[15];
      DOWN_br           = (*MBC_data)->Output_number[16];
      DOWN_vr           = (*MBC_data)->Output_number[17];
      DOWN_vl           = (*MBC_data)->Output_number[18];
      DOWN_RFM          = (*MBC_data)->Output_number[19];

      // BASED ON THE MBC SELECTION BOOL VARIABLES, sort the incoming data
      //accordingly and identify the configuration

}

// Here is the test fn – simple line finding

void test_r()
{
      //Compute the first line's equation

      start1 = 20;                  // The PMS start and end index angles
```

```
	end1 = 28;

	sum_x=0;
	sum_y=0;
	sum_xsq=0;
	sum_ysq=0;
	sum_xy=0;

	no_data=abs((end1-start1))+1;



		for(k=start1;k<=end1;k++)
		{
			sum_x = sum_x + x[k];
			sum_y = sum_y + y[k];
			sum_xsq = sum_xsq + (x[k]*x[k]);
			sum_ysq = sum_ysq + (y[k]*y[k]);
			sum_xy = sum_xy + (x[k]*y[k]);
		}


		//compute the line parameters

		if((sum_xsq*no_data-sum_x*sum_x)>(sum_ysq*no_data-sum_y*sum_y))
		{
			al=((no_data*sum_xy)-(sum_x*sum_y))/((no_data*sum_xsq)-
			(sum_x*sum_x));
			bl=(sum_y - (al*sum_x))/(no_data);
		}
		else
		{
			al=(no_data*sum_ysq-sum_y*sum_y)/(no_data*sum_xy-
			sum_x*sum_y);
			bl=-(sum_x-(sum_y/al))*(al/no_data);
		}


}

void test_l()
{
	//Compute the first line's equation

	start = 20;		// The PMS start and end index angles
	end = 28;

	sum_x=0;
	sum_y=0;
	sum_xsq=0;
	sum_ysq=0;
	sum_xy=0;

	no_data=abs(end-start)+1;

	for(k=start;k<=(int)end;k++)
		{
```

```
                  sum_x = sum_x + x_l[k];
                  sum_y = sum_y + y_l[k];
                  sum_xsq = sum_xsq + (x_l[k]*x_l[k]);
                  sum_ysq = sum_ysq + (y_l[k]*y_l[k]);
                  sum_xy = sum_xy + (x_l[k]*y_l[k]);
            }


            //compute the line parameters

            if((sum_xsq*no_data-sum_x*sum_x)>(sum_ysq*no_data-sum_y*sum_y))
            {
                  al_l=((no_data*sum_xy)-(sum_x*sum_y))/((no_data*sum_xsq)-
                  (sum_x*sum_x));
                  bl_l=(sum_y - (al*sum_x))/(no_data);
            }
            else
            {
                  al_l=(no_data*sum_ysq-sum_y*sum_y)/(no_data*sum_xy-
                  sum_x*sum_y);
                  bl_l=-(sum_x-(sum_y/al))*(al/no_data);
            }
}


void grouping(int SIDE)
{
      int i,j;
      int counter;
      float toggle;
      float r[200];
      float th[200];
      float X[200];
      float Y[200];

      int min_angle=0;
      int max_angle;
      float temp_d;
      float d;
      float jump =5;                //max. threshold for succesive points.

      max_angle = 180-(int)theta_pig;

      if(max_angle>180)
            max_angle=180;

      if (SIDE == 1) //Right side
      {
            for (i=0;i<len;i++)
            {
                  r[i] = radius[i];
                  th[i] = angle[i];
                  X[i] = x[i];
                  Y[i] = y[i];
            }
      }
      else
```

```
	{
		for (i=0;i<len;i++)
		{
			r[i] = radius_l[i];
			th[i] = angle_l[i];
			X[i] = x_l[i];
			Y[i] = y_l[i];
		}
	}


	for(i=0;i<5;i++)
	{
		for(j=0;j<5;j++)
		{
			segment[i][j]=0;
		}
	}

	j=0;
	counter=0;
	toggle=0;

	for(i=min_angle+1;i<=max_angle;i++)
	{
		if(((fabs(r[i]-r[i-1])<jump))&&((r[i-
1]<max_radius)&&(toggle<0.5)))
		{
			segment[j][0]=(int)th[i-1];
			if(i>1)
			{
				if((r[i-2]-r[i-1])>jump)
				{
					segment[j][3]=1;
				}
			}

			toggle=1;
		}

		if((fabs(r[i]-r[i-1])>jump)&&(toggle>0.5)&&(r[i]<max_radius))
		{
			segment[j][1]=(int)th[i-1];

			if((r[i]-r[i-1])<0)
			{
				segment[j+1][3]=1;
			}

			else
			{
				segment[j][4]=1;
			}

			toggle=0;
			j=j+1;
		}
```

```
            else if ((toggle>0.5)&&(r[i]>max_radius-0.01))
            {
                  segment[j][1]=(int)th[i-1];

                  if((max_radius-r[i-1])>jump)
                  {
                        segment[j][4]=1;
                  }

                  toggle=0;
                  j++;
            }

            if (((i==max_angle)&&(toggle>0.5))&&(r[i]<max_radius-0.01))
            {
                  segment[j][1]=(int)th[i];
            }

      }

      //search for the group which contain the maximum length from the first
      //to last point
      d = 0;
      for(i=0;i<5;i++) // there is only one group of data extracted at a time
      {
            temp_d = sqrt((X[segment[i][1]]-
X[segment[i][0]])*(X[segment[i][1]]-X[segment[i][0]])+(Y[segment[i][1]]-
Y[segment[i][0]])*(Y[segment[i][1]]-Y[segment[i][0]]));
            if (temp_d > d)
            {
                  if((segment[i][3]+segment[i][4]>0)) //if the line group has
                                                     //at least one occluding
                                                     //corner in it
                  {
                        start=segment[i][0];         //assign the start point
                                                     //for line fit
                        end=segment[i][1];           //assign the end point
                                                     //for the line fit
                        temp_index=i;                //assign the row # of
                                                     //segment[i][j]
                        occ_corner = 1;
                  }
                  else
                  {
                        start=segment[i][0];         //assign the start point
                                                     //for line fit
                        end=segment[i][1];           //assign the end point
                                                     //for the line fit
                        temp_index=i;                //assign the row # of
                                                     //segment[i][j]
                        occ_corner = 0;
                  }
                  d = temp_d;
            }
      }
      max_length = d;
```

```
}

//REGRESSION FIT TO DATA

void line_fit()
{

      int t_start;
      int t_end;
      int k;

      //parameter for line splitting loop
      //int max_no_line = 11;       // maximum number of lines that allow in
                                    //the loop
      int angle_serie[max_no_line]; // this array contains start and end
                                    //angles of each splitted line.
      float max_deviation=2;
      int i;
      float a;
      float b;
      float temp_th;
      float temp_d=0;
      float x_k;
      float y_k;
      float d;
      int max_i;
      int terminate;
      int add_i;
      float progress_dist=3;
      float sick_offset=4;
//    float al2;
//    float bl2;
//    float theta;
// START ALGORITHM//

            //Recursive line splitting technique
            //1. initialize angle_serie
            angle_serie[0]=start;
            angle_serie[1]=end;
            for(i=2;i<max_no_line;i++)
            {
                  angle_serie[i]=0;
            }
            //2. begin to split line

            terminate = 0;

            while ( (terminate == 0))
            {
                  //count number of lines to be splitted in a loop
                  i = 0;
                  terminate = 0;

                  while (((angle_serie[i]!=0)||(i==0))&&(i<max_no_line-1))
                  {
                        i++;
                  }
```

```
                  if ((i==max_no_line-2)&&(angle_serie[i+1]!=0))
                  {
                        i = i+2;
                        terminate = 1;
                  }

                  max_i = i-2;

                  add_i = 0;
                  for(i=0;i<=max_i;i++)
                  {
                        //index no. cant be more than 180

                        temp_d = 0;
                        a = (y[angle_serie[i+add_i]]-
y[angle_serie[i+1+add_i]])/(x[angle_serie[i+add_i]]-
x[angle_serie[i+1+add_i]]);
                        b = -
(a*x[angle_serie[i+add_i]])+y[angle_serie[i+add_i]];

      for(k=angle_serie[i+add_i]+1;k<=angle_serie[i+1+add_i]-1;k++)
                        {
                              //find intersection pt.
                              if ((a<0.008)&&(a>-0.008))          //if the
line almost parallel to x-axis (-0.5<th<0.5 deg.)
                              {
                                    x_k=x[k];
                                    y_k=b;
                              }
                              else
                              {
                                    x_k=((x[k]/a)+y[k]-b)/(a+(1/a));
                                    y_k=a*x_k+b;
                              }
                              d = sqrt((x[k]-x_k)*(x[k]-x_k)+(y[k]-y_k)*(y[k]-
y_k));
                              if (temp_d < d)
                              {
                                    temp_th = k;
                                    temp_d = d;
                              }
                        }
                        //store temp_th into the angle_serie if temp_d exceeds
                        //specified maximum deviation
                        if ((temp_d > max_deviation)&&(max_no_line-2>i))
                        {
                              //1.shift all angles which bigger than temp_th
down
                              for(k=max_no_line-2;k>i+add_i;k--)
                              {
                                    angle_serie[k+1]=angle_serie[k];

                              }
                              //2.insert temp_th

                              angle_serie[i+1+add_i]=temp_th;
```

```
                        //3.set  terminate to 0;

                          terminate = 0;

                          //4.Shift add_i by 1
                          add_i = add_i+1;
                    }
                    else
                    {
                          terminate = 1;
                    }

              }

        }
        ////////////////////End splitting lines//////////////////

        ///////////////FIT TWO SIDEMOST GROUPS/////////////////
    start1 = angle_serie[0];
        end1 = angle_serie[1];
        i = 0;
        for(k=0;k<max_no_line;k++)
        {
              if(angle_serie[k]>0)
                    i = k;
        }
        start2 = angle_serie[i-1];
        end2 = angle_serie[i];

        //Compute the first line's equation

        sum_x=0;
        sum_y=0;
        sum_xsq=0;
        sum_ysq=0;
        sum_xy=0;

        no_data=abs((end1-start1))+1;



        for(k=start1;k<=end1;k++)
        {
              sum_x = sum_x + x[k];
              sum_y = sum_y + y[k];
              sum_xsq = sum_xsq + (x[k]*x[k]);
              sum_ysq = sum_ysq + (y[k]*y[k]);
              sum_xy = sum_xy + (x[k]*y[k]);
        }


        //compute the line parameters

        if((sum_xsq*no_data-sum_x*sum_x)>(sum_ysq*no_data-sum_y*sum_y))
        {
              al=((no_data*sum_xy)-(sum_x*sum_y))/((no_data*sum_xsq)-
(sum_x*sum_x));
```

```
                  bl=(sum_y - (al*sum_x))/(no_data);
            }
            else
            {
                  al=(no_data*sum_ysq-sum_y*sum_y)/(no_data*sum_xy-
sum_x*sum_y);
                  bl=-(sum_x-(sum_y/al))*(al/no_data);
            }

        //Compute the second line's equation

            sum_x=0;
            sum_y=0;
          sum_xsq=0;
            sum_ysq=0;
            sum_xy=0;

            no_data=abs((end2-start2))+1;

            for(k=start2;k<=end2;k++)
            {
                  sum_x = sum_x + x[k];
                  sum_y = sum_y + y[k];
                  sum_xsq = sum_xsq + (x[k]*x[k]);
                  sum_ysq = sum_ysq + (y[k]*y[k]);
                  sum_xy = sum_xy + (x[k]*y[k]);
            }
            //compute the line parameters

            if((sum_xsq*no_data-sum_x*sum_x)>(sum_ysq*no_data-sum_y*sum_y))
            {
                  al2=((no_data*sum_xy)-(sum_x*sum_y))/((no_data*sum_xsq)-
                  (sum_x*sum_x));
                  bl2=(sum_y - (al2*sum_x))/(no_data);
            }
            else
            {
                  al2=(no_data*sum_ysq-sum_y*sum_y)/(no_data*sum_xy-
                  sum_x*sum_y);
                  bl2=-(sum_x-(sum_y/al2))*(al2/no_data);
            }
            ////////////////END FIT TWO LINeS/////////////
            angle_btw_line = fabs(atan(al)-atan(al2))*180/pi;
            if(angle_btw_line>30)
            {//YES, it's a corner.
                  if
((occ_corner==0)||((segment[temp_index][3]==1)&&(segment[temp_index][4]==1)))

// it's a corner seen from zone#2
                  {

                        if (fabs(al2)>fabs(al))
                        {
                              case_no = 1.1;
                              //1. Locate the current position
                              //1.1 locate origin of global coordinate
system(XY)
```

```
                              //1.2 choose the wall(line1 or line2) used as an
angle reference
                              //1.3 calculate current position(x,y) of MBC
                              x_ref = (bl2-bl)/(al-al2);
                              /*if (fabs(al2)<fabs(al))
                              {
                                    y_ref = (al2*x_ref)+bl2;
                                    th_cur = -atan(al2);
                                    theta = (3*pi/2)-th_cur;
                                    x_cur = -x_ref*cos(theta)-
y_ref*sin(theta)-sick_offset*sin(theta);
                                    y_cur = -
y_ref*cos(theta)+x_ref*sin(theta)-sick_offset*cos(theta);
                              }
                              else
                              {*/
                                    y_ref = (al*x_ref)+bl;
                                    th_cur = -atan(al);
                                    theta = pi-th_cur;
                                    x_cur = -x_ref*cos(theta)-
y_ref*sin(theta)-sick_offset*sin(theta);
                                    y_cur = -
y_ref*cos(theta)+x_ref*sin(theta)-sick_offset*cos(theta);
                              //}
                              th_cur = th_cur*180/pi;

                              //2.Locate the position of center of arc path
                              //2.1 initialize the  values of xd & yd;
                              xd=15*1.2;
                              yd=20*1.2;

                              xc = xd+x_cur;  // pos of MBC in frame Center
                              yc = yd+y_cur;

                              R_mbc = sqrt((xc*xc)+(yc*yc));


                              th_cen_mbc = atan2(yc,xc);
                              th_cen_mbc = check_theta(th_cen_mbc);
                              //y_cur = (R_mbc*sin(th_cen_mbc))-yd;

                              //compute r_dist for the next target point

                              th_tar=(th_cen_mbc) - 5*pi/180;

                              x_tar = 29*1.20*cos(th_tar)-xd;

                              y_tar = (29*1.20*sin(th_tar))-yd;

                              th_tar = (th_tar*180/pi)-90;
      //convert to degrees;
                        }
                        else
                        {
                              case_no = 1.20;
                              x_ref = (bl2-bl)/(al-al2);
                              /*
```

```
                              if (fabs(al2)<fabs(al))//||(abs((int)a_small-
start)>abs(end-(int)a_small)))
                              {
                                    y_ref = (al2*x_ref)+bl2;
                              }
                              else
                              {
                                    y_ref = (al*x_ref)+bl;
                              }*/

                              //if (fabs(al2)<fabs(al))
                              //{
                                    y_ref = (al2*x_ref)+bl2;
                                    th_cur = -atan(al2);
                                    theta = (3*pi/2)-th_cur;
                                    x_cur = -x_ref*cos(theta)-
y_ref*sin(theta)-sick_offset*sin(theta);
                                    y_cur = -
y_ref*cos(theta)+x_ref*sin(theta)-sick_offset*cos(theta);
                              /*}
                              else
                              {
                                    y_ref = (al*x_ref)+bl;
                                    th_cur = -atan(al);
                                    theta = pi-th_cur;
                                    x_cur = -x_ref*cos(theta)-
y_ref*sin(theta)-sick_offset*sin(theta);
                                    y_cur = -
y_ref*cos(theta)+x_ref*sin(theta)-sick_offset*cos(theta);

                              //}*/

                              th_cur = ((-pi/2)+th_cur)*180/pi;
                              /*if ((al2<0.008)&&(al2>-0.008))          //if
the line almost parallel to x-axis (-0.5<th<0.5 deg.)
                              {
                                    x_pos=0;
                                    y_pos=bl2;
                              }
                              else
                              {
                                    x_pos=-bl2/(al2+(1/al2));
                                    //y_pos=bl2;
                                    //y_pos = -(x_pos/al2);
                                    y_pos = (x_pos*al2)+bl2;
                              }*/
//find the distance between the two points (x_ref, y_ref) & (x_pos, y_pos)
//x_cur=-2000;    // return 1000 to alert other function
                  //y_cur=(float)sqrt((x_pos*x_pos)+(y_pos*y_pos));
                              //th_cur = -atan(al)*180/pi;

                              /*
                              //locate current position
                              y_cur=-(float)sqrt(((x_ref-x_pos)*(x_ref-
x_pos))+((y_ref-y_pos)*(y_ref-y_pos)));   // return 1000 to alert other
function
                              th_cur = atan(al2);
```

```
      x_cur=(float)sqrt((x_pos*x_pos)+(y_pos*y_pos))+(sick_offset*cos(th_cur))
;
                              th_cur = -atan(al2)*180/pi-(90);
                              //locate target point using straight line eq.
                              //x_tar=x_cur+progress_dist;
                              //y_tar=(float)15.80; //in.
                              //th_tar=0;
                              */
/*
                              //initialize the  values of xd & ud;

                              xd=20*1.2;
                              yd=15*1.2;

                              xc=x_ref +
sqrt((xd*xd)+(yd*yd))*cos(atan(al2)+atan(yd/xd)+pi/2);
                              yc=y_ref +
sqrt((xd*xd)+(yd*yd))*sin(atan(al2)+atan(yd/xd)+pi/2);

                              //compute target angle and target distance

                              R_mbc =
sqrt((xc*xc)+((yc+sick_offset)*(yc+sick_offset)));
                              th_cen_mbc = (atan((yc+sick_offset)/xc)-
(atan(al2)+pi/2));

                              //y_cur = (R_mbc*sin(th_cen_mbc))-yd;
                              //compute r_dist for the next target point

                              th_tar=(th_cen_mbc) - 5*pi/180;
                              x_tar = 29*1.2*cos(th_tar)-xd;
                              y_tar = (29*1.2*sin(th_tar))-yd;
                              th_tar = (th_tar*180/pi)-90;
      //convert to degrees;
                              ////////////////////////////
*/
                              xd=15*1.2;
                              yd=20*1.2;

                              xc = xd+x_cur;  // position of MBC in frame
Center
                              yc = yd+y_cur;

                              R_mbc = sqrt((xc*xc)+(yc*yc));


                              th_cen_mbc = atan2(yc,xc);
                              th_cen_mbc = check_theta(th_cen_mbc);
                              //y_cur = (R_mbc*sin(th_cen_mbc))-yd;


                              //compute r_dist for the next target point

                              th_tar=(th_cen_mbc) - 5*pi/180;

                              x_tar = 29*1.20*cos(th_tar)-xd;
```

```
                              y_tar = (29*1.20*sin(th_tar))-yd;

                              th_tar = (th_tar*180/pi)-90;
      //convert to degrees;
                        }

                  }
                  else  // it's a corner seen from zone#1
                  {
//locate intersection pt. between line1(y=al*x+bl) and a perpendicular line
//which pass the occ. corner pt.(t_end)
                        if(segment[temp_index][4]==1) //if SICK can see the
                                                      //corner directly
                        {
                              case_no = 2;
                              t_end=segment[temp_index][1];

                              x_ref=x[t_end];         //corresponds to ths END
                                                      //angle
                              y_ref=y[t_end];

                              if ((al<0.008)&&(al>-0.008))

            //if the line almost parallel to x-axis (-0.5<th<0.5 deg.)
                              {
                                    x_pos=x_ref;
                                    y_pos=bl;
                              }
                              else
                              {
                                    x_pos=((x_ref/al)+y_ref-bl)/(al+(1/al));
                                    y_pos=al*x_pos+bl;
                              }
                              x_ref = x_pos;          // assign temp. values
                                                      //back to original
                                                      //parameter
                              y_ref = y_pos;

                              /***/
                              //1. Locate the current position
                              //1.1 locate origin of global coordinate
                              //system(XY)
                              //1.2 choose the wall(line1 or line2) used as an
                              //angle reference
                              //1.3 calculate current position(x,y) of MBC

                              /*if (fabs(al2)<fabs(al))
                              {
                                    //y_ref = (al2*x_ref)+bl2;
                                    th_cur = -atan(al2);
                                    theta = (3*pi/2)-th_cur;
                                    x_cur = -x_ref*cos(theta)-
y_ref*sin(theta)-sick_offset*sin(theta);
                                    y_cur = -
y_ref*cos(theta)+x_ref*sin(theta)-sick_offset*cos(theta);
                              }
```

```
                              else
                              {*/
                                    //y_ref = (al*x_ref)+bl;
                                    th_cur = -atan(al);
                                    theta = pi-th_cur;
                                    x_cur = -x_ref*cos(theta)-
y_ref*sin(theta)-sick_offset*sin(theta);
                                    y_cur = -
y_ref*cos(theta)+x_ref*sin(theta)-sick_offset*cos(theta);
                              //}
                              th_cur = th_cur*180/pi;


//locate intersection pt. between line1(y=al*x+bl) and a perpendicular line
//which pass the origin(location of SICK)
                              if ((al<0.008)&&(al>-0.008))
//if the line almost parallel to x-axis (-0.5<th<0.5 deg.)
                              {
                                    x_pos=0;
                                    y_pos=bl;
                              }
                              else
                              {
                                    x_pos=-bl/(al+(1/al));
                                    //y_pos=bl;
                                    //y_pos = -(x_pos/al);
                                    y_pos = (x_pos*al)+bl;
                              }
//find the distance between the two points (x_ref, y_ref) & (x_pos, y_pos)
                              //x_cur=-(float)sqrt(((x_ref-x_pos)*(x_ref-
x_pos))+((y_ref-y_pos)*(y_ref-y_pos)));

      //y_cur=(float)sqrt((x_pos*x_pos)+(y_pos*y_pos));
                              //th_cur = -atan(al)*180/pi;

                              //locate current position
                              x_cur=-(float)sqrt(((x_ref-x_pos)*(x_ref-
x_pos))+((y_ref-y_pos)*(y_ref-y_pos)));
                              th_cur = atan(al);

      y_cur=(float)sqrt((x_pos*x_pos)+(y_pos*y_pos))+(sick_offset*cos(th_cur))
;
                              th_cur = -atan(al)*180/pi;

                              //locate target point using straight line eq.
                              //x_tar=x_cur+progress_dist;
                              //y_tar=-0.3*x_tar+8.6; //in.
                              //th_tar=-atan(0.3)*180/pi;

                              //locate target position using an arc//////
                              //initialize the  values of xd & ud;
                              ************************************/
                              /*
                              xd=20*1.2;
                              yd=15*1.2;
```

```
                              xc=x_ref +
sqrt((xd*xd)+(yd*yd))*cos(atan(al)+atan(yd/xd));
                              yc=y_ref +
sqrt((xd*xd)+(yd*yd))*sin(atan(al)+atan(yd/xd));

                              //compute target angle and target distance

                              R_mbc =
sqrt((xc*xc)+((yc+sick_offset)*(yc+sick_offset)));
                              th_cen_mbc = (atan((yc+sick_offset)/xc)-
atan(al));

                              //y_cur = (R_mbc*sin(th_cen_mbc))-yd;
                              //compute r_dist for the next target point

                              th_tar=(th_cen_mbc) - 5*pi/180;
                              x_tar = 29*1.2*cos(th_tar)-xd;
                              y_tar = (29*1.2*sin(th_tar))-yd;
                              th_tar = (th_tar*180/pi)-90;
//convert to degrees;
                              ////////////////////////////
*/
                              xd=15*1.2;
                              yd=20*1.2;

                              xc = xd+x_cur;  // position of MBC in frame
Center
                              yc = yd+y_cur;

                              R_mbc = sqrt((xc*xc)+(yc*yc));


                              th_cen_mbc = atan2(yc,xc);
                              th_cen_mbc = check_theta(th_cen_mbc);
                              //y_cur = (R_mbc*sin(th_cen_mbc))-yd;


                              //compute r_dist for the next target point

                              th_tar=(th_cen_mbc) - 5*pi/180;

                              x_tar = 29*1.20*cos(th_tar)-xd;

                              y_tar = (29*1.20*sin(th_tar))-yd;

                              th_tar = (th_tar*180/pi)-90;
//convert to degrees;
                        }
                        else if(segment[temp_index][3] == 1)
                        {
                              //locate intersection pt. between
line1(y=al*x+bl) and a perpendicular line which pass the origin(location of
SICK)
                              case_no = 3;
                              t_start=segment[temp_index][0];
```

```
                              x_ref=x[t_start];             //corresponds to
ths START angle
                              y_ref=y[t_start];

                              if ((al2<0.008)&&(al2>-0.008))            //if
the line almost parallel to x-axis (-0.5<th<0.5 deg.)
                              {
                                    x_pos=x_ref;
                                    y_pos=bl2;
                              }
                              else
                              {
                                    x_pos=((x_ref/al2)+y_ref-
bl2)/(al2+(1/al2));
                                    y_pos=al2*x_pos+bl2;
                              }
                              x_ref = x_pos;// assign temp. values back to
original parameter
                              y_ref = y_pos;


                              //y_ref = (al2*x_ref)+bl2;
                              th_cur = -atan(al2);
                              theta = (3*pi/2)-th_cur;
                              x_cur = -x_ref*cos(theta)-y_ref*sin(theta)-
sick_offset*sin(theta);
                              y_cur = -y_ref*cos(theta)+x_ref*sin(theta)-
sick_offset*cos(theta);
                              th_cur = ((-pi/2)+th_cur)*180/pi;
                              /*if ((al2<0.008)&&(al2>-0.008))          //if
the line almost parallel to x-axis (-0.5<th<0.5 deg.)
                              {
                                    x_pos=0;
                                    y_pos=bl2;
                              }
                              else
                              {
                                    x_pos=-bl2/(al2+(1/al2));
                                    //y_pos=bl2;
                                    //y_pos = -(x_pos/al2);
                                    y_pos = (x_pos*al2)+bl2;
                              }*/
                              //find the distance between the two points
(x_ref, y_ref) & (x_pos, y_pos)
                              //x_cur=-2000;    // return 1000 to alert other
function

      //y_cur=(float)sqrt((x_pos*x_pos)+(y_pos*y_pos));
                              //th_cur = -atan(al)*180/pi;

                              /*
                              //locate current position
                              y_cur=-(float)sqrt(((x_ref-x_pos)*(x_ref-
x_pos))+((y_ref-y_pos)*(y_ref-y_pos)));   // return 1000 to alert other
function
                              th_cur = atan(al2);
```

```
      x_cur=(float)sqrt((x_pos*x_pos)+(y_pos*y_pos))+(sick_offset*cos(th_cur))
;
                              th_cur = -atan(al2)*180/pi-(90);
                              //locate target point using straight line eq.
                              //x_tar=x_cur+progress_dist;
                              //y_tar=(float)15.80; //in.
                              //th_tar=0;
                              */

                              //initialize the  values of xd & ud;

                              xd=20*1.2;
                              yd=15*1.2;

                              xc=x_ref +
sqrt((xd*xd)+(yd*yd))*cos(atan(al2)+atan(yd/xd)+pi/2);
                              yc=y_ref +
sqrt((xd*xd)+(yd*yd))*sin(atan(al2)+atan(yd/xd)+pi/2);

                              //compute target angle and target distance

                              R_mbc =
sqrt((xc*xc)+((yc+sick_offset)*(yc+sick_offset)));
                              th_cen_mbc = (atan2((yc+sick_offset),xc)-
(atan(al2)+pi/2));

                              //y_cur = (R_mbc*sin(th_cen_mbc))-yd;
                              //compute r_dist for the next target point

                              th_tar=(th_cen_mbc) - 5*pi/180;
                              x_tar = 29*1.2*cos(th_tar)-xd;
                              y_tar = (29*1.2*sin(th_tar))-yd;
                              th_tar = (th_tar*180/pi)-90;
      //convert to degrees;
                              ////////////////////////////
                        }
                  }

            }
            else
            {//NO, it's a straight line, locate the corner directly by using
an occluding angle
                  if
((segment[temp_index][3]==1)&&(segment[temp_index][4]==1)&&(max_length<20))
// in case of chamfer in zoon#2
                  {
                        case_no = 40;
                        x_ref =
x[(int)(segment[temp_index][0]+segment[temp_index][1])/2];  // use point
locate at the middle of the group
                        y_ref =
y[(int)(segment[temp_index][0]+segment[temp_index][1])/2];

                        if ((al<0.008)&&(al>-0.008))        //if the line
almost parallel to x-axis (-0.5<th<0.5 deg.)
                        {
```

```
                              x_pos=0;
                              y_pos=bl;
                        }
                        else
                        {
                              x_pos=-bl/(al+(1/al));
                              //y_pos=bl;
                              //y_pos = -(x_pos/al);
                              y_pos = (x_pos*al)+bl;
                        }
                        //find the distance between the two points (x_ref,
y_ref) & (x_pos, y_pos)
                        x_cur = (float)sqrt(((x_ref-x_pos)*(x_ref-
x_pos))+((y_ref-y_pos)*(y_ref-y_pos)));   //LOOP TO LOCATE THE CENTER OF THE
ARC....
                        th_cur = -atan(al);

      y_cur=(float)sqrt((x_pos*x_pos)+(y_pos*y_pos))+(sick_offset*cos(th_cur))
;
                        th_cur = -atan(al)*180/pi;
                        //initialize the  values of xd & ud;

                        xd=20*1.2;
                        yd=15*1.2;

                        xc=x_ref +
sqrt((xd*xd)+(yd*yd))*cos(atan(al)+atan(yd/xd));
                        yc=y_ref +
sqrt((xd*xd)+(yd*yd))*sin(atan(al)+atan(yd/xd));

                        //compute target angle and target distance

                        R_mbc =
sqrt((xc*xc)+((yc+sick_offset)*(yc+sick_offset)));

                        th_cen_mbc = (atan((yc+sick_offset)/xc)-atan(al));

                        //y_cur = (R_mbc*sin(th_cen_mbc))-yd;


                        //compute r_dist for the next target point

                        th_tar=(th_cen_mbc) - 5*pi/180;

                        x_tar = 29*1.2*cos(th_tar)-xd;

                        y_tar = (29*1.2*sin(th_tar))-yd;

                        th_tar = (th_tar*180/pi)-90;
      //convert to degrees;


                  }

                  else if(segment[temp_index][4]==1)//if SICK can see the
corner directly
                  {
```

```
                        case_no = 50;
                        t_end=segment[temp_index][1];
                        //t_end = end2;
                        x_ref=x[t_end];               //corresponds to ths END
angle
                        y_ref=y[t_end];

                        if ((al<0.008)&&(al>-0.008))        //if the line
almost parallel to x-axis (-0.5<th<0.5 deg.)
                        {
                              x_pos=x_ref;
                              y_pos=bl;
                        }
                        else
                        {
                              x_pos=((x_ref/al)+y_ref-bl)/(al+(1/al));
                              y_pos=al*x_pos+bl;
                        }
                        x_ref = x_pos;// assign temp. values back to original
parameter
                        y_ref = y_pos;

                        /***/
                              //1. Locate the current position
                              //1.1 locate origin of global coordinate
system(XY)
                              //1.2 choose the wall(line1 or line2) used as an
angle reference
                              //1.3 calculate current position(x,y) of MBC

                              /*if (fabs(al2)<fabs(al))
                              {
                                    //y_ref = (al2*x_ref)+bl2;
                                    th_cur = -atan(al2);
                                    theta = (3*pi/2)-th_cur;
                                    x_cur = -x_ref*(cos(theta)+sin(theta))-
sick_offset*sin(theta);
                                    y_cur = -y_ref*(cos(theta)-sin(theta))-
sick_offset*cos(theta);
                              }
                              else
                              {*/
                                    //y_ref = (al*x_ref)+bl;
                                    th_cur = -atan(al);
                                    theta = pi-th_cur;
                                    x_cur = -x_ref*cos(theta)-
y_ref*sin(theta)-sick_offset*sin(theta);
                                    y_cur = -
y_ref*cos(theta)+x_ref*sin(theta)-sick_offset*cos(theta);
                              //}
                              th_cur = th_cur*180/pi;

                              /***/

                        /*************
                        if ((al<0.008)&&(al>-0.008))        //if the line
almost parallel to x-axis (-0.5<th<0.5 deg.)
```

```
                        {
                              x_pos=0;
                              y_pos=bl;
                        }
                        else
                        {
                              x_pos=-bl/(al+(1/al));
                              //y_pos=bl;
                              //y_pos = -(x_pos/al);
                              y_pos = (x_pos*al)+bl;
                        }

                        //find the distance between the two points (x_ref,
y_ref) & (x_pos, y_pos)
                        //x_cur=-(float)sqrt(((x_ref-x_pos)*(x_ref-
x_pos))+((y_ref-y_pos)*(y_ref-y_pos)));
                        //y_cur=(float)sqrt((x_pos*x_pos)+(y_pos*y_pos));
                        //th_cur = -atan(al)*180/pi;

                        //locate current position
                              x_cur=-(float)sqrt(((x_ref-x_pos)*(x_ref-
x_pos))+((y_ref-y_pos)*(y_ref-y_pos)));
                              th_cur = -atan(al);

      y_cur=(float)sqrt((x_pos*x_pos)+(y_pos*y_pos))+(sick_offset*cos(th_cur))
;
                              th_cur = -atan(al)*180/pi;
                              ********************/
                              //locate target point
                              x_tar=x_cur+progress_dist;
                              y_tar=19;//-0.3*x_tar+8.6; //in.
      // was 19
                              th_tar=0;//-atan(0.3)*180/pi;

                  }

                  else if(segment[temp_index][3]==1)//if SICK can not see the
corner
                  {
                        case_no = 6;

                        t_start=segment[temp_index][0];

                              x_ref=x[t_start];             //corresponds to
ths START angle
                              y_ref=y[t_start];

                              if ((al2<0.008)&&(al2>-0.008))            //if
the line almost parallel to x-axis (-0.5<th<0.5 deg.)
                              {
                                    x_pos=x_ref;
                                    y_pos=bl2;
                              }
                              else
                              {
                                    x_pos=((x_ref/al2)+y_ref-
bl2)/(al2+(1/al2));
```

```
                                    y_pos=al2*x_pos+bl2;
                              }
                              x_ref = x_pos;// assign temp. values back to
original parameter
                              y_ref = y_pos;


                              //y_ref = (al2*x_ref)+bl2;
                              th_cur = -atan(al2);
                              theta = (3*pi/2)-th_cur;
                              x_cur = -x_ref*cos(theta)-y_ref*sin(theta)-
sick_offset*sin(theta);
                              y_cur = -y_ref*cos(theta)+x_ref*sin(theta)-
sick_offset*cos(theta);
                              th_cur = ((-pi/2)+th_cur)*180/pi;
                              /*
                        //MBC should do the followings
                        //1.eliminate the pig angle,so that it can see the
corner
                        //2.parallel itself to the wall or a bit positive
angle
                        if ((al<0.008)&&(al>-0.008))        //if the line
almost parallel to x-axis (-0.5<th<0.5 deg.)
                        {
                              x_pos=0;
                              y_pos=bl;
                        }
                        else
                        {
                              x_pos =-bl/(al+(1/al));
                              y_pos = (x_pos*al)+bl;
                        }
                        x_cur=-2000;      // return 1000 to alert other
function
                        th_cur = atan(al);

      y_cur=(float)sqrt((x_pos*x_pos)+(y_pos*y_pos))+(sick_offset*cos(th_cur))
;
                              th_cur = -atan(al)*180/pi;
                  */
                              //locate target point
                              x_tar=19.0;
                              y_tar=y_cur-progress_dist; //in.
                              th_tar=-90;

                  }

                  else //There is no occ. corner, this group of data must be
taken in zone#1 of long mine pillar.
                         //Therefore, we fit the line from start to end angle
                  {
                        case_no = 7;
                        //Compute the line's equation
                        sum_x=0;
                        sum_y=0;
                        sum_xsq=0;
                        sum_ysq=0;
```

```
                        sum_xy=0;

                        no_data=abs(end-start)+1;

                        for(k=start;k<=(int)end;k++)
                        {
                              sum_x = sum_x + x[k];
                              sum_y = sum_y + y[k];
                              sum_xsq = sum_xsq + (x[k]*x[k]);
                              sum_ysq = sum_ysq + (y[k]*y[k]);
                        sum_xy = sum_xy + (x[k]*y[k]);
                        }


                        //compute the line parameters

                        if((sum_xsq*no_data-sum_x*sum_x)>(sum_ysq*no_data-
sum_y*sum_y))
                        {
                              al=((no_data*sum_xy)-
(sum_x*sum_y))/((no_data*sum_xsq)-(sum_x*sum_x));
                              bl=(sum_y - (al*sum_x))/(no_data);
                        }
                        else
                        {
                              al=(no_data*sum_ysq-
sum_y*sum_y)/(no_data*sum_xy-sum_x*sum_y);
                              bl=-(sum_x-(sum_y/al))*(al/no_data);
                        }

                        if ((al<0.008)&&(al>-0.008))        //if the line
almost parallel to x-axis (-0.5<th<0.5 deg.)
                        {
                              x_pos=0;
                              y_pos=bl;
                        }
                        else
                        {
                              x_pos=-bl/(al+(1/al));
                              //y_pos=bl;
                              //y_pos = -(x_pos/al);
                              y_pos = (x_pos*al)+bl;

                        }

                        x_cur=-1000;      // return 1000 to alert other
function
                        th_cur = atan(al);

      y_cur=(float)sqrt((x_pos*x_pos)+(y_pos*y_pos))+(sick_offset*cos(th_cur))
;
                        th_cur = -atan(al)*180/pi;

                        //locate target point
                              x_tar=x_cur+progress_dist;
                              y_tar=(float)19; //in.
                              th_tar=0;
```

```
                  }
            }
}

//function to perform line fitting for the left SICK - used just as a
//ballast...

void line_fit_left()
{

      int t_start;
      int t_end;
      int k;

      //parameter for line splitting loop
      int angle_serie[max_no_line]; // this array contains start and end
                                    //angles of each splitted line.
      float max_deviation=2;
      int i;
      float a;
      float b;
      float temp_th;
      float temp_d=0;
      float x_k;
      float y_k;
      float d;
      int max_i;
      int terminate;
      int add_i;
      float progress_dist=3;
      float sick_offset=4.5;
//    float al2;
//    float bl2;
      float theta;
// START ALGORITHM//

            //Recursive line splitting technique
            //1. initialize angle_serie
            angle_serie[0]=start;
            angle_serie[1]=end;
            for(i=2;i<max_no_line;i++)
            {
                  angle_serie[i]=0;
            }
            //2. begin to split line
            terminate = 0;
            while ( (terminate == 0))
            {
                  //count number of lines to be splitted in a loop
                  i = 0;
                  terminate = 0;

                  while (((angle_serie[i]!=0)||(i==0))&&(i<max_no_line-1))
                  {
                        i++;
                  }
                  if ((i==max_no_line-2)&&(angle_serie[i+1]!=0))
```

```
                  {
                        i = i+2;
                        terminate = 1;
                  }

                  max_i = i-2;

                  add_i = 0;
                  for(i=0;i<=max_i;i++)
                  {
                        //index no. cant be more than 180

                        temp_d = 0;
                        a = (y_l[angle_serie[i+add_i]]-
y_l[angle_serie[i+1+add_i]])/(x_l[angle_serie[i+add_i]]-
x_l[angle_serie[i+1+add_i]]);
                        b = -
(a*x_l[angle_serie[i+add_i]])+y_l[angle_serie[i+add_i]];

      for(k=angle_serie[i+add_i]+1;k<=angle_serie[i+1+add_i]-1;k++)
                        {
                              //find intersection pt.
                              if ((a<0.008)&&(a>-0.008))          //if the
line almost parallel to x-axis (-0.5<th<0.5 deg.)
                              {
                                    x_k=x[k];
                                    y_k=b;
                              }
                              else
                              {
                                    x_k=((x[k]/a)+y[k]-b)/(a+(1/a));
                                    y_k=a*x_k+b;
                              }
                              d = sqrt((x[k]-x_k)*(x[k]-x_k)+(y[k]-y_k)*(y[k]-
y_k));
                              if (temp_d < d)
                              {
                                    temp_th = k;
                                    temp_d = d;
                              }
                        }

                        //store temp_th into the angle_serie if temp_d exceeds
specified maximum deviation
                        if ((temp_d > max_deviation)&&(max_no_line-2>i))
                        {
                              //1.shift all angles which bigger than temp_th
down
                              for(k=max_no_line-2;k>i+add_i;k--)
                              {
                                    angle_serie[k+1]=angle_serie[k];

                              }
                              //2.insert temp_th

                              angle_serie[i+1+add_i]=temp_th;
```

```
                        //3.set  terminate to 0;

                          terminate = 0;

                          //4.Shift add_i by 1
                          add_i = add_i+1;
                    }
                    else
                    {
                          terminate = 1;
                    }

              }

        }
        ////////////////////End splitting lines//////////////////

        ///////////////FIT TWO SIDEMOST GROUPS/////////////////

    start1 = angle_serie[0];
        end1 = angle_serie[1];
        i = 0;
        for(k=0;k<max_no_line;k++)
        {
              if(angle_serie[k]>0)
                    i = k;
        }
        start2 = angle_serie[i-1];
        end2 = angle_serie[i];

        //Compute the first line's equation

        sum_x=0;
        sum_y=0;
        sum_xsq=0;
        sum_ysq=0;
        sum_xy=0;

        no_data=abs((end1-start1))+1;



        for(k=start1;k<=end1;k++)
        {
              sum_x = sum_x + x_l[k];
              sum_y = sum_y + y_l[k];
              sum_xsq = sum_xsq + (x_l[k]*x_l[k]);
              sum_ysq = sum_ysq + (y_l[k]*y_l[k]);
              sum_xy = sum_xy + (x_l[k]*y_l[k]);
        }


        //compute the line parameters

        if((sum_xsq*no_data-sum_x*sum_x)>(sum_ysq*no_data-sum_y*sum_y))
        {
```

```
                  al=((no_data*sum_xy)-(sum_x*sum_y))/((no_data*sum_xsq)-
(sum_x*sum_x));
                  bl=(sum_y - (al*sum_x))/(no_data);
            }
            else
            {
                  al=(no_data*sum_ysq-sum_y*sum_y)/(no_data*sum_xy-
sum_x*sum_y);
                  bl=-(sum_x-(sum_y/al))*(al/no_data);
            }
            al_l = al;
            bl_l = bl;

        //Compute the second line's equation

            sum_x=0;
            sum_y=0;
          sum_xsq=0;
            sum_ysq=0;
            sum_xy=0;

            no_data=abs((end2-start2))+1;

            for(k=start2;k<=end2;k++)
            {
                  sum_x = sum_x + x_l[k];
                  sum_y = sum_y + y_l[k];
                  sum_xsq = sum_xsq + (x_l[k]*x_l[k]);
                  sum_ysq = sum_ysq + (y_l[k]*y_l[k]);
                  sum_xy = sum_xy + (x_l[k]*y_l[k]);
            }
            //compute the line parameters

            if((sum_xsq*no_data-sum_x*sum_x)>(sum_ysq*no_data-sum_y*sum_y))
            {
                  al2=((no_data*sum_xy)-(sum_x*sum_y))/((no_data*sum_xsq)-
(sum_x*sum_x));
                  bl2=(sum_y - (al2*sum_x))/(no_data);
            }
            else
            {
                  al2=(no_data*sum_ysq-sum_y*sum_y)/(no_data*sum_xy-
sum_x*sum_y);
                  bl2=-(sum_x-(sum_y/al2))*(al2/no_data);
            }
            ///////////////END FIT TWO LINeS/////////////
            angle_btw_line = fabs(atan(al)-atan(al2))*180/pi;
            if(angle_btw_line>15)                     // was 30
            {//YES, it's a corner.
                  if
((occ_corner==0)||((segment[temp_index][3]==1)&&(segment[temp_index][4]==1)))
      // it's a corner seen from zone#2
                  {

                        if (fabs(al2)>fabs(al))
                        {
                              case_no = 1.1;
```

```
                              //1. Locate the current position
                              //1.1 locate origin of global coordinate
system(XY)
                              //1.2 choose the wall(line1 or line2) used as an
angle reference
                              //1.3 calculate current position(x,y) of MBC
                              x_ref = (bl2-bl)/(al-al2);
                              if (fabs(al2)<fabs(al))
                              {
                                    y_ref = (al2*x_ref)+bl2;
                                    th_cur = -atan(al2);
                                    theta = (3*pi/2)-th_cur;
                                    x_cur = -
x_ref*(cos(theta)+sin(theta))+sick_offset*cos(th_cur);
                                    y_cur = -y_ref*(cos(theta)-
sin(theta))+sick_offset*sin(th_cur);
                              }
                              else
                              {
                                    y_ref = (al*x_ref)+bl;
                                    th_cur = -atan(al);
                                    theta = pi-th_cur;
                                    x_cur = -
x_ref*(cos(theta)+sin(theta))+sick_offset*cos(th_cur+(pi/2));
                                    y_cur = -y_ref*(cos(theta)-
sin(theta))+sick_offset*sin(th_cur+(pi/2));
                              }


                              //2.Locate the position of center of the arc
path
                              //2.1 initialize the  values of xd & yd;
                              xd=20*1.2;
                              yd=15*1.2;

                              xc = xd+x_cur;  // position of MBC in frame
Center
                              yc = yd+y_cur;

                              R_mbc = sqrt((xc*xc)+(yc*yc));

                              th_cur = th_cur*180/pi;

                              th_cen_mbc = atan(yc/xc);
                              //y_cur = (R_mbc*sin(th_cen_mbc))-yd;


                              //compute r_dist for the next target point

                              th_tar=(th_cen_mbc) - 5*pi/180;

                              x_tar = 29*1.20*cos(th_tar)-xd;

                              y_tar = (29*1.20*sin(th_tar))-yd;

                              th_tar = (th_tar*180/pi)-90;
      //convert to degrees;
```

```
                        }
                        else
                        {
                              case_no = 1.2;
                              x_ref = (bl2-bl)/(al-al2);
                              if (fabs(al2)<fabs(al))//||(abs((int)a_small-
start)>abs(end-(int)a_small)))
                              {
                                    y_ref = (al2*x_ref)+bl2;
                              }
                              else
                              {
                                    y_ref = (al*x_ref)+bl;
                              }

                              if ((al2<0.008)&&(al2>-0.008))            //if
the line almost parallel to x-axis (-0.5<th<0.5 deg.)
                              {
                                    x_pos=0;
                                    y_pos=bl2;
                              }
                              else
                              {
                                    x_pos=-bl2/(al2+(1/al2));
                                    //y_pos=bl2;
                                    //y_pos = -(x_pos/al2);
                                    y_pos = (x_pos*al2)+bl2;
                              }
                              //find the distance between the two points
(x_ref, y_ref) & (x_pos, y_pos)
                              //x_cur=-2000;    // return 1000 to alert other
function

      //y_cur=(float)sqrt((x_pos*x_pos)+(y_pos*y_pos));
                              //th_cur = -atan(al)*180/pi;


                              //locate current position
                              y_cur=-(float)sqrt(((x_ref-x_pos)*(x_ref-
x_pos))+((y_ref-y_pos)*(y_ref-y_pos)));   // return 1000 to alert other
function
                              th_cur = atan(al2);

      x_cur=(float)sqrt((x_pos*x_pos)+(y_pos*y_pos))+(sick_offset*cos(th_cur))
;
                              th_cur = -atan(al2)*180/pi-(90);
                              //locate target point using straight line eq.
                              //x_tar=x_cur+progress_dist;
                              //y_tar=(float)15.80; //in.
                              //th_tar=0;


                              //initialize the  values of xd & ud;

                              xd=20*1.2;
                              yd=15*1.2;
```

```
                              xc=x_ref +
sqrt((xd*xd)+(yd*yd))*cos(atan(al2)+atan(yd/xd)+pi/2);
                              yc=y_ref +
sqrt((xd*xd)+(yd*yd))*sin(atan(al2)+atan(yd/xd)+pi/2);

                              //compute target angle and target distance

                              R_mbc =
sqrt((xc*xc)+((yc+sick_offset)*(yc+sick_offset)));
                              th_cen_mbc = (atan((yc+sick_offset)/xc)-
(atan(al2)+pi/2));

                              //y_cur = (R_mbc*sin(th_cen_mbc))-yd;
                              //compute r_dist for the next target point

                              th_tar=(th_cen_mbc) - 5*pi/180;
                              x_tar = 29*1.2*cos(th_tar)-xd;
                              y_tar = (29*1.2*sin(th_tar))-yd;
                              th_tar = (th_tar*180/pi)-90;
      //convert to degrees;
                              ////////////////////////////


                        }

                  }
                  else  // it's a corner seen from zone#1
                  {
                        //locate intersection pt. between line1(y=al*x+bl) and
a perpendicular line which pass the occ. corner pt.(t_end)
                        if(segment[temp_index][4]==1)//if SICK can see the
corner directly
                        {
                              case_no = 2;
                              t_end=segment[temp_index][1]-1;

                              x_ref=x[t_end];               //corresponds to
ths END angle
                              y_ref=y[t_end];

                              if ((al<0.008)&&(al>-0.008))        //if the
line almost parallel to x-axis (-0.5<th<0.5 deg.)
                              {
                                    x_pos=x_ref;
                                    y_pos=bl;
                              }
                              else
                              {
                                    x_pos=((x_ref/al)+y_ref-bl)/(al+(1/al));
                                    y_pos=al*x_pos+bl;
                              }
                              x_ref = x_pos;// assign temp. values back to
original parameter
                              y_ref = y_pos;
```

```
                              //locate intersection pt. between
line1(y=al*x+bl) and a perpendicular line which pass the origin(location of
SICK)
                              if ((al<0.008)&&(al>-0.008))        //if the
line almost parallel to x-axis (-0.5<th<0.5 deg.)
                              {
                                    x_pos=0;
                                    y_pos=bl;
                              }
                              else
                              {
                                    x_pos=-bl/(al+(1/al));
                                    //y_pos=bl;
                                    //y_pos = -(x_pos/al);
                                    y_pos = (x_pos*al)+bl;
                              }
                              //find the distance between the two points
(x_ref, y_ref) & (x_pos, y_pos)
                              //x_cur=-(float)sqrt(((x_ref-x_pos)*(x_ref-
x_pos))+((y_ref-y_pos)*(y_ref-y_pos)));

      //y_cur=(float)sqrt((x_pos*x_pos)+(y_pos*y_pos));
                              //th_cur = -atan(al)*180/pi;

                              //locate current position
                              x_cur=-(float)sqrt(((x_ref-x_pos)*(x_ref-
x_pos))+((y_ref-y_pos)*(y_ref-y_pos)));
                              th_cur = atan(al);

      y_cur=(float)sqrt((x_pos*x_pos)+(y_pos*y_pos))+(sick_offset*cos(th_cur))
;
                              th_cur = -atan(al)*180/pi;

                              //locate target point using straight line eq.
                              //x_tar=x_cur+progress_dist;
                              //y_tar=-0.3*x_tar+8.6; //in.
                              //th_tar=-atan(0.3)*180/pi;

                              //locate target position using an arc//////
                              //initialize the  values of xd & ud;

                              xd=20*1.2;
                              yd=15*1.2;

                              xc=x_ref +
sqrt((xd*xd)+(yd*yd))*cos(atan(al)+atan(yd/xd));
                              yc=y_ref +
sqrt((xd*xd)+(yd*yd))*sin(atan(al)+atan(yd/xd));

                              //compute target angle and target distance

                              R_mbc =
sqrt((xc*xc)+((yc+sick_offset)*(yc+sick_offset)));
                              th_cen_mbc = (atan((yc+sick_offset)/xc)-
atan(al));

                              //y_cur = (R_mbc*sin(th_cen_mbc))-yd;
```

```
                              //compute r_dist for the next target point

                              th_tar=(th_cen_mbc) - 5*pi/180;
                              x_tar = 29*1.2*cos(th_tar)-xd;
                              y_tar = (29*1.2*sin(th_tar))-yd;
                              th_tar = (th_tar*180/pi)-90;
      //convert to degrees;
                              ////////////////////////////

                        }
                        else if(segment[temp_index][3] == 1)
                        {
                              //al = al2; //return values of al2 and bl2
instead
                              //bl = bl2;
                              //locate intersection pt. between
line1(y=al*x+bl) and a perpendicular line which pass the origin(location of
SICK)
                              case_no = 3;
                              t_start=segment[temp_index][0]+1;

                              x_ref=x[t_start];             //corresponds to
ths START angle
                              y_ref=y[t_start];

                              if ((al2<0.008)&&(al2>-0.008))           //if
the line almost parallel to x-axis (-0.5<th<0.5 deg.)
                              {
                                    x_pos=x_ref;
                                    y_pos=bl2;
                              }
                              else
                              {
                                    x_pos=((x_ref/al2)+y_ref-
bl2)/(al2+(1/al2));
                                    y_pos=al2*x_pos+bl2;
                              }
                              x_ref = x_pos;// assign temp. values back to
original parameter
                              y_ref = y_pos;


                              if ((al2<0.008)&&(al2>-0.008))           //if
the line almost parallel to x-axis (-0.5<th<0.5 deg.)
                              {
                                    x_pos=0;
                                    y_pos=bl2;
                              }
                              else
                              {
                                    x_pos=-bl2/(al2+(1/al2));
                                    //y_pos=bl2;
                                    //y_pos = -(x_pos/al2);
                                    y_pos = (x_pos*al2)+bl2;
                              }
                              //find the distance between the two points
(x_ref, y_ref) & (x_pos, y_pos)
```

```
                              //x_cur=-2000;    // return 1000 to alert other
function

      //y_cur=(float)sqrt((x_pos*x_pos)+(y_pos*y_pos));
                              //th_cur = -atan(al)*180/pi;


                              //locate current position
                              y_cur=-(float)sqrt(((x_ref-x_pos)*(x_ref-
x_pos))+((y_ref-y_pos)*(y_ref-y_pos)));   // return 1000 to alert other
function
                              th_cur = atan(al2);

      x_cur=(float)sqrt((x_pos*x_pos)+(y_pos*y_pos))+(sick_offset*cos(th_cur))
;
                              th_cur = -atan(al2)*180/pi-(90);
                              //locate target point using straight line eq.
                              //x_tar=x_cur+progress_dist;
                              //y_tar=(float)15.80; //in.
                              //th_tar=0;


                              //initialize the  values of xd & ud;

                              xd=20*1.2;
                              yd=15*1.2;

                              xc=x_ref +
sqrt((xd*xd)+(yd*yd))*cos(atan(al2)+atan(yd/xd)+pi/2);
                              yc=y_ref +
sqrt((xd*xd)+(yd*yd))*sin(atan(al2)+atan(yd/xd)+pi/2);

                              //compute target angle and target distance

                              R_mbc =
sqrt((xc*xc)+((yc+sick_offset)*(yc+sick_offset)));
                              th_cen_mbc = (atan2((yc+sick_offset),xc)-
(atan(al2)+pi/2));

                              //y_cur = (R_mbc*sin(th_cen_mbc))-yd;
                              //compute r_dist for the next target point

                              th_tar=(th_cen_mbc) - 5*pi/180;
                              x_tar = 29*1.2*cos(th_tar)-xd;
                              y_tar = (29*1.2*sin(th_tar))-yd;
                              th_tar = (th_tar*180/pi)-90;
      //convert to degrees;
                              ////////////////////////////
                        }
                  }

            }
            else
            {//NO, it's a straight line, locate the corner directly by using
an occluding angle
```

```
                  if
((segment[temp_index][3]==1)&&(segment[temp_index][4]==1))  // in case of
chamfer in zoon#2
                  {
                        case_no = 4;
                        x_ref =
x_l[(int)(segment[temp_index][0]+segment[temp_index][1])/2];  // use point
locate at the middle of the group
                        y_ref =
y_l[(int)(segment[temp_index][0]+segment[temp_index][1])/2];

                        if ((al<0.008)&&(al>-0.008))        //if the line
almost parallel to x-axis (-0.5<th<0.5 deg.)
                        {
                              x_pos=0;
                              y_pos=bl;
                        }
                        else
                        {
                              x_pos=-bl/(al+(1/al));
                              //y_pos=bl;
                              //y_pos = -(x_pos/al);
                              y_pos = (x_pos*al)+bl;
                        }
                        //find the distance between the two points (x_ref,
y_ref) & (x_pos, y_pos)
                        x_cur = (float)sqrt(((x_ref-x_pos)*(x_ref-
x_pos))+((y_ref-y_pos)*(y_ref-y_pos)));   //LOOP TO LOCATE THE CENTER OF THE
ARC....
                        th_cur = -atan(al);

      y_cur=(float)sqrt((x_pos*x_pos)+(y_pos*y_pos))+(sick_offset*cos(th_cur))
;
                        th_cur = -atan(al)*180/pi;
                        //initialize the  values of xd & ud;

                        xd=20*1.2;
                        yd=15*1.2;

                        xc=x_ref +
sqrt((xd*xd)+(yd*yd))*cos(atan(al)+atan(yd/xd));
                        yc=y_ref +
sqrt((xd*xd)+(yd*yd))*sin(atan(al)+atan(yd/xd));

                        //compute target angle and target distance

                        R_mbc =
sqrt((xc*xc)+((yc+sick_offset)*(yc+sick_offset)));

                        th_cen_mbc = (atan((yc+sick_offset)/xc)-atan(al));

                        //y_cur = (R_mbc*sin(th_cen_mbc))-yd;


                        //compute r_dist for the next target point

                        th_tar=(th_cen_mbc) - 5*pi/180;
```

```
                        x_tar = 29*1.2*cos(th_tar)-xd;

                        y_tar = (29*1.2*sin(th_tar))-yd;

                        th_tar = (th_tar*180/pi)-90;
      //convert to degrees;


                  }
                  else if(segment[temp_index][4]==1)//if SICK can see the
corner directly
                  {
                        case_no = 5;
                        t_end=segment[temp_index][1]-1;

                        x_ref=x_l[t_end];             //corresponds to ths END
angle
                        y_ref=y_l[t_end];

                        if ((al<0.008)&&(al>-0.008))        //if the line
almost parallel to x-axis (-0.5<th<0.5 deg.)
                        {
                              x_pos=x_ref;
                              y_pos=bl;
                        }
                        else
                        {
                              x_pos=((x_ref/al)+y_ref-bl)/(al+(1/al));
                              y_pos=al*x_pos+bl;
                        }
                        x_ref = x_pos;// assign temp. values back to original
parameter
                        y_ref = y_pos;

                        if ((al<0.008)&&(al>-0.008))        //if the line
almost parallel to x-axis (-0.5<th<0.5 deg.)
                        {
                              x_pos=0;
                              y_pos=bl;
                        }
                        else
                        {
                              x_pos=-bl/(al+(1/al));
                              //y_pos=bl;
                              //y_pos = -(x_pos/al);
                              y_pos = (x_pos*al)+bl;
                        }

                        //find the distance between the two points (x_ref,
y_ref) & (x_pos, y_pos)
                        //x_cur=-(float)sqrt(((x_ref-x_pos)*(x_ref-
x_pos))+((y_ref-y_pos)*(y_ref-y_pos)));
                        //y_cur=(float)sqrt((x_pos*x_pos)+(y_pos*y_pos));
                        //th_cur = -atan(al)*180/pi;

                        //locate current position
```

```
                              x_cur=-(float)sqrt(((x_ref-x_pos)*(x_ref-
x_pos))+((y_ref-y_pos)*(y_ref-y_pos)));
                              th_cur = -atan(al);

      y_cur=(float)sqrt((x_pos*x_pos)+(y_pos*y_pos))+(sick_offset*cos(th_cur))
;
                              th_cur = -atan(al)*180/pi;

                              //locate target point
                              x_tar=x_cur+progress_dist;
                              y_tar=19;//-0.3*x_tar+8.6; //in.
                              th_tar=0;//-atan(0.3)*180/pi;
                  }
                  else if(segment[temp_index][3]==1)//if SICK can not see the
corner
                  {
                        case_no = 6;
                        //MBC should do the followings
                        //1.eliminate the pig angle,so that it can see the
corner
                        //2.parallel itself to the wall or a bit positive
angle
                        if ((al<0.008)&&(al>-0.008))        //if the line
almost parallel to x-axis (-0.5<th<0.5 deg.)
                        {
                              x_pos=0;
                              y_pos=bl;
                        }
                        else
                        {
                              x_pos=-bl/(al+(1/al));
                              //y_pos=bl;
                              //y_pos = -(x_pos/al);
                              y_pos = (x_pos*al)+bl;
                        }
                        x_cur=-2000;      // return 1000 to alert other
function
                        th_cur = atan(al);

      y_cur=(float)sqrt((x_pos*x_pos)+(y_pos*y_pos))+(sick_offset*cos(th_cur))
;
                        th_cur = -atan(al)*180/pi;

                        //locate target point
                              x_tar=x_cur+progress_dist;
                              y_tar=(float)19; //in.
                              th_tar=0;
                  }
                  else //There is no occ. corner, this group of data must be
taken in zone#1 of long mine pillar.
                         //Therefore, we fit the line from start to end angle
                  {
                        case_no = 7;
                        //Compute the line's equation
                        sum_x=0;
                        sum_y=0;
                        sum_xsq=0;
```

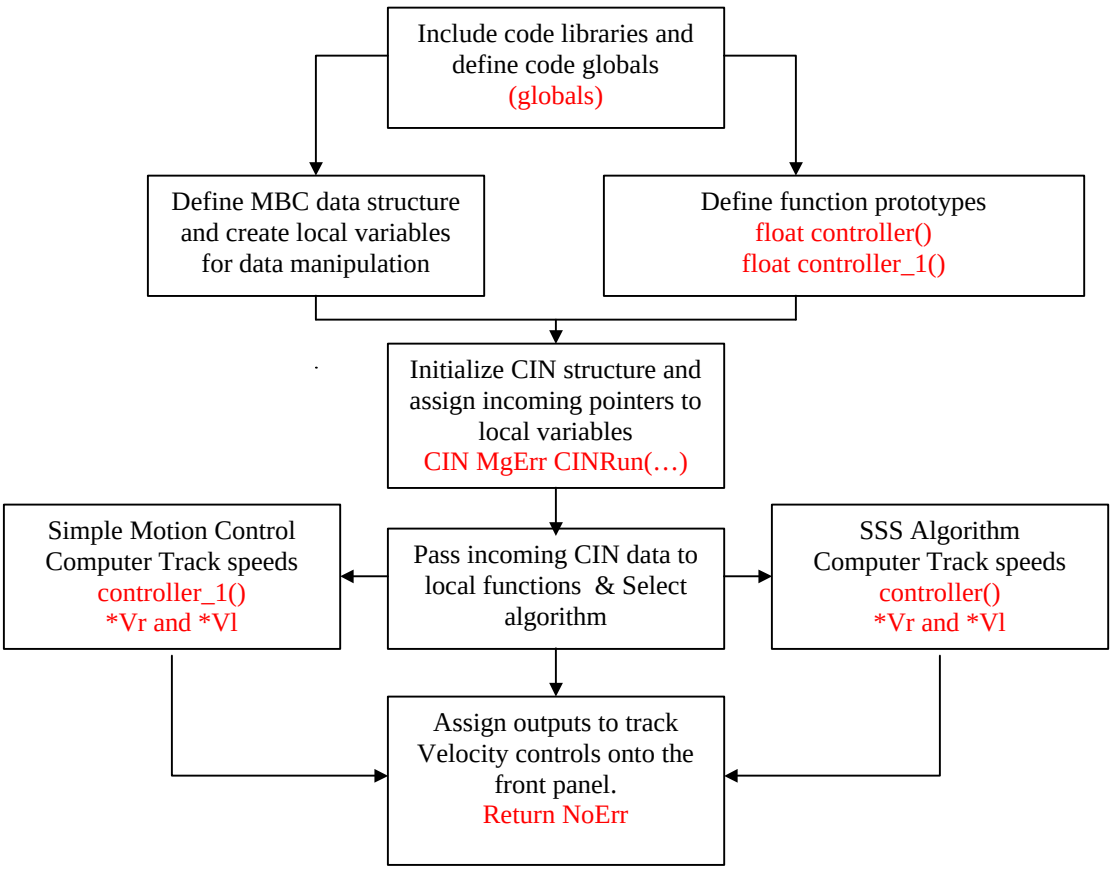
```
                        sum_ysq=0;
                        sum_xy=0;

                        no_data=abs(end-start)+1;

                        for(k=start;k<=(int)end;k++)
                        {
                              sum_x = sum_x + x_l[k];
                              sum_y = sum_y + y_l[k];
                              sum_xsq = sum_xsq + (x_l[k]*x_l[k]);
                              sum_ysq = sum_ysq + (y_l[k]*y_l[k]);
                        sum_xy = sum_xy + (x_l[k]*y_l[k]);
                        }


                        //compute the line parameters

                        if((sum_xsq*no_data-sum_x*sum_x)>(sum_ysq*no_data-
sum_y*sum_y))
                        {
                              al=((no_data*sum_xy)-
(sum_x*sum_y))/((no_data*sum_xsq)-(sum_x*sum_x));
                              bl=(sum_y - (al*sum_x))/(no_data);
                        }
                        else
                        {
                              al=(no_data*sum_ysq-
sum_y*sum_y)/(no_data*sum_xy-sum_x*sum_y);
                              bl=-(sum_x-(sum_y/al))*(al/no_data);
                        }

                        if ((al<0.008)&&(al>-0.008))        //if the line
almost parallel to x-axis (-0.5<th<0.5 deg.)
                        {
                              x_pos=0;
                              y_pos=bl;
                        }
                        else
                        {
                              x_pos=-bl/(al+(1/al));
                              //y_pos=bl;
                              //y_pos = -(x_pos/al);
                              y_pos = (x_pos*al)+bl;

                        }

                        x_cur=-1000;      // return 1000 to alert other
function
                        th_cur = atan(al);

      y_cur=(float)sqrt((x_pos*x_pos)+(y_pos*y_pos))+(sick_offset*cos(th_cur))
;
                        th_cur = -atan(al)*180/pi;

                        //locate target point
                              x_tar=x_cur+progress_dist;
                              y_tar=(float)19; //in.
```

```
                              th_tar=0;
                        }
                  }
}


//function to determine the smallest range and corresponding angle

float smallest(TD1Hdl in, int rows, int cols, int side)
{
      float output;

      output=(*in)->arg1[1];  //assign to the first range - arg1[1]


      for(i=0;i<rows;i++)
      {
            //arg1[odd] - range & arg1[even] contains the range and the angle
            //- alternate each other

            if(output>((*in)->arg1[i*cols + 1]))
            {
                  output=(*in)->arg1[i*cols + 1];
                  if (side == 1)
                        a_small = (*in)->arg1[i*cols+0] - 270;
                  // For RIGHT PMS
                  else if (side == 2)
                        a_small = (*in)->arg1[i*cols+0] - 90;
                  // For LEFT PMS
            }
      }

      return output;

}


//function to check the value of angle then return the angle range from 0 to
//2pi

float check_theta(float th)
{
      if( th < 0 )
            th = th+2*pi;
      return th;
}
```

iv. Implementation of the 'Simple motion control' & SSS algorithm

The object code listed in this section implements the 'Simple motion control' algorithm and the SSS discussed in Section 7.3.1 & 7.3.2 of Vol I of this document. The algorithms are used to position and orient the MBC by computing the track velocities accordingly. These algorithms can be used for the control of a Single unit system, and with simultaneous operation of multiple versions, the algorithm can be used to control a multi-unit system. The code listed here accommodates the PC-PC data sharing routine, and is the code that is used in conjunction with the Control Interface described under Appendix II. It must be noted that the selection of the algorithm (simple motion control and SSS) is done by calling the appropriate function in the code. It is the responsibility of the developer to use the appropriate function and remark the one that is not being used. Calling both the functions simultaneously will result in abnormal driving.

Code Structure



Variable List

The following table lists the variables for both the implementation of the 'Simple motion control algorithm' and the 'SSS' algorithm as both these methods are implemented by one source code. Selction of the appropriate methodology is carried out by the developer.

Variable name	Data type	Declaration	Comments
dist	float	Global	Distance from the wall
t_r	float	Global	A temporary range
t_vr	double	Global	Right track velocity
t_vl	double	Global	Left track velocity
r_90	float	Global	Range corresponding to 90 deg
th_90	float	Global	Smallest angle
thmbc	float	Global	Angle of MBC
d_err	float	Global	Offset = dist – t_r
r1	float	Global	Distance at head of MBC
r2	float	Global	Distance at tail of MBC
len	float	Global	Length of MBC
th_p	float	Global	In-by PIG angle
curr_pos	float	Global	Current position of MBC
st_angle	float	Global	Start angle of line
en_angle	float	Global	End angle of line
slider	float	Global	Dolly distance
th_go	float	Global	Angle to move next
x_go	float	Global	Target X distance
y_go	float	Global	Target Y distance
th_cur	float	Global	Current MBC Angle

x_cur	float	Global	Current X distance
y_cur	float	Global	Current Y distance
B = 10.00	float	Global	Wheel Base - Track - Track center distance
dist_travel	float	Global	linear distance to travel
th_travel	float	Global	angle to pivot
Velo	float	Global	Track velocities – SSS algorithm
delta_x	float	Global	Incremental X distance (SSS)
delta_y	float	Global	Incremental Y distance (SSS)
rear_pig	float	Global	Out-by PIG angle
d_mode	LVBoolean	Global	Mode of driving – For/Rev
debug	LVBoolean	Global	Debug button
thmax	float	Global	Maximum possible pivot angle
float D	float	Global	
UP_front_pig	float	Global	Inter MBC data
UP_rear_pig	float	Global	Inter MBC data
UP_dolly	float	Global	Inter MBC data
UP_bl	float	Global	Inter MBC data
UP_al	float	Global	Inter MBC data
UP_ar	float	Global	Inter MBC data
UP_br	float	Global	Inter MBC data
UP_vr	float	Global	Inter MBC data
UP_vl	float	Global	Inter MBC data
UP_RFM	float	Global	Inter MBC data
DOWN_front_pig	float	Global	Inter MBC data

DOWN_rear_pig	float	Global	Inter MBC data
DOWN_dolly	float	Global	Inter MBC data
DOWN_bl	float	Global	Inter MBC data
DOWN_al	float	Global	Inter MBC data
DOWN_ar	float	Global	Inter MBC data
DOWN_br	float	Global	Inter MBC data
DOWN_vr	float	Global	Inter MBC data
DOWN_vl	float	Global	Inter MBC data
DOWN_RFM	float	Global	Inter MBC data
Inter	LVBoolean	Global	MBC Configuration Selector
First	LVBoolean	Global	MBC Configuration Selector
Last	LVBoolean	Global	MBC Configuration Selector
MBC_data	Td1Hdl	Global	Incoming MBC Data

Function list

Function name	Argument	Return value & type	Comments
controller	Global Scanner data	float, Global assignment to *Vr and *Vl	Computes track velocities – Implements the SSS algorithm
controller_1	Global scanner data	float, Global assignment to *Vr and *Vl	Computes track velocities – Implements the SSS algorithm
CINProperties	int32 prop, void *data	CIN MgErr	Converts the CIN code to an re-entrant code
CINRun	TD1Hdl var1, float32 *cutoff, float64 *accuracy, float64 *a, float64 *b, float64 *small, float32 *th_small, float32 *s_ang, float32 *e_ang	CIN MgErr	The main input-output function. This is the CIN call back routine that processes all the I/O operations.

Complete Code listing

```
//-------------------------------------------------------------------------
//                MBC VELOCITY CONTROL USING PMS DATA - 2 MBC
//              -------------------------------------------
//
//    AUTHOR                  : Aish
//    DATE                    : Jan 13, 2000.
//    SUPPORT                 : Bruce & Mike
//    DEPT                    : Long Airdox & Virginia Tech
//    NOTES                   :
//                             - Aish, Mike & Bruce's UT'd MBC driving rule
//                             - modified for the SICK
//                             - NOTE : THE SICK READS FROM HEAD TO TAIL
//-------------------------------------------------------------------------

#include "extcode.h"
#include<math.h>
#include<stdio.h>

#define pi 3.14159
#define ang_thresh 0
#define dist_thresh 1

//global variables

float dist;
float t_r;                  //temp range
double t_vr;                //MBC Track velocities
double t_vl;
float r_90;                 //the range corresponding to 90.
float th_90;
float thmbc;                //angle of MBC w.r.t vertical thro MBC
float d_err;                //the error in the distance - dist-t_r;
float r1;                   //the distance at the HEAD
float r2;                   //the distance at the TAIL
float len;                  //the length of the MBC – 24“ for the prototype
float th_p;                 //The rear pig angle
float curr_pos;             //referred from the refernce point
float st_angle;             //start angle of the line
float en_angle;             //end angle of the line
float slider;               //dolly travel - fully retracted is zero
float th_go;                //target angle;
float x_go;                 //target x distance to go next
float y_go;                 //target y distance to go next
float th_cur;               //current MBC angle
float x_cur;                //current x distance
float y_cur;                //current y distance
float B = 10.00;            //Wheel Base - Track - Track center distance
float dist_travel;          //linear distance to travel
float th_travel;            //angle to pivot
float velo;                 //track velocities
float delta_x;              //incremental X distance
float delta_y;              //incremental Y distance
float rear_pig;             // angle at front of pig
LVBoolean d_mode;           //the direction of travel
```

```
LVBoolean debug;              //the debug switch
float thmax;
float D;



// GLOBAL DATA FOR INTER MBC DATA..

float UP_front_pig;           // Front PIG angle of CHAIN UP MBC
float UP_rear_pig;            // Rear PIG angle of CHAIN UP MBC
float UP_dolly;               // Dolly distance of the CHAIN UP
float UP_bl;                  // Line data of CHAIN UP MBC - Left Side
float UP_al;
float UP_ar;                  // Line data of CHAIN UP MBC - Right Side
float UP_br;
float UP_vr;                  // Right track velocity of CHAIN UP MBC
float UP_vl;                  // Left track velocity of CHAIN UP MBC
float UP_RFM;                 // CHAIN UP RFM DATA
float DOWN_front_pig;         // Front PIG angle of CHAIN DOWN MBC
float DOWN_rear_pig;          // Rear PIG angle of CHAIN DOWN MBC
float DOWN_dolly;             // Dolly distance of the CHAIN DOWN
float DOWN_bl;                // Line data of CHAIN DOWN MBC - Left Side
float DOWN_al;
float DOWN_ar;                // Line data of CHAIN DOWN MBC - Right Side
float DOWN_br;
float DOWN_vr;                // Right track velocity of CHAIN DOWN MBC
float DOWN_vl;                // Left track velocity of CHAIN DOWN MBC
float DOWN_RFM;               // CHAIN DOWN RFM DATA

// Boolean control variables...

LVBoolean Inter;              // Intermediate MBC Boolean data
LVBoolean First;              // Lead MBC Boolean
LVBoolean Last;               // Rear MBC Boolean

typedef struct {              // MBC Data Structure
      int32 dimSize;
      float64 Output_number[1];
      } TD1;
typedef TD1 **TD1Hdl;

TD1Hdl MBC_data;              // Incoming data handler



//function to convert the CIN as "THREAD SAFE"
//This converts the CIN to be re-entrant -> called many no of times or if
//called by several routines..

float controller();           // Implements the SSS Algorithm
float controller_1();         // RUNS THE SIMPLE MOTION CONTROL CODE

CIN MgErr CINRun( float64 *a,
                  float64 *b,
                  float64 *small,
                  float32 *th_small,
                  float64 *r_dist,
```

```
                  float64 *var6,
                  LVBoolean *mode,
                  float32 *s_ang,
                  float32 *e_ang,
                  float64 *var10,
                  float64 *th_goal,
                  float64 *x_target,
                  float64 *y_target,
                  float64 *th_current,
                  float64 *x_current,
                  float64 *y_current,
                  float64 *var17,
                  float64 *a_l,
                  float64 *b_l,
                  float64 *small_l,
                  float64 *th_small_l,
                  float64 *mbc_pos,
                  float32 *mbc_ang,
                  float64 *Vr,
                  float64 *Vl,
                  LVBoolean *Inter_MBC,
                  LVBoolean *First_MBC,
                  LVBoolean *Last_MBC,
                  TD1Hdl var29);

CIN MgErr CINProperties(int32 prop, void *data);

void MBC();                   // MBC Data handling function....

CIN MgErr CINProperties(int32 prop, void *data)
{
      switch(prop)
      {
      case kCINIsReentrant:
            *(Bool32*)data = TRUE;
            return noErr;
      }

      return mgNotSupported;
}


CIN MgErr CINRun(float64 *a,
                         float64 *b,
                         float64 *small,
                         float32 *th_small,
                         float64 *r_dist,
                         float64 *var6,
                         LVBoolean *mode,
                         float32 *s_ang,
                         float32 *e_ang,
                         float64 *var10,
                         float64 *th_goal,
                         float64 *x_target,
                         float64 *y_target,
                         float64 *th_current,
                         float64 *x_current,
```

```
                        float64 *y_current,
                        float64 *var17,
                        float64 *a_l,
                        float64 *b_l,
                        float64 *small_l,
                        float64 *th_small_l,
                        float64 *mbc_pos,
                        float32 *mbc_ang,
                        float64 *Vr,
                        float64 *Vl,
                        LVBoolean *Inter_MBC,
                        LVBoolean *First_MBC,
                        LVBoolean *Last_MBC,
                        TD1Hdl var29)

{

      //NOTE - *var6 is theta_p - PIG ANGLE
      //       - *var10 is dolly  - DOLLY TRAVEL
      //       - *var17 is rear pig angle of MBC 1


      /* ENTER YOUR CODE HERE */

      double x;
      double y;

      r_90=*small;
      th_90=*th_small;

      th_p=*var6;
      curr_pos=*r_dist;
      st_angle=*s_ang;
      en_angle=*e_ang;
      slider=*var10;
      th_go=*th_goal;
      x_go=*x_target;
      y_go=*y_target;
      th_cur=*th_current;
      x_cur=*x_current;
      y_cur=*y_current;
      rear_pig=*var17;

      MBC_data = var29;             // assign to the incoming data handle...
      Inter  = *Inter_MBC;
      First  = *First_MBC;
      Last   = *Last_MBC;

      //ASSIGN THE MODE TO THE INCOMING VARIABLE

      d_mode=*mode;

      //calculate the MBC angle

      thmbc=-(180/pi)*atan(*a);

      x=-((*b)*(*a))/(((*a) * (*a))+1);
```

```
      y=(*b)/(((*a) * (*a))+1);


      //calculate the distance from the wall

      t_r = sqrt((x*x)+(y*y));

      //calculate the HEAD/TAIL distances

//    r1 = (t_r+(len*sin(thmbc*pi/180))/2);     //distance at the HEAD

//    r2 = (t_r-(len*sin(thmbc*pi/180))/2);     //distance at the TAIL

      //IMPORTANT !!!!!

      //1. Include the r1/r2 clearance from the wall
      //2. Depending on the sign of thmbc, use the appropriate checks

//    th_travel=(th_go - th_cur);               //always in degrees

//    velo = (th_travel*pi/180)*(B/2)/(0.5); --> V = r*w = r*d(th)/d(t), 0.5
//    is update rate of sick -> 2 Hz

      //compute linear distance parameters

      delta_x = (x_go - x_cur);
      delta_y = (y_go - y_cur);

      //DISTANCE CALCULATION

      dist_travel = sqrt((delta_x*delta_x) + (delta_y*delta_y));

      if (fabs(dist_travel)>2)
            dist_travel = 2;

      //HEADING CALCULATION

//    th_go = atan((delta_y/delta_x))*180/pi;   //th_go is from the other CIN

      th_travel = th_go-th_cur;

      //flipping the sign of th_travel for the reverse case
      //forward is LVTRUE

//    if (d_mode==LVFALSE)    //flip signs for reverse
//    {
//          th_travel = 0-th_travel;
//    }

      D=y_cur;

//    thmax = ((0.0028*D*D/144)+(0.0526*D/12)-0.2525)*(180/pi);
// Equation for the FULLSCALE.... (eqn had D in feet!!)
      thmax = ((0.3546*D*D/144)+(0.5331*D/12)-0.3733)*(180/pi);
// Equation for the PROTOTYPE....
```

```
//    if (fabs(th_travel)>thmax)
//          th_travel = thmax*fabs(th_travel)/th_travel;

      if (fabs(th_travel)>15)
            th_travel = 15*fabs(th_travel)/th_travel; // Restrict max
                                                      //angle to 15 degrees...

      //check for either angle or distance correction

      //this is angle correction..

      if((th_travel>2)||(th_travel<-2))
      {
            velo = (th_travel*pi/180)*B/(0.5*2);      //B/2 is half the MBC
                                                      //width and 0.5 is the
                                                      //update rate of sick
      }

      //this is the linear distance correction

      else
      {
            velo = dist_travel/0.5;                   //just the distance to
                                                      //travel - linear
                                                      //correction...
      }

//    if(slider<=3)
//          d_mode=LVFALSE;
//    else
//          d_mode=LVTRUE;

      velo = (velo *1.75)/12;                         // For the prototype -
                                                      //Scale the speed......

// Command fullscale unit at fullspeed and compute time for 12 inches travel

MBC();                                                // call the data
                                                      //handling function

controller_1();                                       // call the temp fn

*mbc_pos=t_r;
*mbc_ang=thmbc;
*Vr=t_vr*0.4;                                         //Reduce the output
                                                      //velocity by
                                                      //multiplying by a
                                                      //reduction factor
*Vl=t_vl*0.4;


      return noErr;

}
```

```
void MBC()
{
	// assign the MBC data to the global variables...

	UP_front_pig	= (*MBC_data)->Output_number[0];
	UP_rear_pig	= (*MBC_data)->Output_number[1];
	UP_dolly	= (*MBC_data)->Output_number[2];
	UP_bl		= (*MBC_data)->Output_number[3];
	UP_al		= (*MBC_data)->Output_number[4];
	UP_ar		= (*MBC_data)->Output_number[5];
	UP_br		= (*MBC_data)->Output_number[6];
	UP_vr		= (*MBC_data)->Output_number[7];
	UP_vl		= (*MBC_data)->Output_number[8];
	UP_RFM		= (*MBC_data)->Output_number[9];

	// Trail MBC Data

	DOWN_front_pig	= (*MBC_data)->Output_number[10];
	DOWN_rear_pig	= (*MBC_data)->Output_number[11];
	DOWN_dolly	= (*MBC_data)->Output_number[12];
	DOWN_bl		= (*MBC_data)->Output_number[13];
	DOWN_al		= (*MBC_data)->Output_number[14];
	DOWN_ar		= (*MBC_data)->Output_number[15];
	DOWN_br		= (*MBC_data)->Output_number[16];
	DOWN_vr		= (*MBC_data)->Output_number[17];
	DOWN_vl		= (*MBC_data)->Output_number[18];
	DOWN_RFM	= (*MBC_data)->Output_number[19];

	// BASED ON THE MBC SELECTION BOOL VARIABLES, sort the incoming data
	//accordingly and identify the configuration

}


float controller()
{

	//METHODOLOGY :  If the MBC's front pig angle is within a range, keep
	//going straight, else turn....if running straight, run until the MBC
	//crosses the intersection.... Determine this by using the line
	//data.....check for line data (a)...if between a range....


	if (th_travel>3)
	{
		t_vr = velo;
		t_vl = 0-velo;		//change the sign - t_vl will be negative

		if (d_mode==LVFALSE)	//flip signs for reverse
		{
			t_vr = 0-velo;
			t_vl = velo;
		}

		if (slider<0.5)
		{
			t_vr = 0;
```

```
                  t_vl = 0;
            }

      }

      else if (th_travel<-3)
      {
            t_vr = velo;
            t_vl = 0-velo;          //t_vl will be positive

            if (d_mode==LVFALSE)    //flip signs for reverse
            {
                  t_vr = 0-velo;
                  t_vl = velo;
            }

            if (slider<0.5)         // This values is in inches, make changes
                                    // for fullscale......
            {
                  t_vr = 0;
                  t_vl = 0;
            }

      }

      else if ((th_travel<3)&&(th_travel>-3))
      {
            t_vr = 1;//fabs(velo);
            t_vl = 1;//fabs(velo);

            if(d_mode==LVFALSE)
            {
                  t_vr = -fabs(velo);
                  t_vl = -fabs(velo);
            }

            if (slider<0.5)
            {
                  t_vr = 0;
                  t_vl = 0;
            }
      }


      return t_vr;
      return t_vl;
}

float controller_1()
{


dist = 14;                    //the setting distance as 28"
d_err=dist-t_r;               //the distance error
```

```
if (d_mode==LVTRUE)
      {
            t_vr=0.6;
            t_vl=0.6;


            if((thmbc>0)||(thmbc<0))      //thmbc is positive or negative
            {

                  if((thmbc<0)&&(d_err>0)) //d_err = -2"/2 and thmbc has -5/+5
                  {
                        t_vr=t_vr+0.4;
                        t_vl=t_vl-0.6;
                  }

                  else if((thmbc>0)&&(d_err>0))
                  {
                        t_vr=t_vr+0.4;
                        t_vl=t_vl+0.4;
                  }

                  else if((thmbc<0)&&(d_err<0))
                  {
                        t_vr=t_vr+0.4;
                        t_vl=t_vl+0.4;
                  }

                  else if((thmbc>0)&&(d_err<0))
                  {
                        t_vr=t_vr-0.6;
                        t_vl=t_vl+0.4;
                  }
            }

            else
            {
                  t_vr=1;
                  t_vl=1;
            }

            if (slider<0.5)
            {
                  t_vr = 0;
                  t_vl = 0;
            }

      }



      //REVERSE MODE

      //maintains the same philospohy as the forward mode
      //have NOT switched the velocities - right and left are same as forward
      //mode have manipulated on the velocity commands....
```

```
	else
	{
		t_vr=-0.6;
		t_vl=-0.6;


		if((thmbc>0)||(thmbc<0))	//thmbc is positive or negative
		{

			if((thmbc<0)&&(d_err>1)) //d_err = -2"/2 and thmbc has -5/+5
			{
				t_vr=t_vr-0.4;
				t_vl=t_vl-0.4;
			}

			else if((thmbc>0)&&(d_err>1))
			{
				t_vr=t_vr-0.4;
				t_vl=t_vl+0.4;
			}

			else if((thmbc<0)&&(d_err<-1))
			{
				t_vr=t_vr+0.4;
				t_vl=t_vl-0.4;
			}

			else if((thmbc>0)&&(d_err<-1))
			{
				t_vr=t_vr-0.4;
				t_vl=t_vl-0.4;
			}
		}

		else
		{
			t_vr=-1;
			t_vl=-1;
		}

		if (slider>5)
		{
			t_vr = 0;
			t_vl = 0;
		}
	}


	return t_vr;
	return t_vl;
}
```