REFERENCES:

[1] Hadi M., "Autopilot Design Based on Fuzzy Supervisory Controller for AUS-Autonomous Vehicle", Master Thesis, Spring 2005, American University Sharjah

[2] Weal S., Mahmoud G., "Implementation of Ground Station to Control Simulated UAV", Senior Design Project, Fall 2004, American University of Sharjah

[3] Egerstedt M., Koo T. J., Hoffmann F., Sastry S., "An Integrated Approach to Path Planning and Flight Controller Scheduling for an Autonomous Helicopter". Proceedings of the 7th Mediterranean Conference on Control and Automation, August 1999, Haifa, Israel

[4] Beard R., Kingston D., Quigley M., Snyder D., Christiansen R., Johnson W., McLain T., Goodrich M., "Autonomous Vehicle Technologies for Small Fixed Wing UAV". AIAA Journal of Aerospace Computing, Information, and Communication (to appear)

[5] Stefano G., Tranchero, B., "Concurrent constraint programming based path planning for uninhabited air vehicles". SPIE/Defense and Security Symposium on Unmanned Ground, Ocean, and Air Technologies, April 2004, pg 12 – 16

[6] Rem W., Beard R. W., "Constrained Nonlinear Tracking Control for Small Fixed wing Unmanned Air Vehicles". American Control Conference, 2004, Boston, MA, pg: 4663 – 4668.

[7] Kingston D. B., "Implementation Issue of Real-Time Trajectory Generation of Small UAV". Master Thesis, 2004, Birmingham Young University

[8] Egziabher D. G., Power J. D., Enge P. K., "Design and Performance Analysis of a Low Cost Aided Dead Reckoning Navigation System". Gyroscope and Navigation, 2001, Volume 4, No: 35, pg: 83 - 92

[9] Kim J. H., Wishart S., Sukkarieh S., "Real Time Navigation, Guidance, and Control of an UAV using Low cost Sensors." International Conference of Field and Service Robotics (FSR03), July 2003, Yamanashi, Japan, pg: 95 – 100.

[10] Hide C., Moore T., "GPS and Low Cost INS Integration for Positioning in the Urban Environment", Institute if Engineering Surveying and Space Geodesy, University of Nottingham, 2002, page 34-56

[11] Barbehenn M., "Note on the Complexity of Dijkstra's Algorithm for Graph with Weighted Vertices". IEEE Transactions on Computers, February 1998, Volume 47, Issue 2, pg 263.

[12] Graesslin M., Schoettle U., "A NLP based reentry flight guidance algorithm", 18th International Symposium on Space Flight Dynamics, October 2004, Munich Germany

[13] Bhatt S. P., Kumar P., “A Feedback Guidance Strategy for an Autonomous Mini-Air Vehicle”. National Conference on Control and Dynamical Systems, January 27-18, 2005, IIT Bombay.

[14] Harbick, K., Montgomery, J. F., Sukhatme, G. S., “Planar Spline Trajectory Following for an Autonomous Helicopter”. Journal of Advanced Computational Intelligence – Computational Intelligence in Robotics and Automation, May 2004, Volume 8, Issue 3, pg: 237 – 242

[15] Chean S. L., Doria, M., Chan, J., Parker S., Goatley, S., “Path Planning and High Level Control of an Unmanned Aerial Vehicle”. Ph.D. Thesis, Fall 2002, Spring University of Sidney”

[16] Lee J., Huang R., Vaughn A., Xiao X., Hedrick J. K., “Strategies of Path Planning for a UAV to Track a Ground Vehicle”. 2nd Annual Symposium on Autonomous Intelligent Networks and Systems, June/July 2003, Bologna, Italy

[17] Ryan A., “A mode-switching path planner for UAV-assisted search and rescue”, Master Thesis, Spring 2003, University of California, Berkeley

[18] Richards A., How J. P, “Model Predictive Control of Vehicle Maneuvers with Guaranteed Completion Time and Robust Feasibility”

[19] Kuwata Y., How J. P., “Three Dimensional Receding Horizon Control for UAVs”, AIAA Guidance, Navigation, and Control Conference and Exhibit, 16-19 August 2004, Providence, Rhode Island

[20] Vandapel N., Kuffner J., Amidi O., “Planning 3-D Path Networks in Unstructured Environments”, Carnegie Mellon University, 2003

[21] Chapman D., “SAMS Teach Yourself Visual C++ 6.0 in 21 days”, Macmillan Computer Publishing Corporation, ebook, published 2002.

[22] Titterton D., Weston J., “Strap down Inertial Navigation Technology”, 2nd Edition, part of IEEE Radar, Sonar and Navigation Series 17, Volume 5, Ch 3, 11-13, pg 17–58, 309–420

[23] Chatfield A. B., “Fundamentals of High Accuracy Inertial Navigation”, part of Progress in Astronautics and Aeronautics, Volume 174, 2004

[24] Jarrah M. A., Hasan M. M., “Hardware in the Loop Simulator for AUS-UAV System, 2nd Annual Symposium on Mechatronics, April 2005, American University of Sharjah, U.A.E. Published: Special Edition of the International Journal of Automation in Austria (IJAA), 2005

[25] Jarrah M. A., Hasan M. M., “HILS Setup of Dynamic Flight Path Planning in 3D Environment with Flexible Mission Planning using Ground Station”, 3rd Annual Symposium on Mechatronics, April 2006, American University of Sharjah, U.A.E. Published: Special Edition of Robotics and Mechatronics Journal, 2006

[26] Jarrah M. A., Dhabbas O., Saeed S. I., Hasan M. M., “Calibration of AUS-UAV Avionics Unitt”, 3rd Annual Symposium on Mechatronics, April 2006, American University of Sharjah, U.A.E. Published: Special Edition of Robotics and Mechatronics Journal, 2006

[27] Titterton D. H., Weston J. L, “Strapdown Inertial Navigation Technology”, Publisher: Stevenage, 2nd Edition, AIAA 2004, “IEEE Radar, Sonar, Navigation and Avionics” Series 17

APPENDIX A

GROUND STATION CODE

A-1: Main Dialog Box Code

```
// GroundStationDlg.cpp : implementation file
//

#include "stdafx.h"
#include "GroundStation.h"
#include "GroundStationDlg.h"
#include "DlgProxy.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif


const COLORREF CGroundStationDlg::m_crColors[29] = {
      RGB (000, 000, 000), //Black
      RGB (64, 64, 64),  //Dark Grey
      RGB (128, 128, 128), //Grey
      RGB (192, 192, 192), //Light Grey
      RGB (255, 255, 255), //White
      RGB (64, 000, 000),  //Dark Red
      RGB (128, 000, 000), //Red
      RGB (192, 000, 000), //Light Red
      RGB (255, 000, 000), //Bright Red
      RGB (000, 000, 64),  //Dark Blue
      RGB (000, 000, 128), //Blue
      RGB (000, 000, 192), //Light Blue
      RGB (000, 000, 255), //Bright Blue
      RGB (000, 64, 000),  //Dark Green
      RGB (000, 128, 000), //Green
      RGB (000, 192, 000), //Light Green
      RGB (000, 255, 000), //Bright Green
      RGB (000, 64, 64),   //Dark Cyan
      RGB (000, 128, 128), //Cyan
      RGB (000, 192, 192), //Light Cyan
      RGB (000, 255, 255), //Bright Cyan
      RGB (64, 000, 64),   //Dark Megenta
      RGB (128, 000, 128), //Megenta
      RGB (192, 000, 192), //Light Megenta
      RGB (255, 000, 255), //Bright Megenta
      RGB (64, 64, 000),   //Dark Yellow
      RGB (128, 128, 000), //Yellow
      RGB (192, 192, 000), //Light Yellow
      RGB (255, 255, 000), //Yellow
};



const int CGroundStationDlg::m_iaZoomValues[64] = {
      2048,1536,64,48,
      1600,1200,60,45,
      1280,960,56,42,
      1024,768,52,39,
      800,600,48,36,
```

```
	640,480,44,33,
	512,384,40,30,
	400,300,36,27,
	320,240,32,24,
	256,192,28,21,
	200,150,24,18,
	160,120,20,15,
	128,96,16,12,
	100,75,12,9,
	80,60,8,6,
	64,48,4,3
};


/////////////////////////////////////////////////////////////////////////
////////
// CAboutDlg dialog used for App About
class CAboutDlg : public CDialog
{
public:
	CAboutDlg();

// Dialog Data
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAboutDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
	//}}AFX_VIRTUAL


// Implementation
protected:
	//{{AFX_MSG(CAboutDlg)
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};


CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}


void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}



BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
```

```
	//{{AFX_MSG_MAP(CAboutDlg)
		// No message handlers
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()


/////////////////////////////////////////////////////////////////////////
////////
// CGroundStationDlg dialog
IMPLEMENT_DYNAMIC(CGroundStationDlg, CDialog);


CGroundStationDlg::CGroundStationDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CGroundStationDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CGroundStationDlg)
	m_strAirSpeed = _T("");
	m_strCommandInterface = _T("");
	m_strCompass = _T("");
	m_strAltimeter = _T("");
	m_strLatitude = _T("");
	m_strLongitude = _T("");
	m_strMap = _T("");
	m_strMapInterface = _T("");
	m_strYaw = _T("");
	m_strAusUavTitle = _T("");
	m_strFontList = _T("");
	m_dAHPitch = 0.0f;
	m_dAHRoll = 0.0f;
	m_strClimbRate = _T("");
	m_strTemp1 = _T("");
	m_strTemp2 = _T("");
	m_strTemp3 = _T("");
	m_iAltSelect = 0;
	m_fXRate = 0.0f;
	m_fYRate = 0.0f;
	m_fZRate = 0.0f;
	m_fSysTime = 0.0f;
	m_fXAcc = 0.0f;
	m_fYAcc = 0.0f;
	m_fZAcc = 0.0f;
	//}}AFX_DATA_INIT
	// Note that LoadIcon does not require a subsequent DestroyIcon
in Win32
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
	m_pAutoProxy = NULL;
	m_dPi = 3.14159265358979323846264338327795;
}


CGroundStationDlg::~CGroundStationDlg()
{
	// If there is an automation proxy for this dialog, set
	//  its back pointer to this dialog to NULL, so it knows
	//  the dialog has been deleted.
	if (m_pAutoProxy != NULL)
		m_pAutoProxy->m_pDialog = NULL;
}
```

```
void CGroundStationDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CGroundStationDlg)
	DDX_Control(pDX, IDC_ALTLEVEL, m_pcAltLevel);
	DDX_Control(pDX, IDC_ALTSELECT, m_scAltSelect);
	DDX_Control(pDX, IDC_TEMP3, m_eTemp3);
	DDX_Control(pDX, IDC_TEMP2, m_eTemp2);
	DDX_Control(pDX, IDC_TEMP1, m_eTemp1);
	DDX_Control(pDX, IDC_CLIMBRATE, m_eClimbRate);
	DDX_Control(pDX, IDC_FONTLIST, m_lbFontList);
	DDX_Control(pDX, IDC_YAW, m_eYaw);
	DDX_Control(pDX, IDC_MAPINTERFACE, m_statMapInterface);
	DDX_Control(pDX, IDC_MAP, m_statMap);
	DDX_Control(pDX, IDC_LONGITUDE, m_eLongitude);
	DDX_Control(pDX, IDC_LATITUDE, m_eLatitude);
	DDX_Control(pDX, IDC_COMPASS, m_statCompass);
	DDX_Control(pDX, IDC_COMMANDINTERFACE, m_statCommandInterface);
	DDX_Control(pDX, IDC_ALTIMETER, m_statAltimeter);
	DDX_Control(pDX, IDC_AIRSPEED, m_statAirSpeed);
	DDX_Control(pDX, IDC_BATTERY3POWER, m_pcBattery3Power);
	DDX_Control(pDX, IDC_BATTERY2POWER, m_pcBattery2Power);
	DDX_Control(pDX, IDC_BATTERY1POWER, m_pcBattery1Power);
	DDX_Control(pDX, IDC_AUSUAVTITLE, m_statAusUavTitle);
	DDX_Text(pDX, IDC_AIRSPEED, m_strAirSpeed);
	DDX_Text(pDX, IDC_COMMANDINTERFACE, m_strCommandInterface);
	DDX_Text(pDX, IDC_COMPASS, m_strCompass);
	DDX_Text(pDX, IDC_ALTIMETER, m_strAltimeter);
	DDX_Text(pDX, IDC_LATITUDE, m_strLatitude);
	DDX_Text(pDX, IDC_LONGITUDE, m_strLongitude);
	DDX_Text(pDX, IDC_MAP, m_strMap);
	DDX_Text(pDX, IDC_MAPINTERFACE, m_strMapInterface);
	DDX_Text(pDX, IDC_YAW, m_strYaw);
	DDX_Text(pDX, IDC_AUSUAVTITLE, m_strAusUavTitle);
	DDX_LBString(pDX, IDC_FONTLIST, m_strFontList);
	DDX_Control(pDX, IDC_HORIZON, m_aHorizon);
	DDX_OCFloat(pDX, IDC_HORIZON, DISPID(72), m_dAHPitch);
	DDX_OCFloat(pDX, IDC_HORIZON, DISPID(73), m_dAHRoll);
	DDX_Text(pDX, IDC_CLIMBRATE, m_strClimbRate);
	DDX_Text(pDX, IDC_TEMP1, m_strTemp1);
	DDX_Text(pDX, IDC_TEMP2, m_strTemp2);
	DDX_Text(pDX, IDC_TEMP3, m_strTemp3);
	DDX_Slider(pDX, IDC_ALTSELECT, m_iAltSelect);
	DDX_Text(pDX, IDC_XRate, m_fXRate);
	DDV_MinMaxFloat(pDX, m_fXRate, -150.f, 150.f);
	DDX_Text(pDX, IDC_YRate, m_fYRate);
	DDV_MinMaxFloat(pDX, m_fYRate, -150.f, 150.f);
	DDX_Text(pDX, IDC_ZRate, m_fZRate);
	DDV_MinMaxFloat(pDX, m_fZRate, -150.f, 150.f);
	DDX_Text(pDX, IDC_SYSTIME, m_fSysTime);
	DDX_Text(pDX, IDC_XACC, m_fXAcc);
	DDV_MinMaxFloat(pDX, m_fXAcc, -2.f, 2.f);
	DDX_Text(pDX, IDC_YACC, m_fYAcc);
	DDV_MinMaxFloat(pDX, m_fYAcc, -2.f, 2.f);
	DDX_Text(pDX, IDC_ZACC, m_fZAcc);
	DDV_MinMaxFloat(pDX, m_fZAcc, -2.f, 2.f);
	//}}AFX_DATA_MAP
}
```

```
BEGIN_MESSAGE_MAP(CGroundStationDlg, CDialog)
	//{{AFX_MSG_MAP(CGroundStationDlg)
	ON_WM_SYSCOMMAND()
	ON_WM_DESTROY()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_WM_CLOSE()
	ON_COMMAND(IDM_ABOUT, OnHelpAbout)
	ON_COMMAND(IDM_EXIT, OnExit)
	ON_BN_CLICKED(IDC_LOADMAPBUTTON, OnLoadmapbutton)
	ON_BN_CLICKED(IDC_UPBUTTON, OnUpbutton)
	ON_BN_CLICKED(IDC_CENTERBUTTON, OnCenterbutton)
	ON_BN_CLICKED(IDC_RIGHTBUTTON, OnRightbutton)
	ON_BN_CLICKED(IDC_LEFTBUTTON, OnLeftbutton)
	ON_BN_CLICKED(IDC_DOWNBUTTON, OnDownbutton)
	ON_BN_CLICKED(IDC_ZOOMINBUTTON, OnZoominbutton)
	ON_BN_CLICKED(IDC_ZOOMOUTBUTTON, OnZoomoutbutton)
	ON_BN_CLICKED(IDC_SIMBUTTON, OnSimbutton)
	ON_WM_TIMER()
	ON_BN_CLICKED(IDC_WAYPOINT, OnWaypoint)
	ON_BN_CLICKED(IDC_GENERATEPATH, OnGeneratepath)
	ON_WM_SETCURSOR()
	ON_BN_CLICKED(IDC_OPENCOM, OnOpencom)
	ON_BN_CLICKED(IDC_3DM, On3dm)
	ON_BN_DOUBLECLICKED(IDC_3DM, On3dm)
	ON_NOTIFY(NM_RELEASEDCAPTURE, IDC_ALTSELECT,
OnReleasedcaptureAltselect)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()


/////////////////////////////////////////////////////////////////////////
////////
// CGroundStationDlg message handlers
BOOL CGroundStationDlg::OnInitDialog()
{
	CDialog::OnInitDialog();

	// Add "About..." menu item to system menu.

	// IDM_ABOUTBOX must be in the system command range.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		CString strAboutMenu;
		strAboutMenu.LoadString(IDS_ABOUTBOX);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
strAboutMenu);
		}
	}

	// Set the icon for this dialog.  The framework does this
automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
```

```
	SetIcon(m_hIcon, FALSE);		// Set small icon

	// TODO: Add extra initialization here

	// Fill the edit boxes.
	m_strLatitude = "00deg 00min 00sec";
	m_strLongitude = "000deg 00min 00sec";
	m_strYaw = "00.0 degrees";
	m_strClimbRate = "0.0 m/s";

	m_dPi = 3.14159265358979323846264338327795;
	m_strAirSpeed = "21.0";
	m_strAltimeter = "135.0";
	m_strCompass = "000.0";
	m_dAirSpeed = atof (m_strAirSpeed);
	m_dAltimeter = atof (m_strAltimeter);
	m_dCompass = atof (m_strCompass);
	m_iGpsIndex = 0;
	// First write the static words for MAP INTERFACE and COMMAND
INTERFACE
	// and AUS UAV GROUND CONTROL

	m_strMapInterface = "Map \n Interface";
	m_strCommandInterface = "Command Interface";
	m_strAusUavTitle = "AUS UAV\nGround Station";

	// Initialize the zoom variables
	m_iXPos = 0;
	m_iYPos = 0;
	m_iWidth = 0;
	m_iHeight = 0;
	m_iZoomFactor = 5;

	// Initialize the UAV positon
	m_irPlaneX1 = 200;
	m_irPlaneX2 = 60;
	m_irPlaneY1 = 70;
	m_irPlaneY2 = 40;
	m_iTrans = 16;
	m_iCounter = 0;

	// Initialize the OnPaint flags
	m_bMap = FALSE;
	m_bPlaneSim = FALSE;
	m_iUpdateCount = 0;

	// Fill the Slider Control
	m_scAltSelect.SetRange (50, 150, true);
	m_iAltSelect = 50 + 150 - 135;
	m_scAltSelect.SetTicFreq (5);
	m_scAltSelect.Invalidate(TRUE);


	m_pcAltLevel.SetRange (50, 150);
	m_pcAltLevel.SetPos (135);
	m_iAltSelLevel = 135;

	// Initialize simulation values
	m_dSimValue = 0;
```

```
	// Call various functions
	FillFontList();		// Gets a list of fonts available to
the application from Windows
	InitChangeFontType();		// Change the type of the fonts
used in the static controls
	CreateSubWindows(); // Create the speedometer, compass, and
altimeter window

	// Update the display
	UpdateData(FALSE);

	// Update the initial controls
	GetDlgItem(IDC_ZOOMOUTBUTTON)->EnableWindow(FALSE);
	GetDlgItem(IDC_UPBUTTON)->EnableWindow(FALSE);
	GetDlgItem(IDC_DOWNBUTTON)->EnableWindow(FALSE);
	GetDlgItem(IDC_LEFTBUTTON)->EnableWindow(FALSE);
	GetDlgItem(IDC_RIGHTBUTTON)->EnableWindow(FALSE);
	GetDlgItem(IDC_CENTERBUTTON)->EnableWindow(FALSE);
	GetDlgItem(IDC_ZOOMINBUTTON)->EnableWindow(FALSE);
	GetDlgItem(IDC_ZOOMOUTBUTTON)->EnableWindow(FALSE);

	// Set the receive buffers to zeros
	int i;
	for (i = 0; i < 140; i++)
		m_cReceiveBuffer[i] = '0';

	m_cReceiveBuffer[139] = '\0';

	for (i = 0; i < 20; i++)
		m_fReceiveBuffer[i] = 0.0;

	return TRUE;  // return TRUE  unless you set the focus to a
control
}


void CGroundStationDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialog::OnSysCommand(nID, lParam);
	}
}


void CGroundStationDlg::OnDestroy()
{
	WinHelp(0L, HELP_QUIT);
	CDialog::OnDestroy();
}


// If you add a minimize button to your dialog, you will need the
code below
//  to draw the icon.  For MFC applications using the document/view
model,
```

```
//  this is automatically done for you by the framework.
void CGroundStationDlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this); // device context for painting

		SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(),
0);

		// Center icon in client rectangle
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;

		// Draw the icon
		dc.DrawIcon(x, y, m_hIcon);
	}
	else
	{
		CDialog::OnPaint();
	}
}


// The system calls this to obtain the cursor to display while the
user drags
//  the minimized window.
HCURSOR CGroundStationDlg::OnQueryDragIcon()
{
	return (HCURSOR) m_hIcon;
}


// Automation servers should not exit when a user closes the UI
//  if a controller still holds on to one of its objects.  These
//  message handlers make sure that if the proxy is still in use,
//  then the UI is hidden but the dialog remains around if it
//  is dismissed.
void CGroundStationDlg::OnClose()
{
	if (CanExit())
		CDialog::OnClose();
}


void CGroundStationDlg::OnOK()
{
	if (CanExit())
		CDialog::OnOK();
}


void CGroundStationDlg::OnCancel()
{
	if (CanExit())
		CDialog::OnCancel();
}
```

```
BOOL CGroundStationDlg::CanExit()
{
	// If the proxy object is still around, then the automation
	//  controller is still holding on to this application.  Leave
	//  the dialog around, but hide its UI.
	if (m_pAutoProxy != NULL)
	{
		ShowWindow(SW_HIDE);
		return FALSE;
	}

	return TRUE;
}


void CGroundStationDlg::OnHelpAbout()
{
	// TODO: Add your command handler code here
	// Instantiate the about dialog
	CAboutDlg dlgAbout;

	// Show the about window by calling the doModal function
	dlgAbout.DoModal();

}


void CGroundStationDlg::OnExit()
{
	// TODO: Add your command handler code here
	OnOK();

}


//////////////////////////////////////////////////////////////////////
//////////
// User Functions
int CALLBACK EnumFontFamProc (LPENUMLOGFONT lpelf, LPNEWTEXTMETRIC
lpntm, DWORD nFontType, long lParam)
{
	CGroundStationDlg* pWnd = (CGroundStationDlg*) lParam;

	//Add the font name to the list box
	pWnd->m_lbFontList.AddString(lpelf->elfLogFont.lfFaceName);
	return 1;
}


BOOL CGroundStationDlg::OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT
message)
{
	// TODO: Add your message handler code here and/or call default
	if (m_bCursor)
		return TRUE;
	else

	return CDialog::OnSetCursor(pWnd, nHitTest, message);
}
```

```
void CGroundStationDlg::FillFontList()
{
	// Initialize variables, to take care of the fact that multiple
font names doesnot repeat
	int iCount;	// the number of of fonts
	int iCurCount;	// The current font
	CString strCurFont;		// The current font name
	CString strPrevFont = "";	// The previois font name
	LOGFONT lf;

	// Initialize the log font structure
	lf.lfCharSet = DEFAULT_CHARSET;
	strcpy(lf.lfFaceName, "");

	// Clear the list box
	m_lbFontList.ResetContent();

	// Create a device context variable
	CClientDC dc(this);

	// Enumerate the font families
	::EnumFontFamiliesEx((HDC) dc, &lf, (FONTENUMPROC)
EnumFontFamProc, (LPARAM) this, 0);

	// Now remove duplicate names
	// Get the number of fonts in the list box
	iCount = m_lbFontList.GetCount();

	// Loop from the last entry in the list box to the firt,
searching for and deleting
	// duplicate entries
	for (iCurCount = iCount; iCurCount > 0; iCurCount--)
	{
		// Get the current font name
		m_lbFontList.GetText((iCurCount - 1), strCurFont);

		// Is it the same as the previous font name
		if (strCurFont == strPrevFont)
		{
			// If yes, then delete it
			m_lbFontList.DeleteString((iCurCount - 1));
		}

		// Set the current font as the previous font
		strPrevFont = strCurFont;
	}
}


void CGroundStationDlg::InitChangeFontType()
{
	// NOW SET THE FONTS OF EVERYTHING
	// Initialize a rectangel variable to work with fonts
	CRect rRect;	// The rectangle of the display area
	int iHeight;	// The height of the display area
	int iWidth;	// The width of the display area
	int CKITALIC = 1; // Should the font be italicized
	CString tempString; // temp dummy
```

```
	// DO THE MAP INTERFACE FIRST
	// Get the dimensions of the font sample display area
	tempString = m_strMapInterface;
	m_statMapInterface.GetWindowRect(&rRect);
	// Capture the area height and width
	iHeight = rRect.top - rRect.bottom;
	iWidth = rRect.left - rRect.right;
	// Make sure the hieght is positive
	if (iHeight < 0)
		iHeight = 0 - iHeight;
	if (iWidth < 0)
		iWidth = 0 - iWidth;

	// Release the current font
	m_fFontType.Detach();

	// Create the font to be used
	m_fFontType.CreateFont((iHeight - 2),
(iWidth/tempString.GetLength() - 2), 0, 0, FW_BOLD, CKITALIC,
		0, 0, DEFAULT_CHARSET, OUT_TT_PRECIS,
CLIP_CHARACTER_PRECIS,
		PROOF_QUALITY, VARIABLE_PITCH | FF_DECORATIVE, "Monotype
Corsiva");

	// Set the font for the sample display area
	m_statMapInterface.SetFont(&m_fFontType);


	// DO THE COMMAND INTERFACE NEXT
	// Get the dimensions of the font sample display area
	tempString = m_strCommandInterface;
	m_statCommandInterface.GetWindowRect(&rRect);
	// Capture the area height and width
	iHeight = rRect.top - rRect.bottom;
	iWidth = rRect.left - rRect.right;
	// Make sure the hieght is positive
	if (iHeight < 0)
		iHeight = 0 - iHeight;
	if (iWidth < 0)
		iWidth = 0 - iWidth;

	// Release the current font
	m_fFontType.Detach();

	// Create the font to be used
	m_fFontType.CreateFont((iHeight - 2),
(iWidth/tempString.GetLength() - 2), 0, 0, FW_SEMIBOLD, CKITALIC,
		0, 0, DEFAULT_CHARSET, OUT_TT_PRECIS,
CLIP_CHARACTER_PRECIS,
		PROOF_QUALITY, VARIABLE_PITCH | FF_DECORATIVE, "MONOTYPE
CORSIVA");

	// Set the font for the sample display area
	m_statCommandInterface.SetFont(&m_fFontType);


	// DO THE UAV TITLE LAST
	// Get the dimensions of the font sample display area
	tempString = m_strAusUavTitle;
	m_statAusUavTitle.GetWindowRect(&rRect);
```

```
	// Capture the area height and width
	iHeight = rRect.top - rRect.bottom;
	iWidth = rRect.left - rRect.right;
	// Make sure the hieght is positive
	if (iHeight < 0)
		iHeight = 0 - iHeight;
	if (iWidth < 0)
		iWidth = 0 - iWidth;

	// Release the current font
	m_fFontType.Detach();

	// Create the font to be used
	m_fFontType.CreateFont(((int)(iHeight/2) - 3),
((int)(1.5*iWidth/tempString.GetLength()) - 2), 0, 0, FW_SEMIBOLD,
CKITALIC,
		0, 0, DEFAULT_CHARSET, OUT_TT_PRECIS,
CLIP_CHARACTER_PRECIS,
		PROOF_QUALITY, VARIABLE_PITCH | FF_DECORATIVE, "MONOTYPE
CORSIVA");

	// Set the font for the sample display area
	m_statAusUavTitle.SetFont(&m_fFontType);
}


void CGroundStationDlg::CreateSubWindows()
{
	// Get the drawing area and calculate its size
	CRect rSpeedometer, rAltimeter, rCompass, rMap;

	// Initialize the drawing areas from different controls
	m_statAirSpeed.GetWindowRect(&rSpeedometer);
	m_statAltimeter.GetWindowRect(&rAltimeter);
	m_statCompass.GetWindowRect(&rCompass);
	m_statMap.GetWindowRect(&rMap);

	// Normalize the rectangles
	rSpeedometer.NormalizeRect();
	rAltimeter.NormalizeRect();
	rCompass.NormalizeRect();
	rMap.NormalizeRect();

	// Define the position of the rectangles
	rSpeedometer.top = rSpeedometer.top + 0;
	rSpeedometer.left = rSpeedometer.left + 0;
	rSpeedometer.bottom = rSpeedometer.bottom - 0;
	rSpeedometer.right = rSpeedometer.right - 0;

	rAltimeter.top = rAltimeter.top + 0;
	rAltimeter.left = rAltimeter.left + 0;
	rAltimeter.bottom = rAltimeter.bottom - 0;
	rAltimeter.right = rAltimeter.right - 0;

	rCompass.top = rCompass.top + 0;
	rCompass.left = rCompass.left + 0;
	rCompass.bottom = rCompass.bottom - 0;
	rCompass.right = rCompass.right - 0;

	rMap.top = rMap.top + 0;
	rMap.left = rMap.left + 0;
```

```
	rMap.bottom = rMap.bottom - 0;
	rMap.right = rMap.right - 0;

	// Now create the window objects and then position them on the
screen according
	// to the positions described earlier and then, make them
visible, and update
	// their display
	m_dlgSpeedometer.Create(IDD_SPEEDOMETER_DIALOG, this);
	m_dlgSpeedometer.ShowWindow(SW_SHOW);
	m_dlgSpeedometer.MoveWindow(rSpeedometer,TRUE);
	m_dlgSpeedometer.Invalidate(TRUE);
	m_dlgSpeedometer.m_bRepaint = TRUE;

	m_dlgAltimeter.Create(IDD_ALTIMETER_DIALOG, this);
	m_dlgAltimeter.ShowWindow(SW_SHOW);
	m_dlgAltimeter.MoveWindow(rAltimeter,TRUE);
	m_dlgAltimeter.Invalidate(TRUE);
	m_dlgAltimeter.m_bRepaint = TRUE;

	m_dlgCompass.Create(IDD_COMPASS_DIALOG, this);
	m_dlgCompass.ShowWindow(SW_SHOW);
	m_dlgCompass.MoveWindow(rCompass,TRUE);
	m_dlgCompass.Invalidate(TRUE);
	m_dlgCompass.m_bRepaint = TRUE;

	m_dlgMap.Create(IDD_MAP_DIALOG, this);
	m_dlgMap.ShowWindow(SW_SHOW);
	m_dlgMap.MoveWindow(rMap,TRUE);
	m_dlgMap.Invalidate(TRUE);
	m_dlgMap.m_iWayPointIndex = 0;
	m_dlgMap.m_iGpsIndex = 0;
	m_dlgMap.m_iSetPointIndex = 0;

	m_dlgUav.Create(IDD_MAP_DIALOG, this);
	m_dlgUav.MoveWindow(m_irPlaneX1,m_irPlaneY1,10,10,TRUE);
	m_dlgUav.ShowWindow(SW_HIDE);
	m_dlgUav.Invalidate(TRUE);
}


void CGroundStationDlg::OnLoadmapbutton()
{
	// TODO: Add your control notification handler code here
	static char BASED_CODE bcFilter[] = "Bitmap Files
(*.bmp)|*.bmp||";
	CFileDialog fdMapBmp(TRUE, ".bmp", NULL, OFN_HIDEREADONLY |
OFN_OVERWRITEPROMPT, bcFilter);
	if (fdMapBmp.DoModal() == IDOK)
	{
		m_strMapBmpFileName = fdMapBmp.GetPathName();
		HBITMAP hMapBmp = (HBITMAP)
::LoadImage(AfxGetInstanceHandle(), m_strMapBmpFileName,
IMAGE_BITMAP, 0, 0, LR_LOADFROMFILE | LR_CREATEDIBSECTION);

		// Do we have a valid handle for the loaded image?
		if (hMapBmp)
		{
			// Delete the current bitmap
			if (m_bmpMap.DeleteObject())
			{
```

```
                        // If there is a bitmap , detach it
                        m_bmpMap.Detach();
                  }

                  // Attach the currently loaded bitmap to the bitmap
object
                  m_bmpMap.Attach(hMapBmp);

                  // Set flag to enable repainting of the bitmap
                  m_bMap = TRUE;

                  // Initialize the bitmap coordinates for
positioning within the map area
                  // First Get the bitmap dimensions
                  BITMAP bmMap;
                  m_bmpMap.GetBitmap(&bmMap);

                  // Then get the map area dimensions
                  CRect rMap;
                  GetDlgItem(IDC_MAP)->GetClientRect(rMap);

                  // Now normalize the bitmap within the map area so
that it appears with a default zoom of one
                  m_iXPos = abs(rMap.Width() - bmMap.bmWidth)/2;
                  m_iYPos = abs(rMap.Height() - bmMap.bmHeight)/2;
                  m_iHeight = rMap.Height();
                  m_iWidth = rMap.Width();

                  // Now repaint the map area
                  m_dlgMap.Invalidate(TRUE);
                  GetDlgItem(IDC_ZOOMINBUTTON)->EnableWindow(TRUE);
                  GetDlgItem(IDC_CENTERBUTTON)->EnableWindow(TRUE);
                  GetDlgItem(IDC_LEFTBUTTON)->EnableWindow(TRUE);
                  GetDlgItem(IDC_RIGHTBUTTON)->EnableWindow(TRUE);
                  GetDlgItem(IDC_UPBUTTON)->EnableWindow(TRUE);
                  GetDlgItem(IDC_DOWNBUTTON)->EnableWindow(TRUE);

                  if (bmMap.bmWidth > (rMap.Width() + 10) &&
bmMap.bmHeight > (rMap.Height() + 10))
                        GetDlgItem(IDC_ZOOMOUTBUTTON)-
>EnableWindow(TRUE);
                  else
                        GetDlgItem(IDC_ZOOMOUTBUTTON)-
>EnableWindow(FALSE);

                  m_iZoomFactor = 5;
            }
      }
}


void CGroundStationDlg::OnCenterbutton()
{
      // TODO: Add your control notification handler code here

                  // Initialize the bitmap coordinates for
positioning within the map area
                  // First Get the bitmap dimensions
                  BITMAP bmMap;
                  m_bmpMap.GetBitmap(&bmMap);
```

```
                  // Then get the map area dimensions
                  CRect rMap;
                  GetDlgItem(IDC_MAP)->GetClientRect(rMap);

                  // Now normalize the bitmap within the map area so
that it appears with a default zoom of one
                  m_iXPos = abs(rMap.Width() - bmMap.bmWidth)/2;
                  m_iYPos = abs(rMap.Height() - bmMap.bmHeight)/2;
                  m_iHeight = rMap.Height();
                  m_iWidth = rMap.Width();

                  // Default the zoom factor
                  m_iZoomFactor = 5;

                  // Now repaint the map area
                  m_dlgMap.Invalidate(TRUE);
                  GetDlgItem(IDC_ZOOMINBUTTON)->EnableWindow(TRUE);
                  GetDlgItem(IDC_LEFTBUTTON)->EnableWindow(TRUE);
                  GetDlgItem(IDC_UPBUTTON)->EnableWindow(TRUE);
                  GetDlgItem(IDC_RIGHTBUTTON)->EnableWindow(TRUE);
                  GetDlgItem(IDC_DOWNBUTTON)->EnableWindow(TRUE);

                  if (bmMap.bmWidth > (rMap.Width() + 10) &&
bmMap.bmHeight > (rMap.Height() + 10))
                        GetDlgItem(IDC_ZOOMOUTBUTTON)-
>EnableWindow(TRUE);
                  else
                        GetDlgItem(IDC_ZOOMOUTBUTTON)-
>EnableWindow(FALSE);

}


void CGroundStationDlg::OnUpbutton()
{
      // TODO: Add your control notification handler code here
      if (m_iYPos >= m_iaZoomValues[m_iZoomFactor*4 + 3])
      {
            m_iYPos = m_iYPos - m_iaZoomValues[m_iZoomFactor*4 + 3];
            if (m_iYPos == 0)
                  GetDlgItem(IDC_UPBUTTON)->EnableWindow(FALSE);
      }
      else
      {
            m_iYPos = 0;
            GetDlgItem(IDC_UPBUTTON)->EnableWindow(FALSE);
      }
      GetDlgItem(IDC_DOWNBUTTON)->EnableWindow(TRUE);

      m_dlgMap.Invalidate(TRUE);
}


void CGroundStationDlg::OnDownbutton()
{
      // TODO: Add your control notification handler code here
      BITMAP bmMap;
      m_bmpMap.GetBitmap(&bmMap);

      if (m_iYPos <= (bmMap.bmHeight - m_iHeight -
m_iaZoomValues[m_iZoomFactor*4 + 3]))
```

```
	{
		m_iYPos = m_iYPos + m_iaZoomValues[m_iZoomFactor*4 + 3];
		if (m_iYPos == (bmMap.bmHeight - m_iHeight))
			GetDlgItem(IDC_DOWNBUTTON)->EnableWindow(FALSE);
	}
	else
	{
		m_iYPos = bmMap.bmHeight - m_iHeight;
		GetDlgItem(IDC_DOWNBUTTON)->EnableWindow(FALSE);
	}
	GetDlgItem(IDC_UPBUTTON)->EnableWindow(TRUE);

	m_dlgMap.Invalidate(TRUE);
}


void CGroundStationDlg::OnLeftbutton()
{
	// TODO: Add your control notification handler code here
	if (m_iXPos >= m_iaZoomValues[m_iZoomFactor*4 + 2])
	{
		m_iXPos = m_iXPos - m_iaZoomValues[m_iZoomFactor*4 + 2];
		if (m_iXPos == 0)
			GetDlgItem(IDC_LEFTBUTTON)->EnableWindow(FALSE);
	}
	else
	{
		m_iXPos = 0;
		GetDlgItem(IDC_LEFTBUTTON)->EnableWindow(FALSE);
	}
	GetDlgItem(IDC_RIGHTBUTTON)->EnableWindow(TRUE);

	m_dlgMap.Invalidate(TRUE);
}


void CGroundStationDlg::OnRightbutton()
{
	// TODO: Add your control notification handler code here
	BITMAP bmMap;
	m_bmpMap.GetBitmap(&bmMap);

	if (m_iXPos <= (bmMap.bmWidth - m_iWidth -
m_iaZoomValues[m_iZoomFactor*4 + 2]))
	{
		m_iXPos = m_iXPos + m_iaZoomValues[m_iZoomFactor*4 + 2];
		if (m_iXPos == (bmMap.bmWidth - m_iWidth))
			GetDlgItem(IDC_RIGHTBUTTON)->EnableWindow(FALSE);
	}
	else
	{
		m_iXPos = bmMap.bmWidth - m_iWidth;
		GetDlgItem(IDC_RIGHTBUTTON)->EnableWindow(FALSE);
	}
	GetDlgItem(IDC_LEFTBUTTON)->EnableWindow(TRUE);

	m_dlgMap.Invalidate(TRUE);
}
```

```
void CGroundStationDlg::OnZoominbutton()
{
	// TODO: Add your control notification handler code here
	BITMAP bmMap;
	m_bmpMap.GetBitmap(&bmMap);

	m_iZoomFactor++;
	if (m_iZoomFactor == 15)
	{
		GetDlgItem(IDC_ZOOMINBUTTON)->EnableWindow(FALSE);
	}
	GetDlgItem(IDC_ZOOMOUTBUTTON)->EnableWindow(TRUE);

	// The purpose is to center the object while zooming.
	// That is the object will be zoomed without the current center
of the object moiving
	// around the screen
	// The new height and widths of the bmp
	m_iWidth = m_iaZoomValues[m_iZoomFactor*4];
	m_iHeight = m_iaZoomValues[m_iZoomFactor*4 + 1];

	// The new screen coordinates
	m_iXPos = m_iXPos + (m_iaZoomValues[(m_iZoomFactor-1)*4] -
m_iWidth)/2;
	m_iYPos = m_iYPos + (m_iaZoomValues[(m_iZoomFactor-1)*4 + 1] -
m_iHeight)/2;

	// Enable the up and left button if valid
	if (m_iYPos >= 10)
		GetDlgItem(IDC_UPBUTTON)->EnableWindow(TRUE);
	if (m_iXPos >= 13)
		GetDlgItem(IDC_LEFTBUTTON)->EnableWindow(TRUE);
	if (m_iYPos <= (bmMap.bmHeight - m_iHeight))
		GetDlgItem(IDC_DOWNBUTTON)->EnableWindow(TRUE);
	if (m_iXPos <= (bmMap.bmWidth - m_iWidth))
		GetDlgItem(IDC_RIGHTBUTTON)->EnableWindow(TRUE);

	m_dlgMap.Invalidate(TRUE);
}


void CGroundStationDlg::OnZoomoutbutton()
{
	BITMAP bmMap;
	m_bmpMap.GetBitmap(&bmMap);

	m_iZoomFactor--;
	if (m_iZoomFactor == 0)
	{
		GetDlgItem(IDC_ZOOMOUTBUTTON)->EnableWindow(FALSE);
	}
	GetDlgItem(IDC_ZOOMINBUTTON)->EnableWindow(TRUE);

	// The purpose is to center the object while zooming.
	// That is the object will be zoomed without the current center
of the object moiving
	// around the screen
	// The new height and widths of the bmp
	m_iWidth = m_iaZoomValues[m_iZoomFactor*4];
	m_iHeight = m_iaZoomValues[m_iZoomFactor*4 + 1];
```

```
	if (m_iWidth >= bmMap.bmWidth || m_iHeight >= bmMap.bmHeight)
	{
		m_iWidth = bmMap.bmWidth;
		m_iHeight = bmMap.bmHeight;
		GetDlgItem(IDC_ZOOMOUTBUTTON)->EnableWindow(FALSE);
		GetDlgItem(IDC_UPBUTTON)->EnableWindow(FALSE);
		GetDlgItem(IDC_DOWNBUTTON)->EnableWindow(FALSE);
		GetDlgItem(IDC_LEFTBUTTON)->EnableWindow(FALSE);
		GetDlgItem(IDC_RIGHTBUTTON)->EnableWindow(FALSE);
	}


	// The new screen coordinates
	m_iXPos = m_iXPos - (m_iWidth -
m_iaZoomValues[(m_iZoomFactor+1)*4])/2;
	m_iYPos = m_iYPos - (m_iHeight -
m_iaZoomValues[(m_iZoomFactor+1)*4 + 1])/2;


	if (m_iXPos <= 0)
	{
		m_iXPos = 0;
		GetDlgItem(IDC_LEFTBUTTON)->EnableWindow(FALSE);
	}

	if (m_iYPos <= 0)
	{
		m_iYPos = 0;
		GetDlgItem(IDC_UPBUTTON)->EnableWindow(FALSE);
	}

	if ((m_iXPos + m_iWidth) >= bmMap.bmWidth)
	{
		m_iXPos = bmMap.bmWidth - m_iWidth;
		GetDlgItem(IDC_RIGHTBUTTON)->EnableWindow(FALSE);
	}

	if ((m_iYPos + m_iHeight) >= bmMap.bmHeight)
	{
		m_iYPos = bmMap.bmHeight - m_iHeight;
		GetDlgItem(IDC_DOWNBUTTON)->EnableWindow(FALSE);
	}

	m_dlgMap.Invalidate(TRUE);
}


void CGroundStationDlg::OnTimer(UINT nIDEvent)
{
	// TODO: Add your message handler code here and/or call default
	// Set the receive buffers to zeros

	int i;
	for (i = 0; i < 100; i++)
		m_cReceiveBuffer[i] = '0';

	m_cReceiveBuffer[139] = '\0';

	for (i = 0; i < 10; i++)
		m_fReceiveBuffer[i] = 0.0;
```

```
	// The Receive algorithm
	// Windwos handle and Data Communication structure
	HANDLE hCom;
	LPDCB lpDcb;

	// Initialize HAndle
	hCom=CreateFile("\\.\\COM1:",GENERIC_READ|GENERIC_WRITE,0,NULL,
OPEN_EXISTING,FILE_FLAG_OVERLAPPED,NULL);
	if (hCom==INVALID_HANDLE_VALUE)
	{
		MessageBox("Invalid Handle Value For Communication
Port.Program Fails!");
		return ;
	}

	// Initialize Data Communication Structure
	lpDcb=new(DCB);

	// Get Communication Port State
	if (!GetCommState(hCom,lpDcb))
	{
		MessageBox("Program Fails To Get Communication Port
State.");
		return ;
	}

	// Build Communication Port State
	CString strTemp=_T("baud=115200 parity=N data=8 stop=1");
	if (!BuildCommDCB(strTemp,lpDcb))
	{
		MessageBox("Program Fails To Build Communication Port
State.");
		return ;
	}

	// Set Communication Port State
	if (!SetCommState(hCom,lpDcb))
	{
		MessageBox("Program Fails To Set Communication Port
State.");
		return ;
	}

	// Communication Parameters
	DWORD dwRead, dwResult;
	DWORD dwTimeout=100;
	OVERLAPPED OL={0};
	BOOL blContinue=TRUE;

	// Allocate memory buffer for communication
	BYTE* bSendBuffer=(BYTE*) malloc(0x10000);
	if(bSendBuffer==NULL)
	{
		MessageBox(_T("Program Failed!Can Not Allocate Memory for
Tx"));
		return;
	}

	BYTE* bReceiveBuffer=(BYTE*) malloc(0x10000);
	if(bReceiveBuffer==NULL)
```

```
	{
		MessageBox(_T("Program Failed!Can Not Allocate Memory for
Rx"));
		return;
	}

	// Create an overlapped event
	OL.hEvent=CreateEvent(NULL, TRUE, FALSE, NULL);
	if (OL.hEvent ==NULL )
	{
		MessageBox(_T("Program Failed!Can Not Create Event"));
		return ;
	}

	// Create file to store data received for further evaluation
	CFile bf;
	if
(!bf.Open("C:\\Rx.txt",CFile::modeCreate|CFile::modeNoTruncate|CFile:
:modeWrite|CFile::typeBinary))
	{
		MessageBox(_T("Program Failed!Can Create File"));
		return;
	}
	bf.SeekToEnd();

	int track1, track8;
	track1 = 0;
	track8 = 0;

	// Start Communication
	// First send flag for the Avionics unit to start sending data
	// Create a memory buffer of data we wish to send, set dwRead
to
	// the number of data in the buffer

	track1 = 0;
	track8 = 0;
	blContinue = TRUE;

	// Now wait for Avionics unit to send data
	do
	{
		dwRead=ReadFile(hCom,bReceiveBuffer,1, &dwRead, &OL);  //
Start Receive Data
		if (dwRead==0)
		{
			// MEANS THAT NO CARRIAGE RETURN OR LINE FEED
OCCURRED
			if (GetLastError() != ERROR_IO_PENDING)
			{
				// NEVER COMES HERE UNLESS SOME GRAVE ERROR
HAS OCCURRED
				dwRead=0 ;
			}
			else
			{
				// THEREFORE GOES HERE INSTEAD. INFACT IF NO
CARRIAGE RETURN OCCURS THEN
				// FLOW ENTERS THIS CONDITION ALWAYS
				// Write is pending.
```

```
                        dwResult = WaitForSingleObject(OL.hEvent,
dwTimeout); // Wait for receive to finish

                        switch(dwResult)
                        {
                        case WAIT_OBJECT_0:
                              if (!GetOverlappedResult(hCom, &OL,
&dwRead, FALSE))
                              {
                                    dwRead=0 ;
                              }
                              break;
                        case WAIT_ABANDONED:
                              dwRead=0 ;
                              break;
                        case WAIT_TIMEOUT:
                              dwRead=0 ;
                              break;
                        default:
                              dwRead=0 ;
                              break;
                        }
                  }

                  if (dwRead==0)
                  {
                        // MEANS TIME OUT OR LAST CHARACTER RECEIVED.
COMES HERE JUST BEFORE EXITING
                        blContinue=FALSE;
                  }
                  else
                  {
                        // MEANS THAT NO TIME OUT HAS OCCURRED....
BLAH BLAH
                        bf.Write(bReceiveBuffer,dwRead);
                        m_cReceiveBuffer[track1] = bReceiveBuffer[0];
                  }
            }
            else
            {
                // MEANS THAT A CARRIAGE RETURN OR LINE FEED OCCURRED
                  bf.Write(bReceiveBuffer,dwRead);
                  m_cReceiveBuffer[track1] = bReceiveBuffer[0];
            }

            // GOES EVERY TIME HEHE
            // We dont want to receive more than 185 bytes at the
same time or more than two strings which ever is first
            track1++;
            if (track1 == 90)
            {
                  break;
            }

      }while(blContinue);


      Sleep(200);
      bf.Write("\n\n\n",3);
      bf.Close();
      CloseHandle(OL.hEvent);
```

```
	CloseHandle(hCom);
	free(bSendBuffer);
	free(bReceiveBuffer);


	// Check for frame length
	if (track1 < 90)
		return;

//	MessageBox ((const char*)m_cReceiveBuffer);
//	return;

	// Now process the information in the received buffer
	char temp[20];
//	unsigned char *addr_ftemp;
	double fRoll, fPitch, fLat, fLon, fYaw, fSpeed, fAlt;
	int j,k;
	i = 0;
	j = 0;
	k = 0;

	while (m_cReceiveBuffer[i] != '@')
		i++;

	while (m_cReceiveBuffer[i] == '@')
		i++;

	while (m_cReceiveBuffer[i] != '*')
	{
		while (m_cReceiveBuffer[i] != '|' && m_cReceiveBuffer[i]
!= '*')
		{
			temp[j] = m_cReceiveBuffer[i];
			i++;
			j++;
		}
		i++;
		temp [j] = '\0';
		j = 0;
		m_fReceiveBuffer[k] = atof (temp);
		k++;
	}

	if (k < 7)
		return;
//	MessageBox ((const char*)m_cReceiveBuffer);

	fYaw = (float)m_fReceiveBuffer[0];
	gcvt(fYaw, 5, temp);
	m_strCompass = CString(temp);


	fRoll = m_fReceiveBuffer [1];
	fPitch = m_fReceiveBuffer [2];


	fSpeed = (float)m_fReceiveBuffer[3];
	gcvt(fSpeed, 5, temp);
	m_strAirSpeed = CString(temp);

	fAlt = (float)m_fReceiveBuffer[4];
```

```
	gcvt(fAlt, 5, temp);
	m_strAltimeter = CString(temp);

	fLat = m_fReceiveBuffer[5];
	fLon = m_fReceiveBuffer[6];
	if (fLat > 32000)
		fLat = fLat - 65536;
	if (fLon > 32000)
		fLon = fLon - 65536;

	m_fXRate = (float)fLat;
	m_fYRate = (float)fLon;


		m_aHorizon.SetAHRoll(fRoll);
		m_aHorizon.SetAHPitch(fPitch);
		m_dAHRoll = -fRoll;
		m_dAHPitch = fPitch;

		m_irPlaneX1 = (int)(fLat/3.3333) + m_iTrans;
		m_irPlaneY1 = (int)(fLon/3.3333) + (int)m_dSimValue;
		m_dlgUav.MoveWindow(m_irPlaneX1,m_irPlaneY1,10,10,TRUE);


		UpdateData(FALSE);
		m_dlgSpeedometer.Invalidate(TRUE);
		m_dlgAltimeter.Invalidate(TRUE);
		m_dlgCompass.Invalidate(TRUE);
		m_aHorizon.Invalidate(TRUE);

		CDialog::OnTimer(nIDEvent);
}


void CGroundStationDlg::OnSimbutton()
{
	// TODO: Add your control notification handler code here
	if (m_bPlaneSim == FALSE)
	{
		m_iGpsIndex = 0;

		CRect rRect;
		m_statMap.GetWindowRect(rRect);
		m_iTrans = rRect.left;
		m_dSimValue = rRect.top;

		m_bPlaneSim = TRUE;
		m_dlgUav.ShowWindow(SW_SHOW);

		// Start the first timer
		// 500 ms is minimum value. (55ms for actual application,
but in this case 500 ms)
		SetTimer(IDT_CLOCK_TIMER, 150, NULL);
	}
	else
	{
		m_bPlaneSim = FALSE;
		m_dlgUav.ShowWindow(SW_HIDE);
		KillTimer(IDT_CLOCK_TIMER);
	}
}
```

```
void CGroundStationDlg::OnWaypoint()
{
	// TODO: Add your control notification handler code here
	HCURSOR lhCursor; // The handle to the cursor to be displayed.
	lhCursor = AfxGetApp()->LoadStandardCursor(IDC_CROSS);
	SetCursor(lhCursor);
	m_bCursor = TRUE;
	m_dlgMap.EnableWindow(TRUE);
	m_dlgMap.m_iWayPointIndex = 0;
	m_dlgMap.m_iGpsIndex = 0;
	m_strCompass = "000.0";

	m_iAltSelect = 50 + 150 - 135;
	m_scAltSelect.Invalidate(TRUE);


	m_pcAltLevel.SetPos (135);
	m_iAltSelLevel = 135;
	UpdateData(FALSE);


}


void CGroundStationDlg::OnGeneratepath()
{
	// TODO: Add your control notification handler code here
	m_bGenPath = TRUE;	// Flag indicating that path has been
generated for simulation
	m_bCursor = FALSE;	// Flag indicating that user cannot
select waypoints anymore
	m_dlgMap.EnableWindow(FALSE); // Make the map disabled so that
the user cannot interact with
						// the map once the
selection of waypoints is over
	m_dlgMap.Invalidate(TRUE);	// Update the map area

	int i, j, k;
	// These variables indicate the three waypoints involved in a
two line path
	// and the two ciruclar center of turn
	double dX1, dX2, dY1, dY2, dX1c, dX2c, dY1c, dY2c, dX1Diff,
dY1Diff, dXc, dYc, dDist;
	// These variables deals with the angles involved
	double dCurrHeading, dPrevHeading, dDeltaAngle, dAngle1,
dAngle2, dAltReq, dAltCurr;
	// These variables deals with the distances involved
	double dCurrHyp, dPrevHyp, dTempHyp, dHyp1, dHyp2, dHyp,
dRadius, dAltTemp, dTimeConst;
	// These variables deals with the direction vectors involved
	double dQiCurr, dQjCurr, dQiPrev, dQjPrev, dQiRc1, dQjRc1,
dQiRc2, dQjRc2, dCrossProduct;
	double UpdateRate = 20.0;	// UpdateRate

	dPrevHyp = 0.0;
	dPrevHeading = atof(m_strCompass);
	dQiPrev = 1;
	dQjPrev = 0;
	dRadius = 75.0;
```

```
	dTimeConst = 350.0;
	dAltCurr = 135.0;

	j = 0;
	k = 0;

	for (i = 0; i < (m_dlgMap.m_iWayPointIndex-1); i++)
	{
		dX1 = (double)m_dlgMap.m_iWayPoints[0][i];
		dY1 = (double)m_dlgMap.m_iWayPoints[1][i];
		dX2 = (double)m_dlgMap.m_iWayPoints[0][i+1];
		dY2 = (double)m_dlgMap.m_iWayPoints[1][i+1];
		dAltReq = (double)m_dlgMap.m_iWayPoints[2][i+1];

		// Get the current direction vector
		dX1Diff = dX2 - dX1; // Get the x distance between the
current two waypoints
		dY1Diff = dY2 - dY1; // Get the y distance between the
current two waypoints
		dCurrHyp = sqrt(dX1Diff*dX1Diff + dY1Diff*dY1Diff); //
Get the length of the line
		dQiCurr = dX1Diff/dCurrHyp; // Get the i direction vector
of the current line
		dQjCurr = dY1Diff/dCurrHyp; // Get the j direction vector
of the current line

		dCurrHeading = 180.0*atan(dY1Diff/dX1Diff)/m_dPi;
		if (dX2<dX1)
			dCurrHeading = 180.0 + dCurrHeading;
		if (dCurrHeading < 0)
			dCurrHeading = 360.0 + dCurrHeading;


		// fDeltaAngle represents the angle that will space the
points on the
		// circular path by m_strAirSpeed meters. [RATE OF TURN]
		dDeltaAngle =
atof(m_strAirSpeed)*360.0/(2*m_dPi*dRadius);
		dDeltaAngle = dDeltaAngle/UpdateRate; // 20 Hz update
		if (dDeltaAngle < 0.5)
			dDeltaAngle = 0.5;

		if (abs((int)dPrevHeading - (int)dCurrHeading) >=
dDeltaAngle
			&& abs((int)dPrevHeading - (360 +
(int)dCurrHeading)) >= dDeltaAngle
			   && abs((360 + (int)dPrevHeading) -
(int)dCurrHeading) >= dDeltaAngle)
		{

			// Over here the factors that are decided are by
how much will be the
			// rate of turn denoted by delta angle.  This
depends on the speed, and
			// the length of the circumference.  The faster the
speed, and the smaller
			// the radius, the more the rate of turn.
			// The second factor that is decided is if the turn
will be clockwise or
			// counterclockwise.  This factor depends on the
current heading and the
```

```
                  // deviation of the target direaction from the
heading.

                  // fDeltaAngle represents the angle that will space
the points on the
                  // circular path by m_strAirSpeed meters. [RATE OF
TURN]
                  dDeltaAngle =
atof(m_strAirSpeed)*360.0/(2*m_dPi*dRadius);
                  dDeltaAngle = dDeltaAngle/UpdateRate; // 20 Hz
update
                  if (dDeltaAngle < 0.5)
                        dDeltaAngle = 0.5;


                  // i.e delta angle give one degree turns

                  // First find the two circular arcs which the UAV
can take given its current
                  // direction.  The UAV must enter the circle in a
tangent

                  // Find the direction vectors for the center of the
circular arc which is
                  // perpendicular to the current direction vector
                  // For current direction vector of ai + bj, the
perpendicular will be bi - aj
                  dQiRc1 = dQjPrev;
                  dQjRc1 = -dQiPrev;
                  // For current direction vector of ai + bj, the
perpendicular will also be -bi + aj
                  dQiRc2 = -dQjPrev;
                  dQjRc2 = dQiPrev;

                  // For these direction vectors find the centers of
the circular arcs from waypoint
                  // For circle 1
                  dX1c = dX1 + dQiRc1*dRadius;
                  dY1c = dY1 + dQjRc1*dRadius;
                  // For circle 2
                  dX2c = dX1 + dQiRc2*dRadius;
                  dY2c = dY1 + dQjRc2*dRadius;

                  // Second find which of the two circular arcs will
provide mimimum distance travel
                  dHyp1 = sqrt((dX1c - dX2)*(dX1c - dX2) + (dY1c -
dY2)*(dY1c - dY2));
                  dHyp2 = sqrt((dX2c - dX2)*(dX2c - dX2) + (dY2c -
dY2)*(dY2c - dY2));

                  // Now make the previous heading into the current
(instantaneous) heading going
                  // into a circle
                  dCurrHeading = dPrevHeading;

                  // dHyp which is smaller will give the minimum
distance travel. But the smaller
                  // dHyp should not be less than the minimum radius.
Therefore select the center
                  // of the circular arc as the smaller dHyp as long
as it is greater of equal to R.
```

```
                  // Once the appropriate dHyp is chosen, determine
the direction of turn (CW or CCW)
                  // given by +/- deltaAngle.  Turn is CW if the
cross product of velocity direction
                  // vector and radius direction vector is negative,
else turn is CCW
                  if (dHyp1 < dHyp2)
                  {
                        if (dHyp1 >= dRadius)
                        {
                              // Enter the circular arc with center
dX1c, and dY1c
                              // Find the cross products of the
direction vectors
                              dCrossProduct = (dQiPrev * dQjRc1) -
(dQiRc1 * dQjPrev);
                              if (dCrossProduct < 0)
                              {
                                    dDeltaAngle = -dDeltaAngle;
                              }
                              dXc = dX1c;
                              dYc = dY1c;
                        }
                        else
                        {
                              // Enter the circular arc with center
dX2c, and dY2c,
                              dCrossProduct = (dQiPrev * dQjRc2) -
(dQiRc2 * dQjPrev);
                              if (dCrossProduct < 0)
                              {
                                    dDeltaAngle = -dDeltaAngle;
                              }
                              dXc = dX2c;
                              dYc = dY2c;

                        }
                  } // if check to see if hyp 1 < hyp 2


                  // If dHyp1 = dHyp2, then the next waypoint lies in
the tangent to the circular path
                  // Therefore it is not necessary to check if the
dHyp is less than the radius
                  // This is because since dHyp is in the tangent, it
will be always outside the circle
                  // no matter how small it is.  We will by default
go the top way
                  if (dHyp1 == dHyp2)
                  {
                        // Enter the circular arc with center dX1c,
and dY1c
                        // Find the cross products of the direction
vectors
                        dCrossProduct = (dQiPrev * dQjRc1) - (dQiRc1
* dQjPrev);
                        if (dCrossProduct < 0)
                        {
                              dDeltaAngle = -dDeltaAngle;
                        }
                        dXc = dX1c;
```

```
                        dYc = dY1c;
                  } // if check to see if hyp 1 = hyp 2

                  if (dHyp2 < dHyp1)
                  {
                        if (dHyp2 >= dRadius)
                        {
                              // Enter the circular arc with center
dX2c, and dY2c,
                              dCrossProduct = (dQiPrev * dQjRc2) -
(dQiRc2 * dQjPrev);
                              if (dCrossProduct < 0)
                              {
                                    dDeltaAngle = -dDeltaAngle;
                              }
                              dXc = dX2c;
                              dYc = dY2c;
                        }
                        else
                        {
                              // Enter the circular arc with center
dX1c, and dY1c,
                              dCrossProduct = (dQiPrev * dQjRc1) -
(dQiRc1 * dQjPrev);
                              if (dCrossProduct < 0)
                              {
                                    dDeltaAngle = -dDeltaAngle;
                              }
                              dXc = dX1c;
                              dYc = dY1c;
                        }
                  } // if check to see if hyp 2 < hyp 1

                  // Now that we determined the circular arc we are
going to fly over, and the
                  // direction of turn, determine the circle
parameteres such as radius, start of
                  // circle, end of circle etc.
                  dAngle1 = 180.0*atan((dY1 - dYc)/(dX1 -
dXc))/m_dPi;
                  if (dX1<dXc)
                        dAngle1 = 180.0 + dAngle1;
                  if (dAngle1 < 0)
                        dAngle1 = 360.0 + dAngle1;

                  dAngle2 = 180.0*atan((dY2 - dY1)/(dX2 -
dX1))/m_dPi;
                  if (dX2<dX1)
                        dAngle2 = 180.0 + dAngle2;
                  if (dAngle2 < 0)
                        dAngle2 = 360.0 + dAngle2;

                  // do a loop as long as the UAV is in the circle
                  while (1)
                  {
                        m_dlgMap.m_iGpsCoor[0][j] = (int)dX1;     //
X position
                        m_dlgMap.m_iGpsCoor[1][j] = (int)dY1;     //
Y position
                        m_dlgMap.m_iGpsCoor[2][j] =
(int)dCurrHeading;      // Heading
```

```
                        m_dlgMap.m_iGpsCoor[3][j] =
atoi(m_strAirSpeed);    // Airspeed
                        m_dlgMap.m_iGpsCoor[4][j] = (int)dAltCurr;
      // Altitude
                        j++;
                        m_dlgMap.m_iGpsIndex = j;

                        // Check if the UAV is facing the direction
of the waypoint
                        // If so break from loop and fly straight
                        if (abs((int)dCurrHeading - (int)dAngle2) <=
abs((int)dDeltaAngle)
                              || abs((int)dCurrHeading - (360 +
(int)dAngle2)) <= abs((int)dDeltaAngle)
                                    || abs((360 + (int)dCurrHeading)
- (int)dAngle2) <= abs((int)dDeltaAngle))
                        {
                              break;
                        }

                        // Update the angle left to turn
                        dAngle1 = dAngle1 + dDeltaAngle;
                        if (dAngle1 >= 360.0)
                              dAngle1 = dAngle1 - 360.0;

                        if (dAngle1 < 0)
                              dAngle1 = dAngle1 + 360.0;

                        // Update the instantaneous position
                        dX1 = dXc + cos(dAngle1*m_dPi/180.0)*dRadius;
                        dY1 = dYc + sin(dAngle1*m_dPi/180.0)*dRadius;

                        // Update the instantaneous heading
                        dCurrHeading = dCurrHeading + dDeltaAngle;
                        if (dCurrHeading >= 360.0)
                              dCurrHeading = dCurrHeading - 360.0;

                        if (dCurrHeading < 0)
                              dCurrHeading = dCurrHeading + 360.0;

                        // Update the heading to the waypoint from
instantaneous position
                        dAngle2 = 180.0*atan((dY2 - dY1)/(dX2 -
dX1))/m_dPi;
                        if (dX2<dX1)
                              dAngle2 = 180.0 + dAngle2;
                        if (dAngle2 < 0)
                              dAngle2 = 360.0 + dAngle2;

                        // Update the current direction vector
                        dQiCurr = cos(dCurrHeading*m_dPi/180.0);
                        dQjCurr = sin(dCurrHeading*m_dPi/180.0);

                        // Update the current hyp
                        dX1Diff = dX2 - dX1;
                        dY1Diff = dY2 - dY1;
                        dCurrHyp = sqrt(dX1Diff*dX1Diff +
dY1Diff*dY1Diff);

                  } // End of while loop of UAV flying in a circle
```

```
                  // Now if the UAV is still far from waypoint fly
straigth
                  dHyp = dCurrHyp;
                  dPrevHyp = dCurrHyp;
                  dPrevHeading = dCurrHeading;
                  dQiPrev = dQiCurr;
                  dQjPrev = dQjCurr;

                  dTempHyp = dHyp + 1;
                  // Create descrete points between the start and the
end of the line
//                while ( dHyp > atof(m_strAirSpeed))

                  dDist = 0;
                  while ( dTempHyp > dHyp )
                  {
                        dTempHyp = dHyp;
                        if (dAltCurr > dAltReq)
                              dAltTemp = dAltReq + (dAltCurr-
dAltReq)*exp(-dDist/dTimeConst);
                        if (dAltCurr < dAltReq)
                              dAltTemp = dAltReq - (dAltReq-
dAltCurr)*exp(-dDist/dTimeConst);
                        if (dAltCurr == dAltReq)
                              dAltTemp = dAltCurr;
                        dX1 = dX1 +
atof(m_strAirSpeed)*cos(dCurrHeading*m_dPi/180.0)/UpdateRate;     //
20 Hz Update
                        dY1 = dY1 +
atof(m_strAirSpeed)*sin(dCurrHeading*m_dPi/180.0)/UpdateRate;     //
20 Hz Update
                        m_dlgMap.m_iGpsCoor[0][j] = (int)dX1;
                        m_dlgMap.m_iGpsCoor[1][j] = (int)dY1;
                        m_dlgMap.m_iGpsCoor[2][j] =
(int)dCurrHeading;
                        m_dlgMap.m_iGpsCoor[3][j] =
atoi(m_strAirSpeed);
                        m_dlgMap.m_iGpsCoor[4][j] = (int)dAltTemp;
                        dX1Diff = dX2 - dX1;
                        dY1Diff = dY2 - dY1;
                        dHyp = sqrt(dX1Diff*dX1Diff +
dY1Diff*dY1Diff);
                        dDist = dDist +
1.5*atof(m_strAirSpeed)/UpdateRate;
                        j++;
                        m_dlgMap.m_iGpsIndex = j;
                  } // end of while loop to plot dots for descrete
paths between waypoints

                  // Enter the final heading and GPS point into the
database
                  m_dlgMap.m_iGpsCoor[0][j] = (int)dX2;
                  m_dlgMap.m_iGpsCoor[1][j] = (int)dY2;
                  m_dlgMap.m_iGpsCoor[2][j] = (int)dCurrHeading;
                  m_dlgMap.m_iGpsCoor[3][j] = atoi(m_strAirSpeed);
                  m_dlgMap.m_iGpsCoor[4][j] = (int)dAltReq;
                  dAltCurr = dAltReq;
                  j++;
                  m_dlgMap.m_iGpsIndex = j;
```

```
            } // end of if loop to check for straight line
            else
            {
                  // Enter the initial heading and GPS coordinates
into database
                  m_dlgMap.m_iGpsCoor[0][j] = (int)dX1;     // X
position
                  m_dlgMap.m_iGpsCoor[1][j] = (int)dY1;     // Y
position
                  m_dlgMap.m_iGpsCoor[2][j] = (int)dPrevHeading;  //
Heading
                  m_dlgMap.m_iGpsCoor[3][j] = atoi(m_strAirSpeed);
      // Airspeed
                  m_dlgMap.m_iGpsCoor[4][j] = (int)dAltCurr;      //
Altitude
                  j++;
                  m_dlgMap.m_iGpsIndex = j;

                  dHyp = dCurrHyp;
                  dPrevHyp = dCurrHyp;
                  dPrevHeading = dCurrHeading;
                  dQiPrev = dQiCurr;
                  dQjPrev = dQjCurr;

                  dTempHyp = dHyp + 1;
                  // Create descrete points between the start and the
end of the line
//                  while ( dHyp > atof(m_strAirSpeed))

                  dDist = 0;
                  while ( dTempHyp > dHyp )
                  {
                        dTempHyp = dHyp;
                        if (dAltCurr > dAltReq)
                              dAltTemp = dAltReq + (dAltCurr-
dAltReq)*exp(-dDist/dTimeConst);
                        if (dAltCurr < dAltReq)
                              dAltTemp = dAltReq - (dAltReq-
dAltCurr)*exp(-dDist/dTimeConst);
                        if (dAltCurr == dAltReq)
                              dAltTemp = dAltCurr;
                        dX1 = dX1 +
atof(m_strAirSpeed)*cos(dCurrHeading*m_dPi/180.0)/UpdateRate;     //
20 Hz Update
                        dY1 = dY1 +
atof(m_strAirSpeed)*sin(dCurrHeading*m_dPi/180.0)/UpdateRate;     //
20 Hz Update
                        m_dlgMap.m_iGpsCoor[0][j] = (int)dX1;
                        m_dlgMap.m_iGpsCoor[1][j] = (int)dY1;
                        m_dlgMap.m_iGpsCoor[2][j] =
(int)dCurrHeading;
                        m_dlgMap.m_iGpsCoor[3][j] =
atoi(m_strAirSpeed);
                        m_dlgMap.m_iGpsCoor[4][j] = (int)dAltTemp;
                        dX1Diff = dX2 - dX1;
                        dY1Diff = dY2 - dY1;
                        dHyp = sqrt(dX1Diff*dX1Diff +
dY1Diff*dY1Diff);
                        dDist = dDist +
1.5*atof(m_strAirSpeed)/UpdateRate;
                        j++;
```

```
                        m_dlgMap.m_iGpsIndex = j;
                  } // end of while loop to plot dots for descrete
paths between waypoints

                  // Enter the final heading and GPS point into the
database
                  m_dlgMap.m_iGpsCoor[0][j] = (int)dX2;
                  m_dlgMap.m_iGpsCoor[1][j] = (int)dY2;
                  m_dlgMap.m_iGpsCoor[2][j] = (int)dCurrHeading;
                  m_dlgMap.m_iGpsCoor[3][j] = atoi(m_strAirSpeed);
                  m_dlgMap.m_iGpsCoor[4][j] = (int)dAltReq; //
Altitude
                  dAltCurr = dAltReq;
                  j++;
                  m_dlgMap.m_iGpsIndex = j;
            }  // end of else check for straight line
      } // end of for check for further waypoints


      // Now store the waypoints list and waypoint detailed list into
file
      FILE *testfile;
      char *testfilename = "c:\\testfile.txt";
      testfile = fopen( testfilename, "w");
      char tempfilebuffer[80], tempstring[20];
      i = 0;
      while (i < m_dlgMap.m_iWayPointIndex)
      {
            itoa(m_dlgMap.m_iWayPoints[0][i],tempstring, 10);
            strcpy (tempfilebuffer, tempstring);
            strcat (tempfilebuffer, "\t");
            itoa(m_dlgMap.m_iWayPoints[1][i],tempstring, 10);
            strcat (tempfilebuffer, tempstring);
            strcat (tempfilebuffer, "\t");
            itoa(m_dlgMap.m_iWayPoints[2][i],tempstring, 10);
            strcat (tempfilebuffer, tempstring);
            strcat (tempfilebuffer, "\n\r");
            fputs (tempfilebuffer, testfile);
            i++;
      }
      strcat (tempfilebuffer, "\n\r");
      fputs (tempfilebuffer, testfile);
      i = 0;
      while (i < m_dlgMap.m_iGpsIndex)
      {
            itoa(m_dlgMap.m_iGpsCoor[0][i], tempstring, 10);
            strcpy (tempfilebuffer, tempstring);
            strcat (tempfilebuffer, "\t");
            itoa(m_dlgMap.m_iGpsCoor[1][i], tempstring, 10);
            strcat (tempfilebuffer, tempstring);
            strcat (tempfilebuffer, "\t");
            itoa(m_dlgMap.m_iGpsCoor[2][i], tempstring, 10);
            strcat (tempfilebuffer, tempstring);
            strcat (tempfilebuffer, "\t");
            itoa(m_dlgMap.m_iGpsCoor[3][i], tempstring, 10);
            strcat (tempfilebuffer, tempstring);
            strcat (tempfilebuffer, "\t");
            itoa(m_dlgMap.m_iGpsCoor[4][i], tempstring, 10);
            strcat (tempfilebuffer, tempstring);
            strcat (tempfilebuffer, "\n\r");
            fputs (tempfilebuffer, testfile);
```

```
            i++;
      }
      strcat (tempfilebuffer, "\n\r");
      fputs (tempfilebuffer, testfile);
      fclose (testfile);

}

void CGroundStationDlg::OnOpencom()
{
      // TODO: Add your control notification handler code here

      // Set the receive buffers to zeros
      int i;
      for (i = 0; i < 140; i++)
            m_cReceiveBuffer[i] = '0';

      m_cReceiveBuffer[139] = '\0';

      for (i = 0; i < 20; i++)
            m_fReceiveBuffer[i] = 0.0;

      // Windwos handle and Data Communication structure
      HANDLE hCom;
      LPDCB lpDcb;

      // Communication Parameters
      DWORD dwRead, dwWritten, dwResult;
      DWORD dwTimeout=350;
      OVERLAPPED OL1={0};
      BOOL blContinue=TRUE;

      // Allocate memory buffer for communication
      BYTE* bSendBuffer=(BYTE*) malloc(0xFFFF);
      if(bSendBuffer==NULL)
      {
            MessageBox(_T("Program Failed!Can Not Allocate Memory for
Tx"));
            return;
      }

      BYTE* bReceiveBuffer=(BYTE*) malloc(0xFFFF);
      if(bReceiveBuffer==NULL)
      {
            MessageBox(_T("Program Failed!Can Not Allocate Memory for
Rx"));
            return;
      }

      int track1, track8;
      track1 = 0;
      track8 = 0;

      // Initialize HAndle
      hCom=CreateFile("\\.\\COM1:",GENERIC_READ|GENERIC_WRITE,0,NULL,
OPEN_EXISTING,FILE_FLAG_OVERLAPPED,NULL);
      if (hCom==INVALID_HANDLE_VALUE)
      {
            MessageBox("Invalid Handle Value For Communication
Port.Program Fails!");
            return ;
```

```
	}

	// Initialize Data Communication Structure
	lpDcb=new(DCB);

	// Get Communication Port State
	if (!GetCommState(hCom,lpDcb))
	{
		MessageBox("Program Fails To Get Communication Port
State.");
		return ;
	}

	// Build Communication Port State
	CString strTemp=_T("baud=115200 parity=N data=8 stop=1");
	if (!BuildCommDCB(strTemp,lpDcb))
	{
		MessageBox("Program Fails To Build Communication Port
State.");
		return ;
	}

	// Set Communication Port State
	if (!SetCommState(hCom,lpDcb))
	{
		MessageBox("Program Fails To Set Communication Port
State.");
		return ;
	}

	// Create an overlapped event
	OL1.hEvent=CreateEvent(NULL, TRUE, FALSE, NULL);
	if (OL1.hEvent ==NULL )
	{
		MessageBox(_T("Program Failed!Can Not Create Event1"));
		return ;
	}

	// Start Communication
	// First send flag for the Avionics unit to start sending data
	// Create a memory buffer of data we wish to send, set dwRead
to
	// the number of data in the buffer

	// Store buffer with waypoints
	int j,k;
    char temp[6];

	k = 0;

	for (i = 0; i < 5; i++)
	{
		itoa(m_dlgMap.m_iWayPoints[0][i], temp, 10);
		for (j = 0; j < 6; j++)
		{
			if (temp[j] == '\0')
				break;
			bSendBuffer[k] = temp[j];
			k++;
		}
		bSendBuffer[k] = ' ';
```

```
            k++;

            itoa(m_dlgMap.m_iWayPoints[1][i], temp, 10);
            for (j = 0; j < 6; j++)
            {
                  if (temp[j] == '\0')
                        break;
                  bSendBuffer[k] = temp[j];
                  k++;
            }
            bSendBuffer[k] = ' ';
            k++;

            itoa(m_dlgMap.m_iWayPoints[2][i], temp, 10);
            for (j = 0; j < 6; j++)
            {
                  if (temp[j] == '\0')
                        break;
                  bSendBuffer[k] = temp[j];
                  k++;
            }
            bSendBuffer[k] = ' ';
            k++;

      }

      bSendBuffer[k] = '\0';
      dwRead = k;

      // Start sending files
      WriteFile(hCom,bSendBuffer,dwRead,&dwWritten,&OL1);
      if (GetLastError() != ERROR_IO_PENDING)
      {
            dwWritten=0 ;
      }
      else
      {
            // Write is pending.
            dwResult = WaitForSingleObject(OL1.hEvent, INFINITE);
            switch(dwResult)
            {
                  case WAIT_OBJECT_0:
                        if (!GetOverlappedResult(hCom, &OL1,
&dwWritten, FALSE))
                              dwWritten=0 ;
                        break;
                  case WAIT_ABANDONED:
                        dwWritten=0 ;
                        break;
                  case WAIT_TIMEOUT:
                        dwWritten=0 ;
                        break;
                  default:
                        dwWritten=0 ;
                        break;
            }
      }

      track1 = 0;
      track8 = 0;
```

```
      // Now wait for Avionics unit to send data
      do
      {
            dwRead=ReadFile(hCom,bReceiveBuffer,1, &dwRead, &OL1);
// Start Receive Data
            if (dwRead==0)
            {
                  // MEANS THAT NO CARRIAGE RETURN OR LINE FEED
OCCURRED
                  if (GetLastError() != ERROR_IO_PENDING)
                        dwRead=0 ;  // NEVER COMES HERE UNLESS SOME
GRAVE ERROR HAS OCCURRED
                  else
                  {
                        // THEREFORE GOES HERE INSTEAD. INFACT IF NO
CARRIAGE RETURN OCCURS THEN
                        // FLOW ENTERS THIS CONDITION ALWAYS
                        // Write is pending.
                        dwResult = WaitForSingleObject(OL1.hEvent,
dwTimeout); // Wait for receive to finish

                        switch(dwResult)
                        {
                              case WAIT_OBJECT_0:
                                    if (!GetOverlappedResult(hCom,
&OL1, &dwRead, FALSE))
                                          dwRead=0 ;
                                    break;
                              case WAIT_ABANDONED:
                                    dwRead=0 ;
                                    break;
                              case WAIT_TIMEOUT:
                                    dwRead=0 ;
                                    break;
                              default:
                                    dwRead=0 ;
                                    break;
                        }
                  }

                  if (dwRead==0)
                        blContinue=FALSE; // MEANS TIME OUT OR LAST
CHARACTER RECEIVED. COMES HERE JUST BEFORE EXITING
                  else
                  {
                        // MEANS THAT NO TIME OUT HAS OCCURRED....
BLAH BLAH
                        m_cReceiveBuffer[track1] = bReceiveBuffer[0];
                        if (bReceiveBuffer[0] == '@')
                              track8++;
                  }
            }
            else
            {
                // MEANS THAT A CARRIAGE RETURN OR LINE FEED OCCURRED
                  m_cReceiveBuffer[track1] = bReceiveBuffer[0];
                  if (bReceiveBuffer[0] == '@')
                        track8++;
            }

            // GOES EVERY TIME HEHE
```

```
            // We dont want to receive more than 185 bytes at the
same time or more than two strings which ever is first
            track1++;
            if (track1 == 140 || track8 == 3)
            {
                  break;
            }

      }while(blContinue);

      Sleep(700);
      CloseHandle(OL1.hEvent);
      CloseHandle(hCom);
      free(bSendBuffer);
      free(bReceiveBuffer);

      char cReceiveBuffer[100];
      for (i = 0; i <100; i++)
            cReceiveBuffer[i] = m_cReceiveBuffer[i];

      MessageBox(cReceiveBuffer);
}

void CGroundStationDlg::On3dm()
{
      // TODO: Add your control notification handler code here
      int i;
      unsigned char ucReceiveBuffer[25];
      signed short iReceiveBuffer[15];
      char temp[25];

//while(1)
//{
      char *cReceiveBuffer = (char*)malloc(0xFF);
      for (i = 0; i < 25; i++)
            ucReceiveBuffer[i] = '0';

      ucReceiveBuffer[24] = '\0';

      for (i = 0; i < 15; i++)
            iReceiveBuffer[i] = 0;

    cReceiveBuffer[0] = ' ';
      cReceiveBuffer[1] = '\0';

      // Windwos handle and Data Communication structure
      HANDLE hCom;
      LPDCB lpDcb;

      // Communication Parameters
      DWORD dwRead, dwWritten, dwResult;
      DWORD dwTimeout=25;
      OVERLAPPED OL1={0};
      BOOL blContinue=TRUE;

      // Allocate memory buffer for communication
      BYTE* bSendBuffer=(BYTE*) malloc(0xFFFF);
      if(bSendBuffer==NULL)
      {
            MessageBox(_T("Program Failed!Can Not Allocate Memory for
Tx"));
```

```
            return;
      }

      BYTE* bReceiveBuffer=(BYTE*) malloc(0xFFFF);
      if(bReceiveBuffer==NULL)
      {
            MessageBox(_T("Program Failed!Can Not Allocate Memory for
Rx"));
            return;
      }

      // Initialize HAndle
      hCom=CreateFile("\\.\\COM1:",GENERIC_READ|GENERIC_WRITE,0,NULL,
OPEN_EXISTING,FILE_FLAG_OVERLAPPED,NULL);
      if (hCom==INVALID_HANDLE_VALUE)
      {
            MessageBox("Invalid Handle Value For Communication
Port.Program Fails!");
            return ;
      }

      // Initialize Data Communication Structure
      lpDcb=new(DCB);

      // Get Communication Port State
      if (!GetCommState(hCom,lpDcb))
      {
            MessageBox("Program Fails To Get Communication Port
State.");
            return ;
      }

      // Build Communication Port State
      CString strTemp=_T("baud=38400 parity=N data=8 stop=1");
      if (!BuildCommDCB(strTemp,lpDcb))
      {
            MessageBox("Program Fails To Build Communication Port
State.");
            return ;
      }

      // Set Communication Port State
      if (!SetCommState(hCom,lpDcb))
      {
            MessageBox("Program Fails To Set Communication Port
State.");
            return ;
      }

      // Create an overlapped event
      OL1.hEvent=CreateEvent(NULL, TRUE, FALSE, NULL);
      if (OL1.hEvent ==NULL )
      {
            MessageBox(_T("Program Failed!Can Not Create Event1"));
            return ;
      }

      // Start Communication
      // First send flag for the Avionics unit to start sending data
      // Create a memory buffer of data we wish to send, set dwRead
to
```

```
      // the number of data in the buffer

    bSendBuffer[0] = 0x31;
      bSendBuffer[1] = '\0';
      i = 0;
      dwRead = 1;

      // Start sending files
      WriteFile(hCom,bSendBuffer,dwRead,&dwWritten,&OL1);
      if (GetLastError() != ERROR_IO_PENDING)
      {
            dwWritten=0 ;
      }
      else
      {
            // Write is pending.
            dwResult = WaitForSingleObject(OL1.hEvent, INFINITE);
            switch(dwResult)
            {
                  case WAIT_OBJECT_0:
                        if (!GetOverlappedResult(hCom, &OL1,
&dwWritten, FALSE))
                              dwWritten=0 ;
                        break;
                  case WAIT_ABANDONED:
                        dwWritten=0 ;
                        break;
                  case WAIT_TIMEOUT:
                        dwWritten=0 ;
                        break;
                  default:
                        dwWritten=0 ;
                        break;
            }
      }

      // Now wait for Avionics unit to send data
      do
      {
            dwRead=ReadFile(hCom,bReceiveBuffer,1, &dwRead, &OL1);
// Start Receive Data
            if (dwRead==0)
            {
                  // MEANS THAT NO CARRIAGE RETURN OR LINE FEED
OCCURRED
                  if (GetLastError() != ERROR_IO_PENDING)
                        dwRead=0 ;  // NEVER COMES HERE UNLESS SOME
GRAVE ERROR HAS OCCURRED
                  else
                  {
                        // THEREFORE GOES HERE INSTEAD. INFACT IF NO
CARRIAGE RETURN OCCURS THEN
                        // FLOW ENTERS THIS CONDITION ALWAYS
                        // Write is pending.
                        dwResult = WaitForSingleObject(OL1.hEvent,
dwTimeout); // Wait for receive to finish

                        switch(dwResult)
                        {
                              case WAIT_OBJECT_0:
```

```
                                    if (!GetOverlappedResult(hCom,
&OL1, &dwRead, FALSE))
                                          dwRead=0 ;
                                    break;
                              case WAIT_ABANDONED:
                                    dwRead=0 ;
                                    break;
                              case WAIT_TIMEOUT:
                                    dwRead=0 ;
                                    break;
                              default:
                                    dwRead=0 ;
                                    break;
                        }
                  }

                  if (dwRead==0)
                        blContinue=FALSE; // MEANS TIME OUT OR LAST
CHARACTER RECEIVED. COMES HERE JUST BEFORE EXITING
                  else
                  {
                        // MEANS THAT NO TIME OUT HAS OCCURRED....
BLAH BLAH
                        ucReceiveBuffer[i] = bReceiveBuffer[0];
                  }
            }
            else
            {
                // MEANS THAT A CARRIAGE RETURN OR LINE FEED OCCURRED
                  ucReceiveBuffer[i] = bReceiveBuffer[0];
            }

            // GOES EVERY TIME HEHE
            // We dont want to receive more than 185 bytes at the
same time or more than two strings which ever is first
            i++;
            if (i == 23)
            {
                  break;
            }

      }while(blContinue);

      Sleep(15);
      CloseHandle(OL1.hEvent);
      CloseHandle(hCom);
      free(bSendBuffer);
      free(bReceiveBuffer);

    iReceiveBuffer[0] = (signed short)ucReceiveBuffer[0];
      itoa(iReceiveBuffer[0], temp, 10);
      strcat(cReceiveBuffer, temp);
      strcat(cReceiveBuffer, " , ");
    int j = 1;

      for (i = 1; i < 23; i = i + 2)
      {
        iReceiveBuffer[j] = ucReceiveBuffer[i]*256 +
ucReceiveBuffer[i+1];
            j++;
      }
```

```
      float pitch, roll, yaw, xaccel, yaccel, zaccel, xrate, yrate,
zrate;

      pitch = (float)iReceiveBuffer[1]*360/65536;
      if (pitch > -0.1 && pitch < 0.1)
            pitch = 0.0;
            gcvt(pitch, 3, temp);
          strcat(cReceiveBuffer, temp);
            strcat(cReceiveBuffer, " , ");

      roll = (float)iReceiveBuffer[2]*360/65536;
      if (roll > -0.1 && roll < 0.1)
            roll = 0.0;
            gcvt(roll, 4, temp);
          strcat(cReceiveBuffer, temp);
            strcat(cReceiveBuffer, " , ");

      yaw = ((float)((unsigned short)iReceiveBuffer[3]))*360/65536;
      if (yaw > -0.1 && yaw < 0.1)
            yaw = 0.0;
            gcvt(yaw, 4, temp);
            m_strCompass = CString(temp);
          strcat(cReceiveBuffer, temp);
            strcat(cReceiveBuffer, " , ");

      xaccel = (float)iReceiveBuffer[4]*7000/32768000;
      if (xaccel > -0.1 && xaccel < 0.1)
            xaccel = 0.0;
            gcvt(xaccel, 2, temp);
            strcat (temp, "G");
            m_strTemp1 = CString(temp);
          strcat(cReceiveBuffer, temp);
            strcat(cReceiveBuffer, " , ");

      yaccel = (float)iReceiveBuffer[5]*7000/32768000;
      if (yaccel > -0.1 && yaccel < 0.1)
            yaccel = 0.0;
            gcvt(yaccel, 2, temp);
            strcat (temp, "G");
            m_strTemp2 = CString(temp);
          strcat(cReceiveBuffer, temp);
            strcat(cReceiveBuffer, " , ");

      zaccel = (float)iReceiveBuffer[6]*7000/32768000;
      if (zaccel > -0.1 && zaccel < 0.1)
            zaccel = 0.0;
            gcvt(zaccel, 2, temp);
            strcat (temp, "G");
            m_strTemp3 = CString(temp);
          strcat(cReceiveBuffer, temp);
            strcat(cReceiveBuffer, " , ");

      xrate =
(float)((double)iReceiveBuffer[7]*8500*360/(32768000*2*m_dPi));
      if (xrate > -0.1 && xrate < 0.1)
            xrate = 0.0;
            gcvt(xrate, 4, temp);
            strcat (temp, "deg/sec");
            m_strLatitude = CString(temp);
          strcat(cReceiveBuffer, temp);
```

```
            strcat(cReceiveBuffer, " , ");

      yrate =
(float)((double)iReceiveBuffer[8]*8500*360/(32768000*2*m_dPi));
      if (yrate > -0.1 && yrate < 0.1)
            yrate = 0.0;
            gcvt(yrate, 4, temp);
            strcat (temp, "deg/sec");
            m_strLongitude = CString(temp);
          strcat(cReceiveBuffer, temp);
            strcat(cReceiveBuffer, " , ");

      zrate =
(float)((double)iReceiveBuffer[9]*8500*360/(32768000*2*m_dPi));
      if (zrate > -0.1 && zrate < 0.1)
            zrate = 0.0;
            gcvt(zrate, 4, temp);
            strcat (temp, "deg/sec");
            m_strYaw = CString(temp);
          strcat(cReceiveBuffer, temp);
            strcat(cReceiveBuffer, " , ");

      strcat(cReceiveBuffer, "\0");

//    MessageBox(cReceiveBuffer);
      free(cReceiveBuffer);
      m_iUpdateCount++;
      if (m_iUpdateCount >= 15)
      {
            UpdateData(FALSE);
            m_iUpdateCount = 0;
      }

      m_aHorizon.SetAHRoll(pitch);
      m_aHorizon.SetAHPitch(roll);
      m_dlgCompass.Invalidate(TRUE);
      m_aHorizon.Invalidate(TRUE);

//}
}

void CGroundStationDlg::OnReleasedcaptureAltselect(NMHDR* pNMHDR,
LRESULT* pResult)
{
      // TODO: Add your control notification handler code here

      char temp[20];
      *pResult = 0;

      m_dAirSpeed = atof (m_strAirSpeed);
      m_dAltimeter = atof (m_strAltimeter);
      m_dCompass = atof (m_strCompass);

      UpdateData(TRUE);
      m_iAltSelLevel = 50 + 150 - m_iAltSelect;

      m_strAirSpeed = gcvt (m_dAirSpeed, 4, temp);
      m_strCompass = gcvt (m_dCompass, 4, temp);
      m_strAltimeter = gcvt (m_dAltimeter, 4, temp);
}
```

A-2: Compass/Altimeter/Speedometer Layer Code

The compass, altimeter, and speedometer layer codes are the same, since they all draw a circle with numbers in it to indicate the appropriate parameters.

Compass Code

```
// CompassDlg.cpp : implementation file
//

#include "stdafx.h"
#include "GroundStation.h"
#include "CompassDlg.h"

#include "GroundStationDlg.h"
#include "math.h"
#include "STDDEF.h"
#include "STDLIB.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif


/////////////////////////////////////////////////////////////////////////
////////
// CCompassDlg dialog


CCompassDlg::CCompassDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CCompassDlg::IDD, pParent)
{
	EnableAutomation();

	//{{AFX_DATA_INIT(CCompassDlg)
		// NOTE: the ClassWizard will add member initialization
here
	//}}AFX_DATA_INIT
}


void CCompassDlg::OnFinalRelease()
{
	// When the last reference for an automation object is released
	// OnFinalRelease is called.  The base class will automatically
	// deletes the object.  Add additional cleanup required for
your
	// object before calling the base class.

	CDialog::OnFinalRelease();
}
```

```
void CCompassDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CCompassDlg)
		// NOTE: the ClassWizard will add DDX and DDV calls here
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CCompassDlg, CDialog)
	//{{AFX_MSG_MAP(CCompassDlg)
		// NOTE: the ClassWizard will add message map macros here
	ON_WM_PAINT()
	ON_WM_SETCURSOR()
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

BEGIN_DISPATCH_MAP(CCompassDlg, CDialog)
	//{{AFX_DISPATCH_MAP(CCompassDlg)
		// NOTE - the ClassWizard will add and remove mapping
macros here.
	//}}AFX_DISPATCH_MAP
END_DISPATCH_MAP()

// Note: we add support for IID_ICompassDlg to support typesafe
binding
//  from VBA.  This IID must match the GUID that is attached to the
//  dispinterface in the .ODL file.

// {FE7E6C27-8196-44C8-820B-BF16EF75687F}
static const IID IID_ICompassDlg =
{ 0xfe7e6c27, 0x8196, 0x44c8, { 0x82, 0xb, 0xbf, 0x16, 0xef, 0x75,
0x68, 0x7f } };

BEGIN_INTERFACE_MAP(CCompassDlg, CDialog)
	INTERFACE_PART(CCompassDlg, IID_ICompassDlg, Dispatch)
END_INTERFACE_MAP()

////////////////////////////////////////////////////////////////////
////////
// CCompassDlg message handlers

void CCompassDlg::OnPaint()
{
	CPaintDC dc(this); // device context for painting

	// TODO: Add your message handler code here
	// Initialize variables
	CRect Rect, DrawRect, rRect;
	CPoint pStart, pEnd, pPoint;
	long isquare, itheta;
	double fradius, farm, ftheta, fSpeed;

	// Define constants
    const double dPi = 3.14159265358979323846264338327950;   // VALUE
OF PI

	const double dOuter1A = 0.75; // Outer circumference of outer
circle
	const double dOuter1B = 0.125;
```

```
      const double dOuter2A = 0.7;  // Inner circumference of outer
circle

      const double dInner1A = 0.6;  // Outer circumference of inner
circle
      const double dInner1B = 0.2;
      const double dInner2A = 0.55; // Inner circumference of inner
circle

      // Getting the parent window, and using it to get the speed
variable
      CGroundStationDlg *pWnd = (CGroundStationDlg*)GetParent();
      fSpeed = atof(pWnd->m_strCompass);

      // Giving a bound to the speed
      if (fSpeed <= 0.0)
            fSpeed = 0.0;

      if (fSpeed >= 360.0)
            fSpeed = 360.0;

      // Convert the speed into degrees
      ftheta = fSpeed;
      ftheta = ftheta - 90.0;
      itheta = (int)ftheta;

      dc.SetBkColor(RGB(128,128,128));
      dc.SetTextColor(RGB(255,255,255));

      // Creating several pens needed to draw the speed indicator arm
and speed
      //guages
      CPen WhitePen (PS_SOLID,2, RGB(255,255,255));
      CPen RedPen (PS_SOLID,3, RGB(128,128,128));
      CPen BlackPen (PS_SOLID,1, RGB(0,0,0));
    CPen GrayPen (PS_SOLID,1, RGB(128,128,128));

      // Createing brushes to paint a circle for the speedometer
      CBrush BBrush (RGB(0,0,0));
      CBrush WBrush (RGB(255,255,255));
      CBrush GBrush (RGB(128,128,128));

      // Getting rectanle holding of the speedometer window
    GetClientRect(Rect);
    Rect.NormalizeRect();

      // Once the main rectangle is defined, use its coordinates to
find the
      // center point of the rectangle
      pStart.y = Rect.top + Rect.Height()/2;
      pStart.x = Rect.left + Rect.Width()/2;

      // Get the difference between the longer and the shorter side
of the
      // rectangle.  Half the difference is used to position objects
from the
      // middle of the rectangle
      isquare = (long)((Rect.Width() - Rect.Height())/2);

      // The largest rectangle holding the entire speedometer.  This
particular
```

```
      // rectangle is used to draw the largest circle containing the
speed values
      // The following lines gets the coordinates of the rectangle
      DrawRect.top = Rect.top;
      DrawRect.left = Rect.left + isquare;
      DrawRect.bottom = DrawRect.top + Rect.Height() ;
      DrawRect.right = DrawRect.left + Rect.Height();

      // Use the above rectangle to draw the circle
      dc.SelectObject(&GrayPen);
      dc.SelectObject(&GBrush);
      dc.Ellipse(DrawRect);

      // isquare for medium rectangle is 0.75 of the original isquare
      isquare = (int)((Rect.Width() - dOuter1A*Rect.Height())/2);

      // The medium rectangle is to contain the speed guaze.
      // Define the coordinates of the rectangle holding the medium
rectangle
      // The medium rectangle is 0.75 the size of the largest
rectangle
      DrawRect.top = (long)(Rect.top + dOuter1B*Rect.Height());
      DrawRect.left = Rect.left + isquare;
      DrawRect.bottom = (long)(DrawRect.top +
dOuter1A*Rect.Height());
      DrawRect.right = (long)(DrawRect.left +
dOuter1A*Rect.Height());

      // Use the above rectangle to draw the circle
      dc.SelectObject(&BlackPen);
      dc.SelectObject(&BBrush);
      dc.Ellipse(DrawRect);


      // Define the radius of the arm for the medium size rectangle
      fradius = dOuter1A*Rect.Height()/2;
      farm = Rect.Height()/2;

      // Now draw the speed guages
      double i;
      for (i = -90.0; i < 270; i+=22.5)
      {
            pEnd.x = (long)(pStart.x + cos(i*dPi/180)*fradius);
            pEnd.y = (long)(pStart.y + sin(i*dPi/180)*fradius);
            dc.SelectObject(&RedPen);
            dc.MoveTo((int)(pStart.x +
cos(i*dPi/180)*dInner2A*Rect.Height()/2), (int)(pStart.y +
sin(i*dPi/180)*dInner2A*Rect.Height()/2));
            dc.LineTo(pEnd.x,pEnd.y);
      }

      // Now draw the speed values
      CString strSpeedValue;
      for (i = -90; i < 270; i+=90.0)
      {
            pEnd.x = (long)(pStart.x + cos(i*dPi/180)*fradius);
            pEnd.y = (long)(pStart.y + sin(i*dPi/180)*fradius);
            pPoint.x = (long)(pStart.x + cos(i*dPi/180)*farm);
            pPoint.y = (long)(pStart.y + sin(i*dPi/180)*farm);

            // Draw the speed value corresponding to the position
```

```
            if (i == -90.0)
                        strSpeedValue = "N";
            if (i == 0.0)
                        strSpeedValue = "E";
            if (i == 90.0)
                        strSpeedValue = "S";
            if (i == 180)
                        strSpeedValue = "W";

            // Now determine the position of the rectangle where the
speed value
            // will be located.
            // Check the x coordinates
            if (pEnd.x > pPoint.x)
            {
                  rRect.left = pPoint.x;
                  rRect.right = pEnd.x;
            }
            else if (pEnd.x == pPoint.x)
            {
                  rRect.left = pEnd.x - abs(pPoint.y-pEnd.y)/2;
                  rRect.right = pEnd.x + abs(pPoint.y-pEnd.y)/2;
            }
            else
            {
                  rRect.left = pEnd.x;
                  rRect.right = pPoint.x;
            }

            // Check the y coordinates
            if (pEnd.y > pPoint.y)
            {
                  rRect.top = pPoint.y;
                  rRect.bottom = pEnd.y;
            }
            else if (pEnd.y == pPoint.y)
            {
                  rRect.top = pEnd.y - abs(pPoint.x-pEnd.x)/2;
                  rRect.bottom = pEnd.y + abs(pPoint.x-pEnd.x)/2;
                  }
            else
            {
                  rRect.top = pEnd.y;
                  rRect.bottom = pPoint.y;
            }

            // Now draw the appropriate text selected for the
appropriate position
            dc.DrawText(strSpeedValue,rRect,DT_CENTER);
      }

      // isquare for smallest rectangle is 0.6 of the original
isquare
      isquare = (long)((Rect.Width() - dInner1A*Rect.Height())/2);

      // The smallest rectangle is to contain the speed indicator
arm.
      // Define the coordinates of the rectangle holding the smallest
rectangle
      // The medium rectangle is 0.6 the size of the largest
rectangle
```

```
	DrawRect.top = (long)(Rect.top + dInner1B*Rect.Height());
	DrawRect.left = Rect.left + isquare;
	DrawRect.bottom = (long)(DrawRect.top +
dInner1A*Rect.Height());
	DrawRect.right = (long)(DrawRect.left +
dInner1A*Rect.Height());

	// Use the above rectangle to draw the circle
	dc.SelectObject(&BlackPen);
	dc.SelectObject(&BBrush);
	dc.Ellipse(DrawRect);

	// Finally draw the arm depending on the speed
	fradius = dInner2A*Rect.Height()/2;
	pEnd.x = (long)(pStart.x + cos((ftheta)*dPi/180)*fradius);
	pEnd.y = (long)(pStart.y + sin((ftheta)*dPi/180)*fradius);
	dc.SelectObject(&WhitePen);
	dc.MoveTo(pStart.x,pStart.y);
	dc.LineTo(pEnd.x,pEnd.y);


	// Do not call CDialog::OnPaint() for painting messages
}

BOOL CCompassDlg::OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT
message)
{
	// TODO: Add your message handler code here and/or call default
	CGroundStationDlg *cWnd = (CGroundStationDlg*)GetParent();
	bool bCursor = cWnd->m_bCursor;
	if (bCursor == TRUE)
		return TRUE;
	else


	return CDialog::OnSetCursor(pWnd, nHitTest, message);
}
```

Altimeter Code

```
// AltimeterDlg.cpp : implementation file
//

#include "stdafx.h"
#include "GroundStation.h"
#include "AltimeterDlg.h"

#include "GroundStationDlg.h"
#include "math.h"
#include "STDDEF.h"
#include "STDLIB.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
////////////////////////////////////////////////////////////////////////
////////
// CAltimeterDlg dialog


CAltimeterDlg::CAltimeterDlg(CWnd* pParent /*=NULL*/)
      : CDialog(CAltimeterDlg::IDD, pParent)
{
      EnableAutomation();

      //{{AFX_DATA_INIT(CAltimeterDlg)
            // NOTE: the ClassWizard will add member initialization
here
      //}}AFX_DATA_INIT
}


void CAltimeterDlg::OnFinalRelease()
{
      // When the last reference for an automation object is released
      // OnFinalRelease is called.  The base class will automatically
      // deletes the object.  Add additional cleanup required for
your
      // object before calling the base class.

      CDialog::OnFinalRelease();
}

void CAltimeterDlg::DoDataExchange(CDataExchange* pDX)
{
      CDialog::DoDataExchange(pDX);
      //{{AFX_DATA_MAP(CAltimeterDlg)
            // NOTE: the ClassWizard will add DDX and DDV calls here
      //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CAltimeterDlg, CDialog)
      //{{AFX_MSG_MAP(CAltimeterDlg)
            // NOTE: the ClassWizard will add message map macros here
      ON_WM_PAINT()
      ON_WM_SETCURSOR()
      //}}AFX_MSG_MAP
END_MESSAGE_MAP()

BEGIN_DISPATCH_MAP(CAltimeterDlg, CDialog)
      //{{AFX_DISPATCH_MAP(CAltimeterDlg)
            // NOTE - the ClassWizard will add and remove mapping
macros here.
      //}}AFX_DISPATCH_MAP
END_DISPATCH_MAP()

// Note: we add support for IID_IAltimeterDlg to support typesafe
binding
//  from VBA.  This IID must match the GUID that is attached to the
//  dispinterface in the .ODL file.

// {4717B343-EF9B-4A17-B60A-82830111E9F7}
static const IID IID_IAltimeterDlg =
{ 0x4717b343, 0xef9b, 0x4a17, { 0xb6, 0xa, 0x82, 0x83, 0x1, 0x11,
0xe9, 0xf7 } };
```

```
BEGIN_INTERFACE_MAP(CAltimeterDlg, CDialog)
      INTERFACE_PART(CAltimeterDlg, IID_IAltimeterDlg, Dispatch)
END_INTERFACE_MAP()

////////////////////////////////////////////////////////////////////////
////////
// CAltimeterDlg message handlers

void CAltimeterDlg::OnPaint()
{
      CPaintDC dc(this); // device context for painting

      // TODO: Add your message handler code here
      // Initialize variables
      CRect Rect, DrawRect, rRect;
      CPoint pStart, pEnd, pPoint;
      long isquare, itheta;
      double fradius, farm, ftheta, fSpeed;

      // Define constants
    const double dPi = 3.14159265358979323846264338327795;   // VALUE
OF PI

      const double dOuter1A = 0.75; // Outer circumference of outer
circle
      const double dOuter1B = 0.125;
      const double dOuter2A = 0.7;  // Inner circumference of outer
circle

      const double dInner1A = 0.6;  // Outer circumference of inner
circle
      const double dInner1B = 0.2;
      const double dInner2A = 0.55; // Inner circumference of inner
circle

      // Getting the parent window, and using it to get the speed
variable
      CGroundStationDlg *pWnd = (CGroundStationDlg*)GetParent();
      fSpeed = atof(pWnd->m_strAltimeter);
      fSpeed = fSpeed*360.0/400.0;

      // Giving a bound to the speed
      if (fSpeed <= 0.0)
            fSpeed = 0.0;

      if (fSpeed >= 360.0)
            fSpeed = 360.0;

      // Convert the speed into degrees
      ftheta = fSpeed;
      itheta = (int)ftheta;

      dc.SetBkColor(RGB(128,128,128));
      dc.SetTextColor(RGB(255,255,255));

      // Creating several pens needed to draw the speed indicator arm
and speed
      //guages
      CPen WhitePen (PS_SOLID,2, RGB(255,255,255));
      CPen RedPen (PS_SOLID,2, RGB(255,0,0));
```

```
      CPen YellowPen (PS_SOLID,2, RGB(255,255,0));
      CPen GreenPen (PS_SOLID,2, RGB(0,255,0));
      CPen BlackPen (PS_SOLID,1, RGB(0,0,0));
    CPen GrayPen (PS_SOLID,1, RGB(128,128,128));

      // Createing brushes to paint a circle for the speedometer
      CBrush BBrush (RGB(0,0,0));
      CBrush WBrush (RGB(255,255,255));
      CBrush GBrush (RGB(128,128,128));

      // Getting rectanle holding of the speedometer window
    GetClientRect(Rect);
    Rect.NormalizeRect();

      // Once the main rectangle is defined, use its coordinates to
find the
      // center point of the rectangle
      pStart.y = Rect.top + Rect.Height()/2;
      pStart.x = Rect.left + Rect.Width()/2;

      // Get the difference between the longer and the shorter side
of the
      // rectangle.  Half the difference is used to position objects
from the
      // middle of the rectangle
      isquare = (long)((Rect.Width() - Rect.Height())/2);

      // The largest rectangle holding the entire speedometer.  This
particular
      // rectangle is used to draw the largest circle containing the
speed values
      // The following lines gets the coordinates of the rectangle
      DrawRect.top = Rect.top;
      DrawRect.left = Rect.left + isquare;
      DrawRect.bottom = DrawRect.top + Rect.Height() ;
      DrawRect.right = DrawRect.left + Rect.Height();

      // Use the above rectangle to draw the circle
      dc.SelectObject(&GrayPen);
      dc.SelectObject(&GBrush);
      dc.Ellipse(DrawRect);

      // isquare for medium rectangle is 0.75 of the original isquare
      isquare = (int)((Rect.Width() - dOuter1A*Rect.Height())/2);

      // The medium rectangle is to contain the speed guaze.
      // Define the coordinates of the rectangle holding the medium
rectangle
      // The medium rectangle is 0.75 the size of the largest
rectangle
      DrawRect.top = (long)(Rect.top + dOuter1B*Rect.Height());
      DrawRect.left = Rect.left + isquare;
      DrawRect.bottom = (long)(DrawRect.top +
dOuter1A*Rect.Height());
      DrawRect.right = (long)(DrawRect.left +
dOuter1A*Rect.Height());

      // Use the above rectangle to draw the circle
      dc.SelectObject(&BlackPen);
      dc.SelectObject(&BBrush);
      dc.Ellipse(DrawRect);
```

```
      // Define the radius of the arm for the medium size rectangle
      fradius = dOuter2A*Rect.Height()/2;

      // Now draw the speed guages
      int i;
      for (i = -90; i < 270; i+=9)
      {
            pEnd.x = (long)(pStart.x + cos(i*dPi/180)*fradius);
            pEnd.y = (long)(pStart.y + sin(i*dPi/180)*fradius);

            if (i < 90)
                  dc.SelectObject(&GreenPen);
            else if (i < 180)
                  dc.SelectObject(&YellowPen);
            else
                  dc.SelectObject(&RedPen);

            dc.MoveTo((int)(pStart.x +
cos(i*dPi/180)*dInner2A*Rect.Height()/2), (int)(pStart.y +
sin(i*dPi/180)*dInner2A*Rect.Height()/2));
            dc.LineTo(pEnd.x,pEnd.y);
      }

      // Now draw the speed values
      char caSpeedValue[5];
      fradius = dOuter1A*Rect.Height()/2;
      farm = Rect.Height()/2;

      for (i = -90; i < 270; i+=45)
      {
            pEnd.x = (long)(pStart.x + cos(i*dPi/180)*fradius);
            pEnd.y = (long)(pStart.y + sin(i*dPi/180)*fradius);
            pPoint.x = (long)(pStart.x + cos(i*dPi/180)*farm);
            pPoint.y = (long)(pStart.y + sin(i*dPi/180)*farm);

            // Draw the speed value corresponding to the position
            itoa((10 + (i/9)), caSpeedValue, 10);

            // Now determine the position of the rectangle where the
speed value
            // will be located.
            // Check the x coordinates
            if (pEnd.x > pPoint.x)
            {
                  rRect.left = pPoint.x;
                  rRect.right = pEnd.x;
            }
            else if (pEnd.x == pPoint.x)
            {
                  rRect.left = pEnd.x - abs(pPoint.y-pEnd.y)/2;
                  rRect.right = pEnd.x + abs(pPoint.y-pEnd.y)/2;
            }
            else
            {
                  rRect.left = pEnd.x;
                  rRect.right = pPoint.x;
            }

            // Check the y coordinates
            if (pEnd.y > pPoint.y)
```

```
            {
                  rRect.top = pPoint.y;
                  rRect.bottom = pEnd.y;
            }
            else if (pEnd.y == pPoint.y)
            {
                  rRect.top = pEnd.y - abs(pPoint.x-pEnd.x)/2;
                  rRect.bottom = pEnd.y + abs(pPoint.x-pEnd.x)/2;
                  }
            else
            {
                  rRect.top = pEnd.y;
                  rRect.bottom = pPoint.y;
            }

            // Now draw the appropriate text selected for the
appropriate position
            dc.DrawText(caSpeedValue,rRect,DT_CENTER);
      }

      // isquare for smallest rectangle is 0.6 of the original
isquare
      isquare = (long)((Rect.Width() - dInner1A*Rect.Height())/2);

      // The smallest rectangle is to contain the speed indicator
arm.
      // Define the coordinates of the rectangle holding the smallest
rectangle
      // The medium rectangle is 0.6 the size of the largest
rectangle
      DrawRect.top = (long)(Rect.top + dInner1B*Rect.Height());
      DrawRect.left = Rect.left + isquare;
      DrawRect.bottom = (long)(DrawRect.top +
dInner1A*Rect.Height());
      DrawRect.right = (long)(DrawRect.left +
dInner1A*Rect.Height());

      // Use the above rectangle to draw the circle
      dc.SelectObject(&BlackPen);
      dc.SelectObject(&BBrush);
      dc.Ellipse(DrawRect);

      // Finally draw the arm depending on the speed
      fradius = dInner2A*Rect.Height()/2;
      pEnd.x = (long)(pStart.x + cos((ftheta-90.0)*dPi/180)*fradius);
      pEnd.y = (long)(pStart.y + sin((ftheta-90.0)*dPi/180)*fradius);
      dc.SelectObject(&WhitePen);
      dc.MoveTo(pStart.x,pStart.y);
      dc.LineTo(pEnd.x,pEnd.y);


      // Do not call CDialog::OnPaint() for painting messages
}

BOOL CAltimeterDlg::OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT
message)
{
      // TODO: Add your message handler code here and/or call default
      CGroundStationDlg *cWnd = (CGroundStationDlg*)GetParent();
      bool bCursor = cWnd->m_bCursor;
      if (bCursor == TRUE)
```

```
            return TRUE;
      else

      return CDialog::OnSetCursor(pWnd, nHitTest, message);
}
```

Speedometer Code

```
// SpeedometerDlg.cpp : implementation file
//

#include "stdafx.h"
#include "GroundStation.h"
#include "SpeedometerDlg.h"

#include "GroundStationDlg.h"
#include "math.h"
#include "STDDEF.h"
#include "STDLIB.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif



/////////////////////////////////////////////////////////////////////////
////////
// CSpeedometerDlg dialog


CSpeedometerDlg::CSpeedometerDlg(CWnd* pParent /*=NULL*/)
      : CDialog(CSpeedometerDlg::IDD, pParent)
{
      EnableAutomation();

      //{{AFX_DATA_INIT(CSpeedometerDlg)
            // NOTE: the ClassWizard will add member initialization
here
      //}}AFX_DATA_INIT
}


void CSpeedometerDlg::OnFinalRelease()
{
      // When the last reference for an automation object is released
      // OnFinalRelease is called.  The base class will automatically
      // deletes the object.  Add additional cleanup required for
your
      // object before calling the base class.

      CDialog::OnFinalRelease();
}
```

```
void CSpeedometerDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CSpeedometerDlg)
		// NOTE: the ClassWizard will add DDX and DDV calls here
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CSpeedometerDlg, CDialog)
	//{{AFX_MSG_MAP(CSpeedometerDlg)
		// NOTE: the ClassWizard will add message map macros here
	ON_WM_PAINT()
	ON_WM_SETCURSOR()
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

BEGIN_DISPATCH_MAP(CSpeedometerDlg, CDialog)
	//{{AFX_DISPATCH_MAP(CSpeedometerDlg)
		// NOTE - the ClassWizard will add and remove mapping
macros here.
	//}}AFX_DISPATCH_MAP
END_DISPATCH_MAP()

// Note: we add support for IID_ISpeedometerDlg to support typesafe
binding
//  from VBA.  This IID must match the GUID that is attached to the
//  dispinterface in the .ODL file.

// {DEA1947A-6D70-454A-B8C0-A16DC68EED6E}
static const IID IID_ISpeedometerDlg =
{ 0xdea1947a, 0x6d70, 0x454a, { 0xb8, 0xc0, 0xa1, 0x6d, 0xc6, 0x8e,
0xed, 0x6e } };

BEGIN_INTERFACE_MAP(CSpeedometerDlg, CDialog)
	INTERFACE_PART(CSpeedometerDlg, IID_ISpeedometerDlg, Dispatch)
END_INTERFACE_MAP()

////////////////////////////////////////////////////////////////////
////////
// CSpeedometerDlg message handlers


void CSpeedometerDlg::OnPaint()
{
	CPaintDC dc(this); // device context for painting

	// TODO: Add your message handler code here

	// Initialize variables
	CRect Rect, DrawRect, rRect;
	CPoint pStart, pEnd, pPoint;
	long isquare, itheta;
	double fradius, farm, ftheta, fSpeed;

	// Define constants
    const double dPi = 3.14159265358979323846264338327795;   // VALUE
OF PI

	const double dOuter1A = 0.75; // Outer circumference of outer
circle
```

```
      const double dOuter1B = 0.125;
      const double dOuter2A = 0.7;  // Inner circumference of outer
circle

      const double dInner1A = 0.6;  // Outer circumference of inner
circle
      const double dInner1B = 0.2;
      const double dInner2A = 0.55; // Inner circumference of inner
circle

      // Getting the parent window, and using it to get the speed
variable
      CGroundStationDlg *pWnd = (CGroundStationDlg*)GetParent();
      fSpeed = atof(pWnd->m_strAirSpeed);

      // Giving a bound to the speed
      if (fSpeed <= 0)
            fSpeed = 0;

      if (fSpeed >= 30)
            fSpeed = 30;

      // Convert the speed into degrees
      ftheta = fSpeed/0.1111;
      itheta = (long)ftheta;

      dc.SetBkColor(RGB(128,128,128));
      dc.SetTextColor(RGB(255,255,255));

      // Creating several pens needed to draw the speed indicator arm
and speed
      //guages
      CPen WhitePen (PS_SOLID,2, RGB(255,255,255));
      CPen RedPen (PS_SOLID,2, RGB(255,0,0));
      CPen YellowPen (PS_SOLID,2, RGB(255,255,0));
      CPen GreenPen (PS_SOLID,2, RGB(0,255,0));
      CPen BlackPen (PS_SOLID,1, RGB(0,0,0));
    CPen GrayPen (PS_SOLID,1, RGB(128,128,128));

      // Createing brushes to paint a circle for the speedometer
      CBrush BBrush (RGB(0,0,0));
      CBrush WBrush (RGB(255,255,255));
      CBrush GBrush (RGB(128,128,128));

      // Getting rectanle holding of the speedometer window
    GetClientRect(Rect);
    Rect.NormalizeRect();

      // Once the main rectangle is defined, use its coordinates to
find the
      // center point of the rectangle
      pStart.y = Rect.top + Rect.Height()/2;
      pStart.x = Rect.left + Rect.Width()/2;

      // Get the difference between the longer and the shorter side
of the
      // rectangle.  Half the difference is used to position objects
from the
      // middle of the rectangle
      isquare = (long)((Rect.Width() - Rect.Height())/2);
```

```
      // The largest rectangle holding the entire speedometer.  This
particular
      // rectangle is used to draw the largest circle containing the
speed values
      // The following lines gets the coordinates of the rectangle
      DrawRect.top = Rect.top;
      DrawRect.left = Rect.left + isquare;
      DrawRect.bottom = DrawRect.top + Rect.Height() ;
      DrawRect.right = DrawRect.left + Rect.Height();

      // Use the above rectangle to draw the circle
      dc.SelectObject(&GrayPen);
      dc.SelectObject(&GBrush);
      dc.Ellipse(DrawRect);

      // isquare for medium rectangle is 0.75 of the original isquare
      isquare = (long)((Rect.Width() - dOuter1A*Rect.Height())/2);

      // The medium rectangle is to contain the speed guaze.
      // Define the coordinates of the rectangle holding the medium
rectangle
      // The medium rectangle is 0.75 the size of the largest
rectangle
      DrawRect.top = (long)(Rect.top + dOuter1B*Rect.Height());
      DrawRect.left = Rect.left + isquare;
      DrawRect.bottom = (long)(DrawRect.top +
dOuter1A*Rect.Height());
      DrawRect.right = (long)(DrawRect.left +
dOuter1A*Rect.Height());

      // Use the above rectangle to draw the circle
      dc.SelectObject(&BlackPen);
      dc.SelectObject(&BBrush);
      dc.Ellipse(DrawRect);

      // Define the radius of the arm for the medium size rectangle
      fradius = dOuter2A*Rect.Height()/2;

      // Now draw the speed guages
      int i;
      for (i = -45; i <= 225; i+=9)
      {
            pEnd.x = (long)(pStart.x + cos(i*dPi/180)*fradius);
            pEnd.y = (long)(pStart.y + sin(i*dPi/180)*fradius);

            if (i < 90)
                  dc.SelectObject(&GreenPen);
            else if (i < 180)
                  dc.SelectObject(&YellowPen);
            else
                  dc.SelectObject(&RedPen);

            dc.MoveTo((int)(pStart.x +
cos(i*dPi/180)*dInner2A*Rect.Height()/2), (int)(pStart.y +
sin(i*dPi/180)*dInner2A*Rect.Height()/2));
            dc.LineTo(pEnd.x,pEnd.y);
      }

      // Now draw the speed values
      char caSpeedValue[5];
      fradius = dOuter1A*Rect.Height()/2;
```

```
      farm = Rect.Height()/2;

      for (i = -45; i <= 225; i+=45)
      {
            pEnd.x = (long)(pStart.x + cos(i*dPi/180)*fradius);
            pEnd.y = (long)(pStart.y + sin(i*dPi/180)*fradius);
            pPoint.x = (long)(pStart.x + cos(i*dPi/180)*farm);
            pPoint.y = (long)(pStart.y + sin(i*dPi/180)*farm);

            // Draw the speed value corresponding to the position
            itoa((5 + (i/9)), caSpeedValue, 10);

            // Now determine the position of the rectangle where the
speed value
            // will be located.
            // Check the x coordinates
            if (pEnd.x > pPoint.x)
            {
                  rRect.left = pPoint.x;
                  rRect.right = pEnd.x;
            }
            else if (pEnd.x == pPoint.x)
            {
                  rRect.left = pEnd.x - abs(pPoint.y-pEnd.y)/2;
                  rRect.right = pEnd.x + abs(pPoint.y-pEnd.y)/2;
            }
            else
            {
                  rRect.left = pEnd.x;
                  rRect.right = pPoint.x;
            }

            // Check the y coordinates
            if (pEnd.y > pPoint.y)
            {
                  rRect.top = pPoint.y;
                  rRect.bottom = pEnd.y;
            }
            else if (pEnd.y == pPoint.y)
            {
                  rRect.top = pEnd.y - abs(pPoint.x-pEnd.x)/2;
                  rRect.bottom = pEnd.y + abs(pPoint.x-pEnd.x)/2;
                  }
            else
            {
                  rRect.top = pEnd.y;
                  rRect.bottom = pPoint.y;
            }

            // Now draw the appropriate text selected for the
appropriate position
            dc.DrawText(caSpeedValue,rRect,DT_CENTER);
      }

      // isquare for smallest rectangle is 0.6 of the original
isquare
      isquare = (long)((Rect.Width() - dInner1A*Rect.Height())/2);

      // The smallest rectangle is to contain the speed indicator
arm.
```

```
	// Define the coordinates of the rectangle holding the smallest
rectangle
	// The medium rectangle is 0.6 the size of the largest
rectangle
	DrawRect.top = (long)(Rect.top + dInner1B*Rect.Height());
	DrawRect.left = Rect.left + isquare;
	DrawRect.bottom = (long)(DrawRect.top +
dInner1A*Rect.Height());
	DrawRect.right = (long)(DrawRect.left +
dInner1A*Rect.Height());

	// Use the above rectangle to draw the circle
	dc.SelectObject(&BlackPen);
	dc.SelectObject(&BBrush);
	dc.Ellipse(DrawRect);

	// Finally draw the arm depending on the speed
	fradius = dInner2A*Rect.Height()/2;
	pEnd.x = (long)(pStart.x + cos((ftheta-45.0)*dPi/180)*fradius);
	pEnd.y = (long)(pStart.y + sin((ftheta-45.0)*dPi/180)*fradius);
	dc.SelectObject(&WhitePen);
	dc.MoveTo(pStart.x,pStart.y);
	dc.LineTo(pEnd.x,pEnd.y);

	// Do not call CDialog::OnPaint() for painting messages
}

BOOL CSpeedometerDlg::OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT
message)
{
	// TODO: Add your message handler code here and/or call default
	CGroundStationDlg *cWnd = (CGroundStationDlg*)GetParent();
	bool bCursor = cWnd->m_bCursor;
	if (bCursor == TRUE)
		return TRUE;
	else


	return CDialog::OnSetCursor(pWnd, nHitTest, message);
}
```

A-3: Map Layer Code

```
// MapDlg.cpp : implementation file
//

#include "stdafx.h"
#include "GroundStation.h"
#include "MapDlg.h"

#include "GroundStationDlg.h"
#include "math.h"
#include "STDDEF.h"
#include "STDLIB.h"

#ifdef _DEBUG
#define new DEBUG_NEW
```

```
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////
////////
// CMapDlg dialog


CMapDlg::CMapDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CMapDlg::IDD, pParent)
{
	EnableAutomation();

	//{{AFX_DATA_INIT(CMapDlg)
		// NOTE: the ClassWizard will add member initialization
here
	//}}AFX_DATA_INIT
}


void CMapDlg::OnFinalRelease()
{
	// When the last reference for an automation object is released
	// OnFinalRelease is called.  The base class will automatically
	// deletes the object.  Add additional cleanup required for
your
	// object before calling the base class.

	CDialog::OnFinalRelease();
}



void CMapDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CMapDlg)
		// NOTE: the ClassWizard will add DDX and DDV calls here
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CMapDlg, CDialog)
	//{{AFX_MSG_MAP(CMapDlg)
		// NOTE: the ClassWizard will add message map macros here
	ON_WM_PAINT()
	ON_WM_SETCURSOR()
	ON_WM_LBUTTONDOWN()
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

BEGIN_DISPATCH_MAP(CMapDlg, CDialog)
	//{{AFX_DISPATCH_MAP(CMapDlg)
		// NOTE - the ClassWizard will add and remove mapping
macros here.
	//}}AFX_DISPATCH_MAP
END_DISPATCH_MAP()
```

```
// Note: we add support for IID_IMapDlg to support typesafe binding
//  from VBA.  This IID must match the GUID that is attached to the
//  dispinterface in the .ODL file.

// {22FE9BB8-42FC-4ED9-96E5-CBD5E699AA85}
static const IID IID_IMapDlg =
{ 0x22fe9bb8, 0x42fc, 0x4ed9, { 0x96, 0xe5, 0xcb, 0xd5, 0xe6, 0x99,
0xaa, 0x85 } };

BEGIN_INTERFACE_MAP(CMapDlg, CDialog)
      INTERFACE_PART(CMapDlg, IID_IMapDlg, Dispatch)
END_INTERFACE_MAP()

////////////////////////////////////////////////////////////////////
////////
// CMapDlg message handlers

void CMapDlg::OnPaint()
{
      CPaintDC dc(this); // device context for painting

      // TODO: Add your message handler code here

      // Get the parent class
      // Getting the parent window, and using it to get the speed
variable
      CGroundStationDlg *pWnd = (CGroundStationDlg*)GetParent();
      int iXPos = pWnd->m_iXPos;
      int iYPos = pWnd->m_iYPos;
      int iWidth = pWnd->m_iWidth;
      int iHeight = pWnd->m_iHeight;
      bool bMap = pWnd->m_bMap;
      bool bPlaneSim    = pWnd->m_bPlaneSim;
      bool bGenPath = pWnd->m_bGenPath;

      // Create a rectangle that has the scree coordinates of the
whole window
      CRect rMap;
      GetClientRect(&rMap);

      if (bMap == TRUE && bPlaneSim == FALSE)
      {
            // Initialize variables and objects
            CBitmap *bmpMap = &(pWnd->m_bmpMap);
            BITMAP bmMap;

            // Get the loaded bitmap
            bmpMap->GetBitmap(&bmMap);
            CDC dcMem;

            // Create a device context to load the bitmap into
            dcMem.CreateCompatibleDC(&dc);

            // Select the bitmap into the compatible device context
            dcMem.SelectObject(bmpMap);

            // Copy and resize the bitmap into the dialog window
            dc.StretchBlt(0,0, rMap.Width(), rMap.Height(), &dcMem,
iXPos, iYPos, iWidth, iHeight, SRCCOPY);
      }
```

```
	if (m_iWayPointIndex > 0)
	{
		CPen pGrayPen(PS_SOLID, 2, RGB(210,210,210));
		dc.SelectObject(&pGrayPen);
		int i;
		for (i = 0; i < m_iWayPointIndex; i++)
		{
			dc.MoveTo(m_iWayPoints[0][i],m_iWayPoints[1][i]);
			dc.LineTo((m_iWayPoints[0][i] +
5),(m_iWayPoints[1][i] + 5));
			dc.MoveTo(m_iWayPoints[0][i],m_iWayPoints[1][i]);
			dc.LineTo((m_iWayPoints[0][i] -
5),(m_iWayPoints[1][i] - 5));
			dc.MoveTo(m_iWayPoints[0][i],m_iWayPoints[1][i]);
			dc.LineTo((m_iWayPoints[0][i] +
5),(m_iWayPoints[1][i] - 5));
			dc.MoveTo(m_iWayPoints[0][i],m_iWayPoints[1][i]);
			dc.LineTo((m_iWayPoints[0][i] -
5),(m_iWayPoints[1][i] + 5));
		}
	}

	if (m_iSetPointIndex > 0)
	{
		CPen pGrayPen(PS_SOLID, 2, RGB(210,210,210));
		dc.SelectObject(&pGrayPen);
		int i;
		for (i = 0; i < m_iSetPointIndex-2; i = i + 3)
		{
			dc.MoveTo(m_iSetPoints[0][i],m_iSetPoints[1][i]);
			dc.LineTo((m_iSetPoints[0][i] +
5),(m_iSetPoints[1][i] + 5));
			dc.MoveTo(m_iSetPoints[0][i],m_iSetPoints[1][i]);
			dc.LineTo((m_iSetPoints[0][i] -
5),(m_iSetPoints[1][i] - 5));
			dc.MoveTo(m_iSetPoints[0][i],m_iSetPoints[1][i]);
			dc.LineTo((m_iSetPoints[0][i] +
5),(m_iSetPoints[1][i] - 5));
			dc.MoveTo(m_iSetPoints[0][i],m_iSetPoints[1][i]);
			dc.LineTo((m_iSetPoints[0][i] -
5),(m_iSetPoints[1][i] + 5));


	dc.MoveTo(m_iSetPoints[0][i+1],m_iSetPoints[1][i+1]);
			dc.LineTo((m_iSetPoints[0][i+1] +
5),(m_iSetPoints[1][i+1] + 5));

	dc.MoveTo(m_iSetPoints[0][i+1],m_iSetPoints[1][i+1]);
			dc.LineTo((m_iSetPoints[0][i+1] -
5),(m_iSetPoints[1][i+1] - 5));

	dc.MoveTo(m_iSetPoints[0][i+1],m_iSetPoints[1][i+1]);
			dc.LineTo((m_iSetPoints[0][i+1] +
5),(m_iSetPoints[1][i+1] - 5));

	dc.MoveTo(m_iSetPoints[0][i+1],m_iSetPoints[1][i+1]);
			dc.LineTo((m_iSetPoints[0][i+1] -
5),(m_iSetPoints[1][i+1] + 5));


	dc.MoveTo(m_iSetPoints[0][i+2],m_iSetPoints[1][i+2]);
```

```
                dc.LineTo((m_iSetPoints[0][i+2] +
5),(m_iSetPoints[1][i+2] + 5));

      dc.MoveTo(m_iSetPoints[0][i+2],m_iSetPoints[1][i+2]);
                dc.LineTo((m_iSetPoints[0][i+2] -
5),(m_iSetPoints[1][i+2] - 5));

      dc.MoveTo(m_iSetPoints[0][i+2],m_iSetPoints[1][i+2]);
                dc.LineTo((m_iSetPoints[0][i+2] +
5),(m_iSetPoints[1][i+2] - 5));

      dc.MoveTo(m_iSetPoints[0][i+2],m_iSetPoints[1][i+2]);
                dc.LineTo((m_iSetPoints[0][i+2] -
5),(m_iSetPoints[1][i+2] + 5));
            }
      }


      if ((bGenPath == TRUE) && (m_iGpsIndex > 0))
      {
            int i;
            for (i = 0; i < m_iGpsIndex; i++)
            {
                  double dAltLevels = (double)m_iGpsCoor[4][i];
                  int red_color, green_color, blue_color;
                  green_color = 0;
                  red_color = (int)(-127.5 +
(255.0/100.0)*dAltLevels);
                  blue_color = (int)(382.5 -
(255.0/100.0)*dAltLevels);

      dc.SetPixelV(m_iGpsCoor[0][i],m_iGpsCoor[1][i],RGB(red_color,gr
een_color,blue_color));
                  dc.SetPixelV((m_iGpsCoor[0][i] +
1),(m_iGpsCoor[1][i] + 1),RGB(red_color,green_color,blue_color));
                  dc.SetPixelV((m_iGpsCoor[0][i] -
1),(m_iGpsCoor[1][i] - 1),RGB(red_color,green_color,blue_color));
                  dc.SetPixelV((m_iGpsCoor[0][i] +
1),(m_iGpsCoor[1][i] - 1),RGB(red_color,green_color,blue_color));
                  dc.SetPixelV((m_iGpsCoor[0][i] -
1),(m_iGpsCoor[1][i] + 1),RGB(red_color,green_color,blue_color));
            }
      }


      if (bMap == FALSE)
      {
            dc.SetBkColor(RGB(210,210,210));
            dc.SetTextColor(RGB(0,0,0));
            dc.DrawText("Please Load a Map",rMap,DT_CENTER);
      }
      // Do not call CDialog::OnPaint() for painting messages
}


BOOL CMapDlg::OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT message)
{
      // TODO: Add your message handler code here and/or call default
      CGroundStationDlg *cWnd = (CGroundStationDlg*)GetParent();
      bool bCursor = cWnd->m_bCursor;
      if (bCursor == TRUE)
```

```
            return TRUE;
      else


      return CDialog::OnSetCursor(pWnd, nHitTest, message);
}


void CMapDlg::OnLButtonDown(UINT nFlags, CPoint point)
{
      // TODO: Add your message handler code here and/or call default
      int iAltSel = 0;

      // Get the device context
      CClientDC dc(this);

      // Choose a pen color and thickness
      CPen pGrayPen(PS_SOLID, 2, RGB(210,210,210));
      dc.SelectObject(&pGrayPen);


      // Getting the parent window, and using it to get the speed
variable
      CGroundStationDlg *pWnd = (CGroundStationDlg*)GetParent();
      iAltSel = pWnd->m_iAltSelLevel;

      // Draw line from a previous point to the current point
      dc.MoveTo(point.x,point.y);
      dc.LineTo((point.x + 5),(point.y + 5));
      dc.MoveTo(point.x,point.y);
      dc.LineTo((point.x - 5),(point.y - 5));
      dc.MoveTo(point.x,point.y);
      dc.LineTo((point.x + 5),(point.y - 5));
      dc.MoveTo(point.x,point.y);
      dc.LineTo((point.x - 5),(point.y + 5));

      m_iWayPoints[0][m_iWayPointIndex] = point.x;
      m_iWayPoints[1][m_iWayPointIndex] = point.y;
      m_iWayPoints[2][m_iWayPointIndex] = iAltSel;
      m_iWayPointIndex++;

      // Draw the pixel
      dc.SetPixel(point.x,point.y,RGB(0,0,0));

      CDialog::OnLButtonDown(nFlags, point);
}
```

A-4: Resource Files

These are a collection of Header files needed by the program. They include header files for the C-files above, and also a resource header file.

Resource Header File

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by GroundStation.rc
//
#define IDM_ABOUTBOX                    0x0010
#define IDD_ABOUTBOX                    100
#define IDP_OLE_INIT_FAILED             100
#define IDS_ABOUTBOX                    101
#define IDD_GROUNDSTATION_DIALOG        102
#define IDP_SOCKETS_INIT_FAILED         103
#define IDR_MAINFRAME                   128
#define IDR_AUSUAVGS                    129
#define IDR_MENU1                       130
#define IDI_IconReceive                 133
#define IDI_IconSend                    134
#define IDI_IconOff                     135
#define IDI_Icon1                       136
#define IDD_SPEEDOMETER_DIALOG          137
#define IDD_ALTIMETER_DIALOG            138
#define IDD_COMPASS_DIALOG              139
#define IDD_MAP_DIALOG                  140
#define IDD_UAV_DIALOG                  141
#define IDD_TXRX_DIALOG                 142
#define IDC_MAP                         1000
#define IDC_RadioParityE                1001
#define IDC_RadioParityN                1002
#define IDC_BATTERY1POWER               1003
#define IDC_BATTERY2POWER               1004
#define IDC_BATTERY3POWER               1005
#define IDC_StaticBaud                  1006
#define IDC_SliderBaud                  1007
#define IDC_LATITUDE                    1008
#define IDC_XRate                       1008
#define IDC_LONGITUDE                   1009
#define IDC_YRate                       1009
#define IDC_TEMP1                       1010
#define IDC_XACC                        1010
#define IDC_YAW                         1011
#define IDC_ZRate                       1011
#define IDC_TEMP2                       1012
#define IDC_YACC                        1012
#define IDC_TEMP3                       1013
#define IDC_ZACC                        1013
#define IDC_CLIMBRATE                   1014
#define IDC_SYSTIME                     1014
#define IDC_RadioParityO                1015
#define IDC_AUSUAVTITLE                 1016
#define IDC_ComboCom                    1017
#define IDC_AIRSPEED                    1018
#define IDC_RadioData82                 1019
#define IDC_COMPASS                     1020
#define IDC_COMMANDINTERFACE            1021
#define IDC_MAPINTERFACE                1022
#define IDC_FONTLIST                    1025
#define IDC_RadioData81                 1026
#define IDC_EditTimeout                 1027
#define IDC_ALTIMETER                   1028
```

```
#define IDC_StaticTimeout               1029
#define IDC_StaticComport               1030
#define IDC_StaticNumber                1031
#define IDC_StaticFile                  1032
#define IDC_EditFile                    1033
#define IDC_ButtonFile                  1034
#define IDC_STARTBUTTON                 1035
#define IDC_ButtonSend                  1036
#define IDC_ButtonReceive               1037
#define IDC_CENTERBUTTON                1038
#define IDC_SIMBUTTON                   1039
#define IDC_WAYPOINT                    1040
#define IDC_GENERATEPATH                1041
#define IDC_RIGHTBUTTON                 1042
#define IDC_DOWNBUTTON                  1043
#define IDC_UPBUTTON                    1044
#define IDC_LEFTBUTTON                  1045
#define IDC_ZOOMINBUTTON                1046
#define IDC_ZOOMOUTBUTTON               1047
#define IDC_LOADMAPBUTTON               1048
#define IDC_STOPBUTTON                  1049
#define IDC_OPENCOM                     1050
#define IDC_SliderMemory                1051
#define IDC_StaticMemory                1052
#define IDC_ButtonExit                  1053
#define IDC_StaticSendLED               1054
#define IDC_StaticLEDReceive            1055
#define IDC_LabelComport                1056
#define IDC_HORIZON                     1057
#define IDC_MAH1                        1058
#define IDC_MAH2                        1059
#define IDC_MAH3                        1060
#define IDC_MAH4                        1061
#define IDC_3DM                         1062
#define IDC_ALTSELECT                   1063
#define IDC_ALTLEVEL                    1065
#define COLOR_BLACK                     2000
#define COLOR_DARKGREY                  2001
#define COLOR_GREY                      2002
#define COLOR_LIGHTGREY                 2003
#define COLOR_WHITE                     2004
#define COLOR_DARKRED                   2005
#define COLOR_RED                       2006
#define COLOR_LIGHTRED                  2007
#define COLOR_BRIGHTRED                 2008
#define COLOR_DARKBLUE                  2009
#define COLOR_BLUE                      2010
#define COLOR_LIGHTBLUE                 2011
#define COLOR_BRIGHTBLUE                2012
#define COLOR_DARKGREEN                 2013
#define COLOR_GREEN                     2014
#define COLOR_LIGHTGREEN                2015
#define COLOR_BRIGHTGREEN               2016
#define COLOR_DARKCYAN                  2017
#define COLOR_CYAN                      2018
#define COLOR_LIGHTCYAN                 2019
#define COLOR_BRIGHTCYAN                2020
#define COLOR_DARKMEGENTA               2021
#define COLOR_MEGENTA                   2022
#define COLOR_LIGHTMEGENTA              2023
#define COLOR_BRIGHTMEGENTA             2024
```

```
#define COLOR_DARKYELLOW                2025
#define COLOR_YELLOW                    2026
#define COLOR_LIGHTYELLOW               2027
#define COLOR_BRIGHTYELLOW              2028
#define IDT_CLOCK_TIMER                 5000
#define IDM_FILEOPEN                    32771
#define IDM_NEWFILE                     32772
#define IDM_EXIT                        32773
#define IDM_ABOUT                       32774

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        145
#define _APS_NEXT_COMMAND_VALUE         32775
#define _APS_NEXT_CONTROL_VALUE         1066
#define _APS_NEXT_SYMED_VALUE           105
#endif
#endif
```

Ground Station Dialog Header

```
// GroundStationDlg.h : header file
//
//{{AFX_INCLUDES()
#include "air.h"
//}}AFX_INCLUDES

#if
!defined(AFX_GROUNDSTATIONDLG_H__8A7BFAF8_A8E4_49CD_B0DD_21210BE9E93D
__INCLUDED_)
#define
AFX_GROUNDSTATIONDLG_H__8A7BFAF8_A8E4_49CD_B0DD_21210BE9E93D__INCLUDE
D_

#include "SpeedometerDlg.h"
#include "AltimeterDlg.h"
#include "CompassDlg.h"
#include "MapDlg.h"
#include "UavDlg.h"
#include "TxRxDlg.h"
#include "Air.h"

#include "math.h"
#include "string.h"
#include "stdio.h"
#include "STDDEF.h"
#include "STDLIB.h"

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CGroundStationDlgAutoProxy;
int CALLBACK EnumFontFamProc (LPENUMLOGFONT lpelf, LPNEWTEXTMETRIC
lpntm, DWORD nFontType, long lParam);
```

```
////////////////////////////////////////////////////////////////////////
////////
// CGroundStationDlg dialog

class CGroundStationDlg : public CDialog
{
	DECLARE_DYNAMIC(CGroundStationDlg);
	friend class CGroundStationDlgAutoProxy;

// Construction
public:
	int m_iCounter;
	CGroundStationDlg(CWnd* pParent = NULL);  // standard
constructor
	virtual ~CGroundStationDlg();

	// User Variables
	int m_iGpsIndex;
	double m_dPi;
	bool m_bGenPath;

	// Constants, statics, virtuals
	static const int m_iaZoomValues[64];
	static const COLORREF m_crColors[29];

	// Primitives
	int m_iZoomFactor;
	int m_iHeight;
	int m_iWidth;
	int m_iYPos;
	int m_iXPos;
	int m_iTrans;
	int m_irPlaneY2;
	int m_irPlaneY1;
	int m_irPlaneX2;
	int m_irPlaneX1;
	int m_iUpdateCount;
	int m_iAltSelLevel;
	double m_dSimValue;
	bool m_bMap;
	bool m_bPlaneSim;
	bool m_bCursor;
	double m_dAltimeter;
	double m_dAirSpeed;
	double m_dCompass;
	unsigned char m_cReceiveBuffer[140];
	double m_fReceiveBuffer[20];

	// Class Objects
	CBitmap m_bmpMap;
	CString m_strMapBmpFileName;
	CPROTO1Dlg(CWnd* pParent = NULL);   // standard constructor
	CFont m_fFontType;
	CString m_strlstFontList[200];

	// User Classes
	CSpeedometerDlg m_dlgSpeedometer;
	CAltimeterDlg m_dlgAltimeter;
	CCompassDlg m_dlgCompass;
	CMapDlg m_dlgMap;
```

```
        CUavDlg m_dlgUav;
        CTxRxDlg m_dlgTxRx;

        // funtions
        void InitChangeFontType();
        void FillFontList();
        void CreateSubWindows();

    // Dialog Data
        //{{AFX_DATA(CGroundStationDlg)
        enum { IDD = IDD_GROUNDSTATION_DIALOG };
        CProgressCtrl     m_pcAltLevel;
        CSliderCtrl m_scAltSelect;
        CEdit m_eTemp3;
        CEdit m_eTemp2;
        CEdit m_eTemp1;
        CEdit m_eClimbRate;
        CListBox    m_lbFontList;
        CEdit m_eYaw;
        CStatic     m_statMapInterface;
        CStatic     m_statMap;
        CEdit m_eLongitude;
        CEdit m_eLatitude;
        CStatic     m_statCompass;
        CStatic     m_statCommandInterface;
        CStatic     m_statAltimeter;
        CStatic     m_statAirSpeed;
        CProgressCtrl     m_pcBattery3Power;
        CProgressCtrl     m_pcBattery2Power;
        CProgressCtrl     m_pcBattery1Power;
        CStatic     m_statAusUavTitle;
        CString     m_strAirSpeed;
        CString     m_strCommandInterface;
        CString     m_strCompass;
        CString     m_strAltimeter;
        CString     m_strLatitude;
        CString     m_strLongitude;
        CString     m_strMap;
        CString     m_strMapInterface;
        CString     m_strYaw;
        CString     m_strAusUavTitle;
        CString     m_strFontList;
        CAir  m_aHorizon;
        double      m_dAHPitch;
        double      m_dAHRoll;
        CString     m_strClimbRate;
        CString     m_strTemp1;
        CString     m_strTemp2;
        CString     m_strTemp3;
        int         m_iAltSelect;
        float m_fXRate;
        float m_fYRate;
        float m_fZRate;
        float m_fSysTime;
        float m_fXAcc;
        float m_fYAcc;
        float m_fZAcc;
        //}}AFX_DATA

        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CGroundStationDlg)
```

```
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	//
DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	CGroundStationDlgAutoProxy* m_pAutoProxy;
	HICON m_hIcon;

	BOOL CanExit();

	// Generated message map functions
	//{{AFX_MSG(CGroundStationDlg)
	virtual BOOL OnInitDialog();
	virtual void OnOK();
	virtual void OnCancel();
	afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
	afx_msg void OnDestroy();
	afx_msg void OnPaint();
	afx_msg HCURSOR OnQueryDragIcon();
	afx_msg void OnClose();
	afx_msg void OnHelpAbout();
	afx_msg void OnExit();
	afx_msg void OnLoadmapbutton();
	afx_msg void OnUpbutton();
	afx_msg void OnCenterbutton();
	afx_msg void OnRightbutton();
	afx_msg void OnLeftbutton();
	afx_msg void OnDownbutton();
	afx_msg void OnZoominbutton();
	afx_msg void OnZoomoutbutton();
	afx_msg void OnSimbutton();
	afx_msg void OnTimer(UINT nIDEvent);
	afx_msg void OnWaypoint();
	afx_msg void OnGeneratepath();
	afx_msg BOOL OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT
message);
	afx_msg void OnOpencom();
	afx_msg void On3dm();
	afx_msg void OnReleasedcaptureAltselect(NMHDR* pNMHDR, LRESULT*
pResult);
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
immediately before the previous line.

#endif //
!defined(AFX_GROUNDSTATIONDLG_H__8A7BFAF8_A8E4_49CD_B0DD_21210BE9E93D
__INCLUDED_)
```

Map Dialog Header

```
#if
!defined(AFX_MAPDLG_H__4E97A1E0_7F4C_4C8B_B52C_B60D033BB0B4__INCLUDED
_)
#define AFX_MAPDLG_H__4E97A1E0_7F4C_4C8B_B52C_B60D033BB0B4__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// MapDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////
////////
// CMapDlg dialog

class CMapDlg : public CDialog
{
// Construction
public:
	CMapDlg(CWnd* pParent = NULL);   // standard constructor
	int m_iWayPoints[3][20];
	int m_iSetPoints[3][20];
	int m_iGpsCoor[5][50000];
	int m_iGpsIndex;
	int m_iWayPointIndex;
	int m_iSetPointIndex;

// Dialog Data
	//{{AFX_DATA(CMapDlg)
	enum { IDD = IDD_MAP_DIALOG };
		// NOTE: the ClassWizard will add data members here
	//}}AFX_DATA


// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CMapDlg)
	public:
	virtual void OnFinalRelease();
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
	//}}AFX_VIRTUAL

// Implementation
protected:

	// Generated message map functions
	//{{AFX_MSG(CMapDlg)
		// NOTE: the ClassWizard will add member functions here
	afx_msg void OnPaint();
	afx_msg BOOL OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT
message);
	afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
	// Generated OLE dispatch map functions
```

```
      //{{AFX_DISPATCH(CMapDlg)
            // NOTE - the ClassWizard will add and remove member
functions here.
      //}}AFX_DISPATCH
      DECLARE_DISPATCH_MAP()
      DECLARE_INTERFACE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
immediately before the previous line.

#endif //
!defined(AFX_MAPDLG_H__4E97A1E0_7F4C_4C8B_B52C_B60D033BB0B4__INCLUDED
_)
```

Altimeter Station Dialog Header

```
#if
!defined(AFX_ALTIMETERDLG_H__890EAF16_E475_40A9_9F27_B71B5C132E74__IN
CLUDED_)
#define
AFX_ALTIMETERDLG_H__890EAF16_E475_40A9_9F27_B71B5C132E74__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// AltimeterDlg.h : header file
//

/////////////////////////////////////////////////////////////////////
////////
// CAltimeterDlg dialog

class CAltimeterDlg : public CDialog
{
// Construction
public:
      CAltimeterDlg(CWnd* pParent = NULL);   // standard constructor
      bool m_bRepaint;

// Dialog Data
      //{{AFX_DATA(CAltimeterDlg)
      enum { IDD = IDD_ALTIMETER_DIALOG };
            // NOTE: the ClassWizard will add data members here
      //}}AFX_DATA


// Overrides
      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CAltimeterDlg)
      public:
      virtual void OnFinalRelease();
      protected:
      virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
      //}}AFX_VIRTUAL
```

```
// Implementation
protected:

	// Generated message map functions
	//{{AFX_MSG(CAltimeterDlg)
		// NOTE: the ClassWizard will add member functions here
	afx_msg void OnPaint();
	afx_msg BOOL OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT
message);
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
	// Generated OLE dispatch map functions
	//{{AFX_DISPATCH(CAltimeterDlg)
		// NOTE - the ClassWizard will add and remove member
functions here.
	//}}AFX_DISPATCH
	DECLARE_DISPATCH_MAP()
	DECLARE_INTERFACE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
immediately before the previous line.

#endif //
!defined(AFX_ALTIMETERDLG_H__890EAF16_E475_40A9_9F27_B71B5C132E74__IN
CLUDED_)
```

Compass Dialog Header

```
#if
!defined(AFX_COMPASSDLG_H__D540DCF1_75EF_4D72_9AF7_40A948DD9366__INCL
UDED_)
#define
AFX_COMPASSDLG_H__D540DCF1_75EF_4D72_9AF7_40A948DD9366__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// CompassDlg.h : header file
//

/////////////////////////////////////////////////////////////////////
////////
// CCompassDlg dialog

class CCompassDlg : public CDialog
{
// Construction
public:
	CCompassDlg(CWnd* pParent = NULL);   // standard constructor
	bool m_bRepaint;

// Dialog Data
	//{{AFX_DATA(CCompassDlg)
	enum { IDD = IDD_COMPASS_DIALOG };
```

```
            // NOTE: the ClassWizard will add data members here
      //}}AFX_DATA


// Overrides
      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CCompassDlg)
      public:
      virtual void OnFinalRelease();
      protected:
      virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
      //}}AFX_VIRTUAL

// Implementation
protected:

      // Generated message map functions
      //{{AFX_MSG(CCompassDlg)
            // NOTE: the ClassWizard will add member functions here
      afx_msg void OnPaint();
      afx_msg BOOL OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT
message);
      //}}AFX_MSG
      DECLARE_MESSAGE_MAP()
      // Generated OLE dispatch map functions
      //{{AFX_DISPATCH(CCompassDlg)
            // NOTE - the ClassWizard will add and remove member
functions here.
      //}}AFX_DISPATCH
      DECLARE_DISPATCH_MAP()
      DECLARE_INTERFACE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
immediately before the previous line.

#endif //
!defined(AFX_COMPASSDLG_H__D540DCF1_75EF_4D72_9AF7_40A948DD9366__INCL
UDED_)
```

Speedometer Dialog Header

```
#if
!defined(AFX_SPEEDOMETERDLG_H__00B44E32_BA80_4CC4_B45A_A6BAF585683B__
INCLUDED_)
#define
AFX_SPEEDOMETERDLG_H__00B44E32_BA80_4CC4_B45A_A6BAF585683B__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// SpeedometerDlg.h : header file
//
```

```
/////////////////////////////////////////////////////////////////////////
////////
// CSpeedometerDlg dialog

class CSpeedometerDlg : public CDialog
{
// Construction
public:
	CSpeedometerDlg(CWnd* pParent = NULL);	// standard
constructor
	bool m_bRepaint;

// Dialog Data
	//{{AFX_DATA(CSpeedometerDlg)
	enum { IDD = IDD_SPEEDOMETER_DIALOG };
		// NOTE: the ClassWizard will add data members here
	//}}AFX_DATA


// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CSpeedometerDlg)
	public:
	virtual void OnFinalRelease();
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV
support
	//}}AFX_VIRTUAL

// Implementation
protected:

	// Generated message map functions
	//{{AFX_MSG(CSpeedometerDlg)
		// NOTE: the ClassWizard will add member functions here
	afx_msg void OnPaint();
	afx_msg BOOL OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT
message);
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
	// Generated OLE dispatch map functions
	//{{AFX_DISPATCH(CSpeedometerDlg)
		// NOTE - the ClassWizard will add and remove member
functions here.
	//}}AFX_DISPATCH
	DECLARE_DISPATCH_MAP()
	DECLARE_INTERFACE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
immediately before the previous line.

#endif //
!defined(AFX_SPEEDOMETERDLG_H__00B44E32_BA80_4CC4_B45A_A6BAF585683B__
INCLUDED_)
```

APPENDIX B

EMBEDDED SYSTEM CODE

B-1: DFC Code

```
/****   Include  the following files ***/
#include <mc9s12dp256.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include <stddef.h>
#include <time.h>
#include <C:\Code\UAVFin5\mc9s12dg256_int_vec.c>
#include <C:\Code\UAVFin5\function_library.h>

// Define The Transmmit buffer
#define  CAN0TXFG0   _P(0x170)
#define  CAN0TXFG1   _P(0x171)
#define  CAN0TXFG2   _P(0x172)
#define  CAN0TXFG3   _P(0x173)
#define  CAN0TXFG4   _P(0x174)
#define  CAN0TXFG5   _P(0x175)
#define  CAN0TXFG6   _P(0x176)
#define  CAN0TXFG7   _P(0x177)
#define  CAN0TXFG8   _P(0x178)
#define  CAN0TXFG9   _P(0x179)
#define  CAN0TXFGA   _P(0x17A)
#define  CAN0TXFGB   _P(0x17B)
#define  CAN0TXFGC   _P(0x17C)
#define  CAN0TXFGD   _P(0x17D)

// Timer associated variables
unsigned int counter = 0;
unsigned int sci_counter = 0;
unsigned int serial_flag = 0;
char sci_output[256], sci_input[256];

// Global parameter values
float GP2,AP2,XR2,YR2,ZR2,XA2,YA2,ZA2,XT2,YT2,AOA2,AOS2; // Current
param values
float compass, loncurr, latcurr, currtime; // Current parameter
values

// These are the memory address of where the strings are stored
unsigned char *addr_GP1, *addr_GP2, *addr_GP3, *addr_GP4;
unsigned char *addr_compass1, *addr_compass2, *addr_compass3,
*addr_compass4;
unsigned char *addr_loncurr1, *addr_loncurr2, *addr_loncurr3,
*addr_loncurr4;
unsigned char *addr_latcurr1, *addr_latcurr2, *addr_latcurr3,
*addr_latcurr4;

// Some constants such as base location in latitude and longitude,
and pi
const float latbase = 2548.5155;
const float lonbase = 5448.5125;
const float clockwiseturn = 10.0;
const float anticlockwiseturn = -10.0;
const float noturn = 0.0;
const float m_pi = 3.14159;   // PI
```

```
// Servo motor variables
float M0z,M1z,M2z,M3z;  // Servo Motor offset and trims
float M0s,M1s,M2s,M3s;  // Servo Motor gains


void main(void)
{


   /***** INITIALIZING TIME AND COUNTER VARIABLES ************/
   // Parameter values
   unsigned int GP,AP,XR,YR,ZR,XA,YA,ZA,XT,YT,AOA,AOS;  // Sensor sum
of 8
   unsigned int GP1,AP1,XR1,YR1,ZR1,XA1,YA1,ZA1,XT1,YT1,AOA1,AOS1; //
Sensor avg
   unsigned int alttemp[10], i, j;
   signed int lontemp[10], lattemp[10];
   char *tempstr, tempchar, temparray[15];
   int *status;

   // Trajectory tracker variables
   unsigned int reqdist, radius, hyp1, hyp2, waypointcounter;
   signed int xc1diff, yc1diff, xc2diff, yc2diff, diffxpos, diffypos,
xc2, yc2;
   signed int deslat, deslon, lat, lon, xc1, yc1;
   float dirXvec, dirYvec, dirX1vec, dirY1vec, dirX2vec, dirY2vec,
crsprod;
   float difflat, difflon, heading, angle;

   // PID lateral control
   float Ra, Rt, Rr, Re, Ka, Kt, Kr, Ke, Ea, Et, Ee, Er;


   // Discrete LQR State Feedback Longitudinal Control
float K[2][5], Nbar[2][2], KdX[2], NRef[2]; // Statefeedback and
refence gain
   float xinc[5], dX[5], Ref; // Initial State and difference in
states
   float wvel, prev_alt, prevtime;   // Longitudinal autopilot
variables


   // Initialize the digital output ports
   DDRA = 0xFF;  // Port A represent the direction of turn
   DDRB = 0xFF;  // Port B represent the decision to turn
   PORTA = 0x00; // Initially set Port A to CW turn
   PORTB = 0x00; // Initially set Port B to no turn

   serial_flag = 0; // Reset serial flag

   // Servo Motor offsets trims
   M0z=38.0;             // Aileron
   M1z=38.0;             // Throttle
   M2z=38.0;             // Rudder
   M3z=38.0;             // Elevator

   // Servo Motor gains
   M0s=0.25;             // Aileron
   M1s=0.25;             // Throttle
   M2s=0.25;             // Rudder
   M3s=0.25;             // Elevator
```

```
   // Proportional Gains
   Ka=0.25;       // Roll Controller
   Kt=0.25;             // Throttle
   Kr=0.25;             // Yaw Damper
   Ke=0.25;             // Pitch Controller

   // Proportional Reference
   Ra=0.0;              // Roll Controller
   Rt=0.0;              // Throttle?
   Rr=0.0;              // Yaw Damper
   Re=0.0;              // Pitch Controller

   // Set all the errors to zero
   Ea = 0.0;
   Et = 0.0;
   Ee = 0.0;
   Er = 0.0;

      // Initialize the K feedback gains for the longitudinal
controller
      K[0][0] = -0.0016;
      K[0][1] = 0.0012;
      K[0][2] = -0.0032;
      K[0][3] = -0.1090;
      K[0][4] = -0.0018;
      K[1][0] = 0.0084;
      K[1][1] = -0.0003;
      K[1][2] = 0.0005;
      K[1][3] = 0.0209;
      K[1][4] = 0.0017;

      // Initialize the Nbar gain for the longitudinal controller
      Nbar[0][0] = 0.0112;
      Nbar[0][1] = -0.0018;
      Nbar[1][0] = 0.0115;
      Nbar[1][1] = 0.0017;

      // Initialize the aircraft initial conditions
      xinc[0] = 20.0;     // speed m/s
      xinc[1] = 0.0;      // up/down speed m/s
      xinc[2] = 0.0;      // pitch rate rads/s
      xinc[3] = 0.0;      // pitch rads
      xinc[4] = 100;      // altitude m

      // Set all the initial state errors to zero
      dX[0] = 20.0;       // speed m/s
      dX[1] = 0.0;        // up/down speed m/s
      dX[2] = 0.0;        // pitch rate rads/s
      dX[3] = 0.0;        // pitch rads
      dX[4] = 100;        // altitude m


   // Center all servos
   motor (0, 0);
   motor (1, 5);
   motor (2, 0);
   motor (3, 0);
```

```
addr_GP1 = (unsigned char*)&GP2+0;
addr_GP2 = (unsigned char*)&GP2+1;
addr_GP3 = (unsigned char*)&GP2+2;
addr_GP4 = (unsigned char*)&GP2+3;

addr_compass1 = (unsigned char*)&compass+0;
addr_compass2 = (unsigned char*)&compass+1;
addr_compass3 = (unsigned char*)&compass+2;
addr_compass4 = (unsigned char*)&compass+3;

addr_latcurr1 = (unsigned char*)&latcurr+0;
addr_latcurr2 = (unsigned char*)&latcurr+1;
addr_latcurr3 = (unsigned char*)&latcurr+2;
addr_latcurr4 = (unsigned char*)&latcurr+3;

addr_loncurr1 = (unsigned char*)&loncurr+0;
addr_loncurr2 = (unsigned char*)&loncurr+1;
addr_loncurr3 = (unsigned char*)&loncurr+2;
addr_loncurr4 = (unsigned char*)&loncurr+3;



// Hardcoded list of waypoints
lattemp[0] = 1200;
lontemp[0] = 0;
alttemp[0] = 100;


lattemp[1] = 600;
lontemp[1] = 600;
alttemp[1] = 110;


lattemp[2] = 1200;
lontemp[2] = 1200;
alttemp[2] = 100;


lattemp[3] = 600;
lontemp[3] = 600;
alttemp[3] = 90;

lattemp[4] = 0;
lontemp[4] = 0;
alttemp[4] = 100;


clc_setup();  // Increase the bus clock to 24 MHz
sci_setup();  // Set the max possible baud rate of 115200 baud
adc_setup();
inc_setup();
pwm_setup();
rti_setup();
can_setup(1);
asm("cli");

waypointcounter = 0;
prevtime = time_sec();
prev_alt = 100;
Ref = 0.0;
```

```
   while (1)
   {

//=================================================================
=========//
//                                                  PART ONE
//=================================================================
=========//

        // First get all the required analog values
        // Get the Height
      ATD0CTL5 = 0x82;
      while ((ATD0STAT0 & 0x80) == 0);
        AP = 0;
        AP = AP + ((256*ATD0DR0H) + ATD0DR0L);
        AP = AP + ((256*ATD0DR1H) + ATD0DR1L);
        AP = AP + ((256*ATD0DR2H) + ATD0DR2L);
        AP = AP + ((256*ATD0DR3H) + ATD0DR3L);
        AP = AP + ((256*ATD0DR4H) + ATD0DR4L);
        AP = AP + ((256*ATD0DR5H) + ATD0DR5L);
        AP = AP + ((256*ATD0DR6H) + ATD0DR6L);
        AP = AP + ((256*ATD0DR7H) + ATD0DR7L);

        // Get the Roll Rate
      ATD0CTL5 = 0x83;
      while ((ATD0STAT0 & 0x80) == 0);
        XR = 0;
        XR = XR + ((256*ATD0DR0H) + ATD0DR0L);
        XR = XR + ((256*ATD0DR1H) + ATD0DR1L);
        XR = XR + ((256*ATD0DR2H) + ATD0DR2L);
        XR = XR + ((256*ATD0DR3H) + ATD0DR3L);
        XR = XR + ((256*ATD0DR4H) + ATD0DR4L);
        XR = XR + ((256*ATD0DR5H) + ATD0DR5L);
        XR = XR + ((256*ATD0DR6H) + ATD0DR6L);
        XR = XR + ((256*ATD0DR7H) + ATD0DR7L);

        // Get the Pitch Rate
      ATD0CTL5 = 0x84;
      while ((ATD0STAT0 & 0x80) == 0);
        YR = 0;
        YR = YR + ((256*ATD0DR0H) + ATD0DR0L);
        YR = YR + ((256*ATD0DR1H) + ATD0DR1L);
        YR = YR + ((256*ATD0DR2H) + ATD0DR2L);
        YR = YR + ((256*ATD0DR3H) + ATD0DR3L);
        YR = YR + ((256*ATD0DR4H) + ATD0DR4L);
        YR = YR + ((256*ATD0DR5H) + ATD0DR5L);
        YR = YR + ((256*ATD0DR6H) + ATD0DR6L);
        YR = YR + ((256*ATD0DR7H) + ATD0DR7L);

        // Get the Yaw Rate
      ATD0CTL5 = 0x85;
      while ((ATD0STAT0 & 0x80) == 0);
        ZR = 0;
        ZR = ZR + ((256*ATD0DR0H) + ATD0DR0L);
        ZR = ZR + ((256*ATD0DR1H) + ATD0DR1L);
        ZR = ZR + ((256*ATD0DR2H) + ATD0DR2L);
        ZR = ZR + ((256*ATD0DR3H) + ATD0DR3L);
        ZR = ZR + ((256*ATD0DR4H) + ATD0DR4L);
        ZR = ZR + ((256*ATD0DR5H) + ATD0DR5L);
        ZR = ZR + ((256*ATD0DR6H) + ATD0DR6L);
        ZR = ZR + ((256*ATD0DR7H) + ATD0DR7L);
```

```
  // Get the Roll
ATD0CTL5 = 0x86;
while ((ATD0STAT0 & 0x80) == 0);
  XT = 0;
  XT = XT + ((256*ATD0DR0H) + ATD0DR0L);
  XT = XT + ((256*ATD0DR1H) + ATD0DR1L);
  XT = XT + ((256*ATD0DR2H) + ATD0DR2L);
  XT = XT + ((256*ATD0DR3H) + ATD0DR3L);
  XT = XT + ((256*ATD0DR4H) + ATD0DR4L);
  XT = XT + ((256*ATD0DR5H) + ATD0DR5L);
  XT = XT + ((256*ATD0DR6H) + ATD0DR6L);
  XT = XT + ((256*ATD0DR7H) + ATD0DR7L);

  // Get the Pitch
ATD0CTL5 = 0x87;
while ((ATD0STAT0 & 0x80) == 0);
  YT = 0;
  YT = YT + ((256*ATD0DR0H) + ATD0DR0L);
  YT = YT + ((256*ATD0DR1H) + ATD0DR1L);
  YT = YT + ((256*ATD0DR2H) + ATD0DR2L);
  YT = YT + ((256*ATD0DR3H) + ATD0DR3L);
  YT = YT + ((256*ATD0DR4H) + ATD0DR4L);
  YT = YT + ((256*ATD0DR5H) + ATD0DR5L);
  YT = YT + ((256*ATD0DR6H) + ATD0DR6L);
  YT = YT + ((256*ATD0DR7H) + ATD0DR7L);

  // Get the Y Acceleration
ATD1CTL5 = 0x80;
while ((ATD1STAT0 & 0x80) == 0);
  YA = 0;
  YA = YA + ((256*ATD1DR0H) + ATD1DR0L);
  YA = YA + ((256*ATD1DR1H) + ATD1DR1L);
  YA = YA + ((256*ATD1DR2H) + ATD1DR2L);
  YA = YA + ((256*ATD1DR3H) + ATD1DR3L);
  YA = YA + ((256*ATD1DR4H) + ATD1DR4L);
  YA = YA + ((256*ATD1DR5H) + ATD1DR5L);
  YA = YA + ((256*ATD1DR6H) + ATD1DR6L);
  YA = YA + ((256*ATD1DR7H) + ATD1DR7L);

  // Get the X Acceleration
ATD1CTL5 = 0x81;
while ((ATD1STAT0 & 0x80) == 0);
  XA = 0;
  XA = XA + ((256*ATD1DR0H) + ATD1DR0L);
  XA = XA + ((256*ATD1DR1H) + ATD1DR1L);
  XA = XA + ((256*ATD1DR2H) + ATD1DR2L);
  XA = XA + ((256*ATD1DR3H) + ATD1DR3L);
  XA = XA + ((256*ATD1DR4H) + ATD1DR4L);
  XA = XA + ((256*ATD1DR5H) + ATD1DR5L);
  XA = XA + ((256*ATD1DR6H) + ATD1DR6L);
  XA = XA + ((256*ATD1DR7H) + ATD1DR7L);

  // Get the Z Acceleration
ATD1CTL5 = 0x82;
while ((ATD1STAT0 & 0x80) == 0);
  ZA = 0;
  ZA = ZA + ((256*ATD1DR0H) + ATD1DR0L);
  ZA = ZA + ((256*ATD1DR1H) + ATD1DR1L);
  ZA = ZA + ((256*ATD1DR2H) + ATD1DR2L);
  ZA = ZA + ((256*ATD1DR3H) + ATD1DR3L);
```

```
  ZA = ZA + ((256*ATD1DR4H) + ATD1DR4L);
  ZA = ZA + ((256*ATD1DR5H) + ATD1DR5L);
  ZA = ZA + ((256*ATD1DR6H) + ATD1DR6L);
  ZA = ZA + ((256*ATD1DR7H) + ATD1DR7L);

  // Get the Speed
ATD1CTL5 = 0x83;
while ((ATD1STAT0 & 0x80) == 0);
  GP = 0;
  GP = GP + ((256*ATD1DR0H) + ATD1DR0L);
  GP = GP + ((256*ATD1DR1H) + ATD1DR1L);
  GP = GP + ((256*ATD1DR2H) + ATD1DR2L);
  GP = GP + ((256*ATD1DR3H) + ATD1DR3L);
  GP = GP + ((256*ATD1DR4H) + ATD1DR4L);
  GP = GP + ((256*ATD1DR5H) + ATD1DR5L);
  GP = GP + ((256*ATD1DR6H) + ATD1DR6L);
  GP = GP + ((256*ATD1DR7H) + ATD1DR7L);

  // Get the Angle of Attack
ATD1CTL5 = 0x84;
while ((ATD1STAT0 & 0x80) == 0);
  AOA = 0;
  AOA = AOA + ((256*ATD1DR0H) + ATD1DR0L);
  AOA = AOA + ((256*ATD1DR1H) + ATD1DR1L);
  AOA = AOA + ((256*ATD1DR2H) + ATD1DR2L);
  AOA = AOA + ((256*ATD1DR3H) + ATD1DR3L);
  AOA = AOA + ((256*ATD1DR4H) + ATD1DR4L);
  AOA = AOA + ((256*ATD1DR5H) + ATD1DR5L);
  AOA = AOA + ((256*ATD1DR6H) + ATD1DR6L);
  AOA = AOA + ((256*ATD1DR7H) + ATD1DR7L);

  // Get the Angle of Side Slip
ATD1CTL5 = 0x85;
while ((ATD1STAT0 & 0x80) == 0);
  AOS = 0;
  AOS = AOS + ((256*ATD1DR0H) + ATD1DR0L);
  AOS = AOS + ((256*ATD1DR1H) + ATD1DR1L);
  AOS = AOS + ((256*ATD1DR2H) + ATD1DR2L);
  AOS = AOS + ((256*ATD1DR3H) + ATD1DR3L);
  AOS = AOS + ((256*ATD1DR4H) + ATD1DR4L);
  AOS = AOS + ((256*ATD1DR5H) + ATD1DR5L);
  AOS = AOS + ((256*ATD1DR6H) + ATD1DR6L);
  AOS = AOS + ((256*ATD1DR7H) + ATD1DR7L);

  // Average 8 values of each parameter to reduce noise
AP1 = AP >> 3;
XR1 = XR >> 3;
YR1 = YR >> 3;
ZR1 = ZR >> 3;
XT1 = XT >> 3;
YT1 = YT >> 3;
YA1 = YA >> 3;
XA1 = XA >> 3;
ZA1 = ZA >> 3;
GP1 = GP >> 3;
AOA1 = GP >> 3;
AOS1 = GP >> 3;
```

```
        // Calculate parameter using formula obtain via Avionics Unit
Calibaration
//      V = -0.002077 · h + 3
        AP2 = (3.0 - ((float)AP1 * 5.0/1023))/0.002077;
//      XR2 = (-76.15     * ((float)XR1 * 5.0/1023.0)) + 184.79;
        XR2 = (-76.15     * ((float)XR1 * 5.0/1023.0)) + 182.29;
//      YR2 = (-78.516    * ((float)YR1 * 5.0/1023.0)) + 191.56;
        YR2 = (-78.516    * ((float)YR1 * 5.0/1023.0)) + 190.00;
//      ZR2 = (77.054     * ((float)ZR1 * 5.0/1023.0)) - 174.84;
        ZR2 = (77.054     * ((float)ZR1 * 5.0/1023.0)) - 172.54;
        XT2 = (-29.10203  * ((float)XT1 * 5.0/1023.0)) + 68.5342;
//      YT2 = (-34.426    * ((float)YT1 * 5.0/1023.0)) +  67.767;
        YT2 = (-34.426    * ((float)YT1 * 5.0/1023.0)) +  73.467;
        YA2 = (2.768166   * ((float)YA1 * 5.0/1023.0)) - 6.65;
        XA2 = (2.806295   * ((float)XA1 * 5.0/1023.0)) - 7.3581;
//      ZA2 = (2.781436   * ((float)ZA1 * 5.0/1023.0)) - 6.82724;
        ZA2 = (2.781436   * ((float)ZA1 * 5.0/1023.0)) - 7.12724;
//      GP2 = (1.8 - ((float)GP1 * 5.0/1023)) * 23/1.4;
        GP2 = sqrt((1.72 - ((float)GP1 * 5.0/1023)) *
15.0*15.0/0.88);
        AOA2 = (((float)AOA1 * 5.0/1023) - 2.5)*360.0/5.0;
        AOS2 = (((float)AOS1 * 5.0/1023) - 2.5)*360.0/5.0;


//=================================================================
=========//
//                                                PART TWO
//=================================================================
=========//

      // NextGet the serial inputs.  GPS and Compass are serial
        i = 0;
        while ((tempchar = getchar(1)) != '$'); // Wait for a dollar
sign
                                                                  //
to indicate the start of a line

        // Capture 127 characters from the line.  127 characters are
sent by HILS
        // Line in NMEA format
        while (i < 85)
        {
         sci_input[i] = getchar(0);
             i++;
        }
        sci_input[i] = '\n';
        sci_input[i+2] = '\0';

        // Find the Heading
        tempstr = strstr(sci_input, "HPR,");
        i = 4;
        j = 0;
        while (*(tempstr+i) != ',')
        {
         temparray[j] = *(tempstr+i);
             i++;
             j++;
        }
        temparray[j] = '\0';
        compass = atof(temparray);
```

```
//=================================================================
=========//
//                                                PART THREE
//=================================================================
=========//

      // Calculate current position based on offset from a reference
position
        difflat = latcurr - latbase;  // Find offset latitude
        difflon = loncurr - lonbase;  // Find offset longitude

        // Calculate in meters the x,y position of the UAV
        lat = (signed int)(difflat*60.0*30.0); //Calc the current
offset xpos in m
        lon = (signed int)(difflon*60.0*27.0); //Calc the current
offset ypos in m

       // TRAJECTORY CONTROLLER.  THIS PORTION OF THE CODE IS THE
ALGORITHM OF THE
       // TRAJECTORY CONTROLLER.  IT USES FEEDBACK GUIDANCE STRATEGY
THAT GIVES US
       // MINIMUM PATH

       // Calculate the remaining distance to the next waypoint

       // Get the next waypoint in X (Lat) and Y (Lon) and Z (Alt)
       deslat = lattemp[waypointcounter];
       deslon = lontemp[waypointcounter];

      // Find the difference in lat and lon between our current and
desired pos
        diffxpos = deslat - lat;
        diffypos = deslon - lon;

        // Find the distance in meters between the current pos and
desired pos
reqdist = (unsigned
int)sqrt(pow((float)diffxpos,2.0)+pow((float)diffypos,2.0));


      // Calculate the radius of the circle that the UAV can fly for
a given
        // speed and bank (roll) angle
        radius = (unsigned
int)(pow(GP2,2)/(9.81*tan(fabs(XT2)*m_pi/180.0)));
        if (radius > 255)
             radius = 255;


      // Find the BEARING to the next waypoint from our current
location
        heading = 180*atan2 ((float)diffypos, (float)diffxpos)/m_pi;
        if (heading < 0.0)
         heading = 360.0 + heading;


      // Find the difference between our current and the required
heading
      angle = fabs(compass - heading);
```

```
      if (angle < 8  || angle > 352)
        {
         Ref = (float)alttemp[waypointcounter] - xinc[4];
        }

      // Update waypoints if we reach to within 50 meter of the
desired waypoint
        if (reqdist <= 50)
         waypointcounter++;

      if (waypointcounter >= 5)
         waypointcounter = 0;


      // Calculate current direction vector
      dirXvec = cos(compass*m_pi/180.0);
      dirYvec = sin(compass*m_pi/180.0);

      // Calculate the two direction vectors for the two different
circles
      dirX1vec = dirYvec;
      dirY1vec = -dirXvec;

      dirX2vec = -dirYvec;
      dirY2vec = dirXvec;

      // Calculate the center of the two circles
      xc1 = lat + (signed int)((float)radius*dirX1vec);
      yc1 = lon + (signed int)((float)radius*dirY1vec);

      xc2 = lat + (signed int)((float)radius*dirX2vec);
      yc2 = lon + (signed int)((float)radius*dirY2vec);

      // Calculate distance from center of two circles to next
waypoint
      xc1diff = deslat - xc1;
      yc1diff = deslon - yc1;
  hyp1 = (unsigned int)sqrt (pow((float)xc1diff,2.0) +
pow((float)yc1diff,2.0));

      xc2diff = (float)(deslat - xc2);
      yc2diff = (float)(deslon - yc2);
  hyp2 = (unsigned int)sqrt (pow((float)xc2diff,2.0) +
pow((float)yc2diff,2.0));

      if (hyp1 < hyp2)
      {
           if (hyp1 >= radius)
                crsprod = (dirXvec*dirY1vec) - (dirX1vec*dirYvec);
             else
                crsprod = (dirXvec*dirY2vec) - (dirX2vec*dirYvec);
      }
        else
        {
           if (hyp2 >= radius)
                crsprod = (dirXvec*dirY2vec) - (dirX2vec*dirYvec);
             else
                crsprod = (dirXvec*dirY1vec) - (dirX1vec*dirYvec);
      }
```

```
        if (crsprod < 0)
      {
           Ra = anticlockwiseturn;
         PORTA = 0xFF;
        }
      else
      {
         Ra = clockwiseturn;
         PORTA = 0x00;
        }

        // If the heading error is more than 10 degrees, then we turn
      if (angle > 10  && angle < 350)
      {
         PORTB = 0xFF;
        }
      else
      {
//          Ra = noturn;
         Ra = heading - compass;
             if (Ra >= 350.0)
                  Ra = Ra - 360.0;
             if (Ra <= -350.0)
                  Ra = Ra + 360.0;
             PORTB = 0x00;
      }

      Ea = Ka*(Ra - XT2) - 0.1*XR2;
        Er = Kr*(0 - ZR2);
        motor (0, Ea);
        motor (2, Er);


//=================================================================
=========//
//                                                  PART FOUR
//=================================================================
=========//
      // Autopilot calculations
        // Find the difference in states

      currtime = time_sec();
      wvel = GP2*sin(YT2*m_pi/180.0) - ((AP2-prev_alt)/(currtime -
prevtime));
        wvel = wvel/cos(YT2*m_pi/180.0);
        dX[0] = GP2 - xinc[0];
        dX[1] = wvel - xinc[1];
        dX[2] = YR2*m_pi/180.0 - xinc[2];
        dX[3] = YT2*m_pi/180.0 - xinc[3];
        dX[4] = AP2 - xinc[4];

      KdX[0] = K[0][0]*dX[0] + K[0][1]*dX[1] + K[0][2]*dX[2] +
K[0][3]*dX[3] + K[0][4]*dX[4];
      KdX[1] = K[1][0]*dX[0] + K[1][1]*dX[1] + K[1][2]*dX[2] +
K[1][3]*dX[3] + K[1][4]*dX[4];
//      Ref = 0.0;
        NRef[0] = Nbar[0][0] * 0.0 + Nbar[0][1] * Ref;
        NRef[1] = Nbar[1][0] * 0.0 + Nbar[1][1] * Ref;

        Ee = (NRef[0] - KdX[0])*180.0/m_pi;
        Et = (NRef[1] - KdX[1])*180.0/m_pi + 12;
```

```
        motor (1, Et);
      motor (3, Ee);

        prevtime = currtime;
        prev_alt = AP2;


//=====================================================================
=========//
//                                                  PART FIVE
//=====================================================================
=========//

      sci_counter = 0;
        store_buffer ("@");

      strcpy(temparray,ftoa(compass, status));
      i = 0;
      while(temparray[i] != '.')
      {
         i++;
      }
      temparray[i+2] = '\0';

      strcpy(temparray,ftoa(XT2, status));
      i = 0;
      while(temparray[i] != '.')
      {
         i++;
      }
      temparray[i+2] = '\0';

      strcpy(temparray,ftoa(YT2, status));
      i = 0;
      while(temparray[i] != '.')
      {
         i++;
      }
      temparray[i+2] = '\0';

      strcpy(temparray,ftoa(GP2, status));
      i = 0;
      while(temparray[i] != '.')
      {
         i++;
      }
      temparray[i+2] = '\0';

      strcpy(temparray,ftoa(AP2, status));
      i = 0;
      while(temparray[i] != '.')
      {
         i++;
      }
      temparray[i+2] = '\0';

      strcpy(temparray,ftoa((float)lat, status));
      i = 0;
      while(temparray[i] != '.')
```

```
      {
         i++;
      }
      temparray[i+2] = '\0';

      strcpy(temparray,ftoa((float)lon, status));
      i = 0;
      while(temparray[i] != '.')
      {
         i++;
      }
      temparray[i+2] = '\0';

        store_buffer ("*");
        printstring(sci_output, 0);

   } // end of main while loop

} //end of main


extern void _incO_isr(void)
{
        TFLG2 = 0x80;
        counter++;
      //printchar('P');
}


extern void _sci0_isr(void)
{
   unsigned char flagstore, datastore;
   flagstore = SCI0SR1;
   datastore = SCI0DRL;

   if (datastore == '#')
      serial_flag = 1;  // set serial flag if serial transmission
requested by
                                    // Ground Station
}


extern void _rti_isr(void)
{
      // STEP 1: SELECT TRANSMIT BUFFER
      // CANxTBSEL
      // TX2 - TX0 : Selects the approprate CAN Transmit Buffer
      CAN0TBSEL = 0x01;    // Make one of the three Tx buffers
available

      // STEP 2: SET UP THE TRANSMIT MESSAGE FRAME
      CAN0TXFG0 = 'G'; // IDR = Identification
      CAN0TXFG1 = 0xFF; // IDR = Identification
      CAN0TXFG2 = 'N'; // IDR = Identification
      CAN0TXFG3 = 0xFE; // IDR = Identification

      CAN0TXFG4 = *addr_GP1;  // DSR = Data byte 1
      CAN0TXFG5 = *addr_GP2;  // DSR = Data byte 2
      CAN0TXFG6 = *addr_GP3;  // DSR = Data byte 3
      CAN0TXFG7 = *addr_GP4;  // DSR = Data byte 4
      CAN0TXFG8 = *addr_compass1;   // DSR = Data byte 5
```

```
      CAN0TXFG9 = *addr_compass2;   // DSR = Data byte 6
      CAN0TXFGA = *addr_compass3;   // DSR = Data byte 7
      CAN0TXFGB = *addr_compass4;   // DSR = Data byte 8

      CAN0TXFGC = 0x08; // DLR = Data Length (set to 8 for 8 bytes of
data)
      CAN0TXFGD = 0x00; // Highest priority

      // STEP 3: CLEAR TX COMPLETE FLAG
      CAN0TFLG = 0x01;// Clear the appropriate Tx Flag to start
tranmission

   CRGFLG = 0x80;
}


extern void _can0_isr(void)
{
   *addr_latcurr1 = *(CAN0RXFG+4);
   *addr_latcurr2 = *(CAN0RXFG+5);
   *addr_latcurr3 = *(CAN0RXFG+6);
   *addr_latcurr4 = *(CAN0RXFG+7);

   *addr_loncurr1 = *(CAN0RXFG+8);
   *addr_loncurr2 = *(CAN0RXFG+9);
   *addr_loncurr3 = *(CAN0RXFG+10);
   *addr_loncurr4 = *(CAN0RXFG+11);

        CAN0RFLG = 0x01;
}


// Store buffer function
void store_buffer(char data_string[])
{
   int temp_counter = 0;
   while(data_string[temp_counter] != '\0')
   {
      // Creating a string buffer to send to other MCU using SPI
      sci_output[sci_counter] = data_string[temp_counter];
        sci_counter++;
      temp_counter++;
   }
   sci_output[sci_counter] = '\0';
}
```

B-2: GNC Code

```
/****   Include  the following files ***/
#include <mc9s12dp256.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include <stddef.h>
#include <time.h>
#include <C:\Code\UAVFin5\mc9s12dg256_int_vec.c>
#include <C:\Code\UAVFin5\function_library.h>
```

```
// Define The Transmmit buffer
#define  CAN0TXFG0   _P(0x170)
#define  CAN0TXFG1   _P(0x171)
#define  CAN0TXFG2   _P(0x172)
#define  CAN0TXFG3   _P(0x173)
#define  CAN0TXFG4   _P(0x174)
#define  CAN0TXFG5   _P(0x175)
#define  CAN0TXFG6   _P(0x176)
#define  CAN0TXFG7   _P(0x177)
#define  CAN0TXFG8   _P(0x178)
#define  CAN0TXFG9   _P(0x179)
#define  CAN0TXFGA   _P(0x17A)
#define  CAN0TXFGB   _P(0x17B)
#define  CAN0TXFGC   _P(0x17C)
#define  CAN0TXFGD   _P(0x17D)


// Timer associated variables
unsigned int counter = 0;
unsigned int sci_counter = 0;
unsigned int can_counter = 0;
unsigned int rti_counter = 0;
unsigned int serial_flag = 0;
char sci_output[256], sci_input[256];

// Current parameter values
float GP2, compass, loncurr, latcurr, currtime, xdist, ydist;

// These are the memory address of where the strings are stored
unsigned char *addr_GP1, *addr_GP2, *addr_GP3, *addr_GP4;
unsigned char *addr_compass1, *addr_compass2, *addr_compass3,
*addr_compass4;
unsigned char *addr_loncurr1, *addr_loncurr2, *addr_loncurr3,
*addr_loncurr4;
unsigned char *addr_latcurr1, *addr_latcurr2, *addr_latcurr3,
*addr_latcurr4;

// Some constants such as base location in latitude and longitude,
and pi
const float latbase = 2548.5155;
const float lonbase = 5448.5125;
const float m_pi = 3.14159;   // PI

// Servo motor variables
float M0z,M1z,M2z,M3z;  // Servo Motor offset and trims
float M0s,M1s,M2s,M3s;  // Servo Motor gains


void main(void)
{

   /***** INITIALIZING TIME AND COUNTER VARIABLES ************/
   // Parameter values
   unsigned int i, j;
   char *tempstr, tempchar, temparray[15];
   int *status;
   float prevtime, psi, xvel, yvel, deltatime;

   addr_GP1 = (unsigned char*)&GP2+0;
   addr_GP2 = (unsigned char*)&GP2+1;
```

```
   addr_GP3 = (unsigned char*)&GP2+2;
   addr_GP4 = (unsigned char*)&GP2+3;

   addr_compass1 = (unsigned char*)&compass+0;
   addr_compass2 = (unsigned char*)&compass+1;
   addr_compass3 = (unsigned char*)&compass+2;
   addr_compass4 = (unsigned char*)&compass+3;

   addr_latcurr1 = (unsigned char*)&latcurr+0;
   addr_latcurr2 = (unsigned char*)&latcurr+1;
   addr_latcurr3 = (unsigned char*)&latcurr+2;
   addr_latcurr4 = (unsigned char*)&latcurr+3;

   addr_loncurr1 = (unsigned char*)&loncurr+0;
   addr_loncurr2 = (unsigned char*)&loncurr+1;
   addr_loncurr3 = (unsigned char*)&loncurr+2;
   addr_loncurr4 = (unsigned char*)&loncurr+3;

   clc_setup();  // Increase the bus clock to 24 MHz
   sci_setup();  // Set the max possible baud rate of 115200 baud
   adc_setup();
   inc_setup();
   pwm_setup();
   rti_setup();
   can_setup(2);
   asm("cli");

   xdist = 0;
   ydist = 0;
   prevtime = time_sec();

   while (1)
   {

//=====================================================================
=========//
//                                                PART ONE
//=====================================================================
=========//
      psi = compass*m_pi/180.0;     // Get the current heading in
radians
      xvel = cos(psi)*GP2;    // Find the current x axis distance
traveled
      yvel = sin(psi)*GP2;    // Find the current y axis distance
traveled
      currtime = time_sec();  // Capture the current time
      deltatime = currtime - prevtime;    // Find the time difference
      prevtime = currtime;    // Store the current time as the
previous time
      xdist = xdist + xvel*deltatime;     // Integrate the x axis
distance
      ydist = ydist + yvel*deltatime;     // Integrate the y axis
distance
      latcurr = latbase + xdist/(60*30);  // convert x axis distance
to lat
      loncurr = lonbase + ydist/(60*27);  // convert y axis distance
to lon

        if (rti_counter < 25)
           continue;
```

```
        rti_counter = 0;

//=================================================================
=========//
//                                                  PART TWO
//=================================================================
=========//

      // NextGet the serial inputs.  GPS and Compass are serial
        i = 0;
        while ((tempchar = getchar(0)) != '$'); // Wait for a dollar
sign
                                                                    //
to indicate the start of a line

        // Capture 127 characters from the line.  127 characters are
sent by HILS
        // Line in NMEA format
        while (i < 85)
        {
         sci_input[i] = getchar(0);
             i++;
        }
        sci_input[i] = '\n';
        sci_input[i+2] = '\0';

        // Parse the string to find the appropriate parameter
        // Find Latitude
        tempstr = strstr(sci_input, ",V,");
        i = 3;
        j = 0;
        while (*(tempstr+i) != ',')
        {
         temparray[j] = *(tempstr+i);
             i++;
             j++;
        }
        temparray[j] = '\0';
        latcurr = atof(temparray);

        // Find the longitude
        i = i+3;
        j = 0;
        while (*(tempstr+i) != ',')
        {
         temparray[j] = *(tempstr+i);
             i++;
             j++;
        }
        temparray[j] = '\0';
        loncurr = atof(temparray);

        xdist = (latcurr - latbase)*60*30;
        ydist = (loncurr - lonbase)*60*27;

   } // end of main while loop

} //end of main
```

```
extern void _incO_isr(void)
{
        TFLG2 = 0x80;
        counter++;
      //printchar('P');
}


extern void _sci0_isr(void)
{
   unsigned char flagstore, datastore;
   flagstore = SCI0SR1;
   datastore = SCI0DRL;

   if (datastore == '#')
      serial_flag = 1;  // set serial flag if serial transmission
requested by
                                    // Ground Station
}


extern void _rti_isr(void)
{
   // STEP 1: SELECT TRANSMIT BUFFER
   // CANxTBSEL
   // TX2 - TX0 : Selects the approprate CAN Transmit Buffer
   CAN0TBSEL = CAN0TFLG;         // Make one of the three Tx buffers
available

   // STEP 2: SET UP THE TRANSMIT MESSAGE FRAME
   CAN0TXFG0 = 'T'; // IDR = Identification
   CAN0TXFG1 = 0xFF; // IDR = Identification
   CAN0TXFG2 = 'A'; // IDR = Identification
   CAN0TXFG3 = 0xFE; // IDR = Identification

   CAN0TXFG4 = *addr_latcurr1;      // DSR = Data byte 1
   CAN0TXFG5 = *addr_latcurr2;      // DSR = Data byte 2
   CAN0TXFG6 = *addr_latcurr3;      // DSR = Data byte 3
   CAN0TXFG7 = *addr_latcurr4;      // DSR = Data byte 4
   CAN0TXFG8 = *addr_loncurr1;      // DSR = Data byte 5
   CAN0TXFG9 = *addr_loncurr2;      // DSR = Data byte 6
   CAN0TXFGA = *addr_loncurr3;      // DSR = Data byte 7
   CAN0TXFGB = *addr_loncurr4;      // DSR = Data byte 8

   CAN0TXFGC = 0x08; // DLR = Data Length (set to 8 for 8 bytes of
data)
   CAN0TXFGD = 0x00; // Highest priority

   // STEP 3: CLEAR TX COMPLETE FLAG
   CAN0TFLG = CAN0TBSEL;   // Clear the appropriate Tx Flag to start
tranmission

   rti_counter++;
        CRGFLG = 0x80;
}
```

```
extern void _can0_isr(void)
{

      *addr_GP1 = *(CAN0RXFG+4);
      *addr_GP2 = *(CAN0RXFG+5);
      *addr_GP3 = *(CAN0RXFG+6);
      *addr_GP4 = *(CAN0RXFG+7);

      *addr_compass1 = *(CAN0RXFG+4);
      *addr_compass2 = *(CAN0RXFG+5);
      *addr_compass3 = *(CAN0RXFG+6);
      *addr_compass4 = *(CAN0RXFG+7);


        CAN0RFLG = 0x01;
}



// Store buffer function
void store_buffer(char data_string[])
{
   int temp_counter = 0;
   while(data_string[temp_counter] != '\0')
   {
      // Creating a string buffer to send to other MCU using SPI
      sci_output[sci_counter] = data_string[temp_counter];
        sci_counter++;
      temp_counter++;
   }
   sci_output[sci_counter] = '\0';
}
```

B-3: Resource Files

Function Library

```
#include <mc9s12dP256.h>
#include <math.h>
#include <stddef.h>
#include <stdlib.h>
#include <C:\Code\UAVFin4\function_library.h>

extern unsigned int counter;
extern float M0z,M1z,M2z,M3z;   // Servo Motor offset and trims
extern float M0s,M1s,M2s,M3s;   // Servo Motor gains


/*****************************************************************
**********/
// Make the bus clock speed to 24 MHz
void clc_setup(void)
{
```

```
    while ((CRGFLG & 0x08) == 0x00);
    CLKSEL = 0x00;
    SYNR = 0x02;
    REFDV = 0x01;
    while ((CRGFLG & 0x08) == 0x00);
    CLKSEL = 0x80;
    while ((CRGFLG & 0x08) == 0x00);
}


/*****************************************************************
**********/
//void serialsetup(int baud,int comport){
void sci_setup(void)
{
       SCI0BDL=0x0D;
       SCI0CR2=0x0C;     /* Enable transmit and recieve 115200*/
       SCI1BDL=0x0D;
       SCI1CR2=0x0C;     /* Enable transmit and recieve 115200*/
}//end of function


/*****************************************************************
**********/
void adc_setup(void)
{
// ATD0CTL2 ATD on/off, auto SCF clear, externel trig, int on/off,
int flag
// ATD0CTL3  Num of conversions (0-8), FIFO mode on/off (keep fifo
off)
// ATD0CTL4  ATD res (8/10 bits), ATD clock cycle for conv, prescaler
ATD clock
// ATD0CTL5  right/left just, signed/unsigned, cont/single scan,
mult/signle ch
// ATD0STAT0 seq comp flag,ext trig overrun flag,fifo overrun
flag,conv counter
// ATD0STAT1 Individual channel conversion complete flag
// ATD0DIEN       enables digital input in each ATD channels
// PORTAD0        stores the digital value of each ATD channel
// ATD0DR0 2 byte ATD value for the 1st channel that was converted
(non fifo)
// ATD0DR1 2 byte ATD value for the second channel that was converted
(non fifo)
// ATD0DR2 2 byte ATD value for the second channel that was converted
(non fifo)
// ATD0DR3 2 byte ATD value for the second channel that was converted
(non fifo)
// ATD0DR4 2 byte ATD value for the second channel that was converted
(non fifo)
// ATD0DR5 2 byte ATD value for the second channel that was converted
(non fifo)
// ATD0DR6 2 byte ATD value for the second channel that was converted
(non fifo)
// ATD0DR7 2 byte ATD value for the second channel that was converted
(non fifo)

   ATD0CTL2=0x80; //A/D System ON,no power down,no external
trigger,no interrupt
   ATD0CTL3=0x00; // 8 conversions per sequence, no FIFO mode, and no
power down
```

```
   ATD0CTL4=0x77; // 10 bit resolution, 2 + 16 ATD clock, ATD clock
at 500 kHz
   ATD0CTL5=0x80; // right justify, unsigned, single scan, single
channel

   ATD1CTL2=0x80; //A/D System ON,no power down,no external
trigger,no interrupt
   ATD1CTL3=0x00; // 8 conversions per sequence, no FIFO mode, and no
power down
   ATD1CTL4=0x77; // 10 bit resolution, 2 + 16 ATD clock, ATD clock
at 500 kHz
   ATD1CTL5=0x80; // right justify, unsigned, single scan, single
channel
}


/*****************************************************************
**********/
void inc_setup(void)
{
// TIOS  Timer Input Capture or Output Compare Select Register
// TCNT  Timer Count Register  (Free running timer)
// TSCR1 Timer System Control Register 1
// TCTL3    Timer Control Reg 3 (Selects between input caputer in
rising
// TCTL4 Timer Control Reg 4 (edge or falling edge or both for
channels 7-0)
// TIE  Timer Interrupt Enable Register (Enables interrupts for all
channels)
// TSCR2 Timer System Control Reg 2 (Enables overflow interrupt and
prescale)
// TFLG1 Main Timer Interrupt Flag for all channels, check ch 0 (1 to
clear)
// TFLG2 Main Timer Interrupt Flag for Overflow (write 1 to clear)
// TC0      Timer Input Capture Register for Ch 0 (2 bytes) (Stores
value of timer)
// TC1      Timer Input Capture Register for Ch 1 (2 bytes) (Stores
value of timer)
// TC2      Timer Input Capture Register for Ch 2 (2 bytes) (Stores
value of timer)
// TC3      Timer Input Capture Register for Ch 3 (2 bytes) (Stores
value of timer)
// TC4      Timer Input Capture Register for Ch 4 (2 bytes) (Stores
value of timer)
// TC5      Timer Input Capture Register for Ch 5 (2 bytes) (Stores
value of timer)
// TC6      Timer Input Capture Register for Ch 6 (2 bytes) (Stores
value of timer)
// TC7      Timer Input Capture Register for Ch 7 (2 bytes) (Stores
value of timer)

    TIOS  = 0x00; // Set all channels to input capture
    TCTL3 = 0x55; // Set channels 7 - 4 to capture on rising edge
    TCTL4 = 0x55; // Set channels 3 - 0 to capture on rising edge
    TIE   = 0xFF; // Enable interrupt on channel 0 - 7.
    TSCR2 = 0x87; // Enable overflow interrupt. bus clock / 128
    TSCR1 = 0x80; // Enable Timer
}
```

```
/*****************************************************************
**********/
void pwm_setup(void)
{
    PWME=0xFF;       // Turn on PWM in Channel 0 - 7
    PWMPOL=0xFF;   // High duty cycle
    PWMCLK=0xFF;   // SB and SA clocks used instead of B and A clocks
    PWMPRCLK=0x66; // divide clock A, B by 64
    PWMCAE=0x00;   // left align
    PWMCTL=0x00;   // no concatenation of pwm channels, no power down
in freeze
    PWMSCLA=0x03;  // clock further divided by 6. 24MHz / (64 * 6) =
62,500
    PWMSCLB=0x03;  // clock further divided by 6. 24MHz / (64 * 6) =
62,500
    PWMPER0=0xFF;  // Period is 256. So PWM frequency = 62,500/256 =
244 Hz
    PWMPER1=0xFF;  // Period is 256. So PWM frequency = 62,500/256 =
244 Hz
    PWMPER2=0xFF;  // Period is 256. So PWM frequency = 62,500/256 =
244 Hz
    PWMPER3=0xFF;  // Period is 256. So PWM frequency = 62,500/256 =
244 Hz
    PWMPER4=0xFF;  // Period is 256. So PWM frequency = 62,500/256 =
244 Hz
    PWMPER5=0xFF;  // Period is 256. So PWM frequency = 62,500/256 =
244 Hz
    PWMPER6=0xFF;  // Period is 256. So PWM frequency = 62,500/256 =
244 Hz
    PWMPER7=0xFF;  // Period is 256. So PWM frequency = 62,500/256 =
244 Hz
}//end of function


void can_setup(int uc)
{

   // PUT THE CAN IN THE INITIALIZATION MODE
   // TO DO THAT USE THE CANCTL0 REGISTER
   CAN0CTL0 = 0x01;  // REQUEST FOR INIT MODE
   while ((CAN0CTL1 & 0x01) == 0);    // WAIT FOR CAN TO ACKNOWLEDGE
INIT
                                      // MODE REQUEST

   // NOW THE CAN IS IN THE INIT MODE

   // CANxBTR0
   // SJW1 SJW0:  0 0 = 1 time quanta clock cycle, 2(01), 3(10) and
4(11)
   // BRP5:0   :  prescaler (SELECT 3F for prescale by 64)
   CAN0BTR0 = 0xFF; // 4 Tqs and prescale by 64

   // NOTE:  YOU CAN THINK OF TQ AS A CLOCK CYCLE

   // CANxBTR1
   // SAMP     : 1 = 3 time samples 0 = One time sample per bit
(SELECT 1)
   // TSEG2   : SELECT 7 for 8 Tq clock cycles
   // TSEG1   : SELECT 13 also for 14 Tq clock cycles
   CAN0BTR1 = 0xFF; // 3 samples per bit, 8 Tq TSEG2, 16 Tq TSEG1
                    // Total Bit time = 4 + 8 + 16 = 28 bits
```

```
                    // Rate = 24 MHz / (64 * 28) ~ 13392 baud

   // CANxCTL1
   // CANE    : 1 = enable 0 = disable                (SELECT 1)
   // CLKSRC  : 1 = bus clock 0 = oscillator     (SELECT 0)
   // LOOPB   : 1 = loop back mode, 0 = normal   (SELECT 0)
   // LISTEN  : 1 = listen only mode,0 = normal (SELECT 0)
   // 0 = 0
   // WUPM     : 1 = CPU woken only if signal changes from recessive
to
   //                   dominant and the dominant signal lasts for
Twup
   //              0 = CPU woken if signal changes from recessive to
dominant
   //              (SELECT 1)
   // SLPAK   : only a flag.
   // INITAK  : only a flag
   CAN0CTL1 = 0x84;  // enable can and select bus clock as CAN clock
source
                     // and wake up the CAN module only if dominant
signal
                     // last for Twup

   // SET THE MASK REGISTER TO ACCEPT ALL INCOMMING MESSAGES
   // CANxIDMRx: Data Mask register. '1' indicates accept all without
going
   //                    through the acceptance register.
   CAN0IDAC = 0x00;
   if (uc == 1)
   {
      CAN0IDMR0 = 0x00;//0xFF;
      CAN0IDMR1 = 0x00;//0xFF;
      CAN0IDMR2 = 0x00;//0xFF;
      CAN0IDMR3 = 0x00;//0xFF;

      CAN0IDMR4 = 0x00;//0xFF;
      CAN0IDMR5 = 0x00;//0xFF;
      CAN0IDMR6 = 0x00;//0xFF;
      CAN0IDMR7 = 0x00;//0xFF;

      CAN0IDAR0 = 'T';
      CAN0IDAR1 = 0xFF;
      CAN0IDAR2 = 'A';
      CAN0IDAR3 = 0xFE;

      CAN0IDAR4 = 'T';
      CAN0IDAR5 = 0xFF;
      CAN0IDAR6 = 'A';
      CAN0IDAR7 = 0xFE;
   }
   if (uc == 2)
   {
      CAN0IDMR0 = 0x00;//0xFF;
      CAN0IDMR1 = 0x00;//0xFF;
      CAN0IDMR2 = 0x00;//0xFF;
      CAN0IDMR3 = 0x00;//0xFF;

      CAN0IDMR4 = 0x00;//0xFF;
      CAN0IDMR5 = 0x00;//0xFF;
      CAN0IDMR6 = 0x00;//0xFF;
      CAN0IDMR7 = 0x00;//0xFF;
```

```
      CAN0IDAR0 = 'G';
      CAN0IDAR1 = 0xFF;
      CAN0IDAR2 = 'N';
      CAN0IDAR3 = 0xFE;

      CAN0IDAR4 = 'G';
      CAN0IDAR5 = 0xFF;
      CAN0IDAR6 = 'N';
      CAN0IDAR7 = 0xFE;
   }

   // Now we need to leave the initialization mode before we can
change the
   // CANCTL0 register.  To leave the initialization mode, put a 0
into the
   // INTRQ bit and then wait for the acknowledgement
   CAN0CTL0 = 0x00; // Request to leave INIT mode
   while ((CAN0CTL1 & 0x01) == 1);    // WAIT FOR CAN TO ACKNOWLEDGE
REQUEST
                                      // TO LEAVE INIT MODE REQUEST

   // CANxCTL0
   // RXFRM   : only a receive frame flag, (SELECT 1 to clear)
   // RXACT   : only a receiver active flag
   // CSWAI   : 1 = stop in wait mode 0 = normal operation (SELECT 0)
   // SYNCH   : only a synchronize flag
   // TIME    : 0 = time stamp disable 1 = enable (SELECT 1)
   // WUPE    : 1 = wake up enable 0 = disable (SELECT 1)
   // SLPRQ   : 1 = CAN can enter sleep mode 0 = normal  (SELECT 0)
   // INITRQ  : 1 =  MSCAN in init mode, 0 = exits init mode (SELECT
0)
   CAN0CTL0 = 0x8C;  // clear RXFRM flag, enable time stamp, wake CAN

   // Enable receive interrupt
   CAN0RIER = 0x01;

   // STEP 1: WAIT FOR SYNCHRONIZATION
   while (!(CAN0CTL0 & 0x10));  // wait for synchronization

}

void rti_setup (void)
{
   RTICTL = 0x7F; // Provide real time interrupt at
24,000,000/(16*2^16) = 22 Hz
   CRGINT = 0x80; // Enable real time interrupts
}


/*****************************************************************
**********/
char getchar(int chan)
{
       char x;
       if (!chan)
       {
             while ( !(SCI0SR1 & 0x20) );     /* Loop until character
is recieved*/
             x=SCI0DRL;
       }
```

```
       else
       {
             while ( !(SCI1SR1 & 0x20) );     /* Loop until character
is recieved*/
             x=SCI1DRL;
       }
       return x;
}//end of function


/*****************************************************************
**********/
void printfloat(float value, int chan)
{
   int i;
   int *status;
   char *temp = ftoa(value, status);
   i = 0;
   while(temp[i] != '.')
   {
     i++;
   }
   temp[i+2] = '\0';
   printstring(temp, chan);
}//end of function


/*****************************************************************
**********/
void printint(int value, int chan)
{
   char temp[10];
   itoa(temp, value, 10);
   printstring(temp, chan);
}//end of function


/*****************************************************************
**********/
void printstring(char smessage[], int chan)
{
   int count=0;
   while(1)
   {
       if(smessage[count]=='\0')
     {
        break;
     }
       if (smessage[count] == '\n')
          printnl(chan);
       else
       {
             printchar(smessage[count], chan);
       }
     count++;
   }//end of while
}//end of function
```

```
/*********************************************************************
**********/
void printnl(int chan)
{
       printchar(0x0D,chan);
       printchar(0x0A,chan);
}//end of function


/*********************************************************************
**********/
void printchar(char datum, int chan)
{
       if (!chan)
       {
            while ( !(SCI0SR1 & 0x80) );    /* Loop until character
is transmitted*/
            SCI0DRL=datum;
       }
       else
       {
            while ( !(SCI1SR1 & 0x80) );    /* Loop until character
is transmitted*/
            SCI1DRL=datum;
       }
}


/*********************************************************************
**********/
long time_ms (void)
{
   return (350L*(long)counter + (long)TCNT/188L);
}

float time_sec (void)
{
   return ((float)counter*0.34952533 + (float)TCNT/187500.00);
}



/*********************************************************************
**********/
void pwm_signal(float d,int ch)
{
    d=d*2.55;//255 is 0xff which is the period value
    switch(ch)
      {
        case 0:
                PWMDTY0=(int)d;
                  break;
        case 1:
                PWMDTY1=(int)d;
                  break;
        case 2:
                PWMDTY2=(int)d;
                  break;
        case 3:
                PWMDTY3=(int)d;
                  break;
```

```
    }//end of switch
}//end of function


/*****************************************************************
**********/
void motor(int mot, float degree)
{
    float duty;

    switch(mot)
      {
        case 0:
             if(degree<-10.0) degree=-10.0;    //added to saturate
the Aileron
             if(degree>10.0) degree=10.0;      //added to saturate
the Aileron
          duty=M0s*degree+M0z;
          pwm_signal(duty,0);
             break;
        case 1:
//           degree=-1.0*degree;//because throttle is invertly
installed on the aircraft
             if(degree<5.0) degree=5.0;   //added to saturate the
throttle
             if(degree>40.0) degree=40.0; //added to saturate the
throttle
             duty=M1s*degree+M1z;
             pwm_signal(duty,1);
             break;
        case 2:
//           degree=-1.0*degree;//because rudder is invertly
installed on the aircraft
             if(degree<-10.0) degree=-10.0;    //added to saturate
the rudder
             if(degree>10.0) degree=10.0;      //added to saturate
the rudder
             duty=M2s*degree+M2z;
             pwm_signal(duty,2);
             break;
        case 3:
//           degree=-1.0*degree;//because elevator is invertly
installed on the aircraft
             if(degree<-10.0) degree=-10.0;    //added to saturate
the elevator
             if(degree>10.0) degree=10.0;      //added to saturate
the elevator
             duty=M3s*degree+M3z;
             pwm_signal(duty,3);
             break;
    }//end of switch
}//end of function
```

Header File

```
// method prototypes
void clc_setup(void);
void sci_setup (void);
void adc_setup(void);
void inc_setup(void);
void pwm_setup(void);
void can_setup(int);
void rti_setup(void);
char getchar(int);
void printfloat(float, int);
void printint(int, int);
void printstring(char[], int);
void printnl(int);
void printchar(char, int);
long time_ms(void);
float time_sec(void);
void store_buffer(char[]);
void motor (int, float);
void pwm_signal (float, int);
```

Vector File

```
/* To use, either add this file to the project file list (preferred),
or
 * #include it in a single source file if you are just using
 * File->CompileToOutput (this works but using the Project feature is
better
 */

/* Vector Table for Dx256 S12 CPU
 * As is, all interrupts except reset jumps to 0xffff, which is most
 * likely not going to useful. To replace an entry, declare your
function,
 * and then change the corresponding entry in the table. For example,
 * if you have a SCI handler (which must be defined with
 * #pragma interrupt_handler ...) then in this file:
 * add
 *    extern void SCIHandler();
 * before th table.
 * In the SCI entry, change:
 *    DUMMY_ENTRY,
 * to
 *  SCIHandler,
 */

extern void _start(void);   /* entry point in crt??.s */
extern void _incO_isr(void); /* handler for timer overflow interrupts
*/
extern void _sci0_isr(void); /* handler for sci interrupts */
extern void _rti_isr(void); /* handler for real time interrupts */
extern void _can0_isr(void); /* handler for mscan 0 interrupts */

#pragma nonpaged_function _start
```

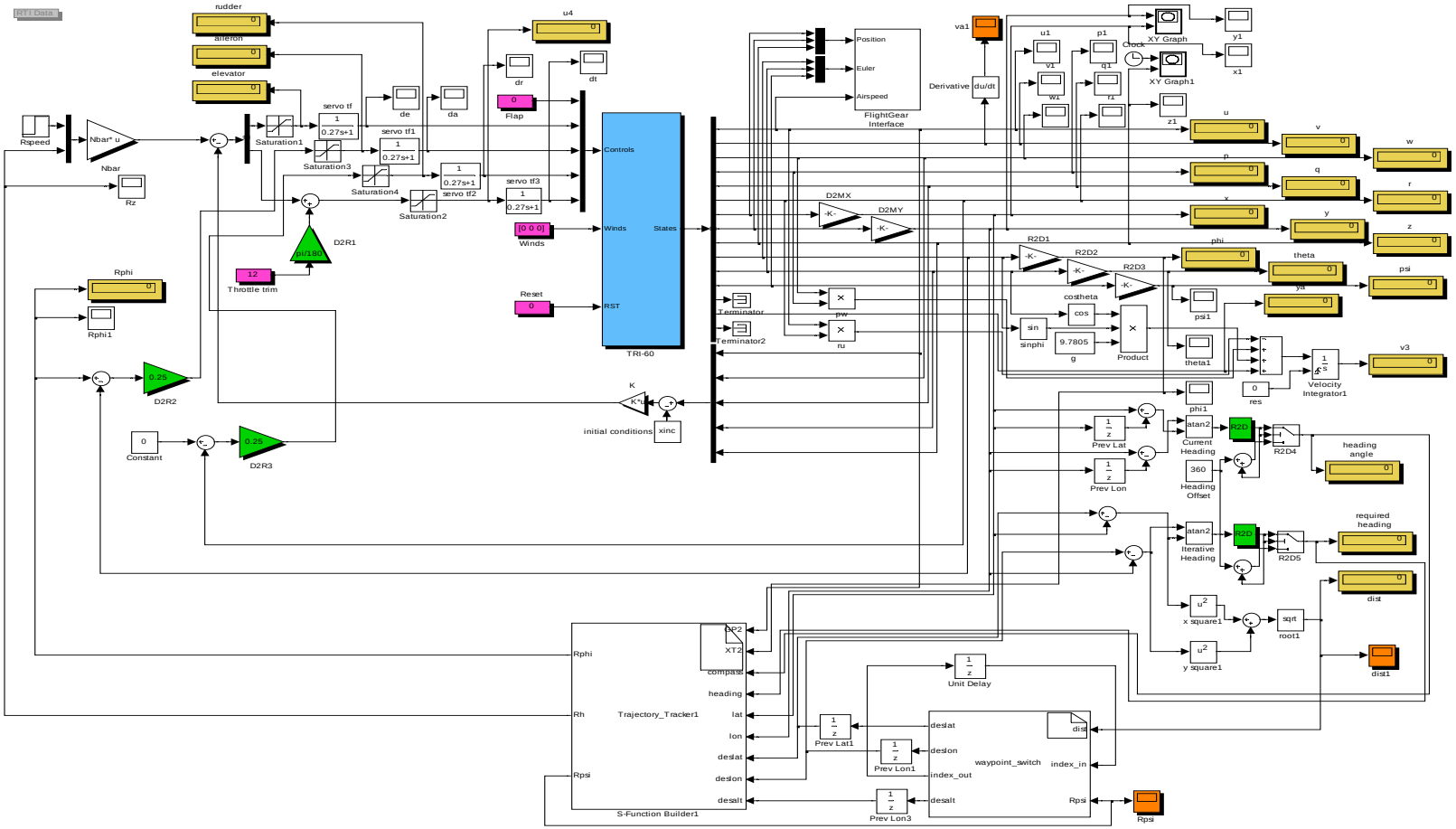
```
#pragma interrupt_handler _incO_isr
#pragma interrupt_handler _sci0_isr
#pragma interrupt_handler _rti_isr
#pragma interrupt_handler _can0_isr

#define DUMMY_ENTRY     (void (*)(void))0xFFFE

#pragma abs_address:0xFF80
/* change the above address if your vector starts elsewhere
 */
void (*interrupt_vectors[])(void) =
      {
      /* to cast a constant, say 0xb600, use
         (void (*)())0xb600
       */
      DUMMY_ENTRY, /*Reserved $FF80*/
      DUMMY_ENTRY, /*Reserved $FF82*/
      DUMMY_ENTRY, /*Reserved $FF84*/
      DUMMY_ENTRY, /*Reserved $FF86*/
      DUMMY_ENTRY, /*Reserved $FF88*/
      DUMMY_ENTRY, /*Reserved $FF8A*/
      DUMMY_ENTRY, /*PWM Emergency Shutdown*/
      DUMMY_ENTRY, /*Port P Interrupt*/
      DUMMY_ENTRY, /*MSCAN 4 Transmit*/
      DUMMY_ENTRY, /*MSCAN 4 Receive*/
      DUMMY_ENTRY, /*MSCAN 4 Error*/
      DUMMY_ENTRY, /*MSCAN 4 Wake-up*/
      DUMMY_ENTRY, /*MSCAN 3 Transmit*/
      DUMMY_ENTRY, /*MSCAN 3 Receive*/
      DUMMY_ENTRY, /*MSCAN 3 Error*/
      DUMMY_ENTRY, /*MSCAN 3 Wake-up*/
      DUMMY_ENTRY, /*MSCAN 2 Transmit*/
      DUMMY_ENTRY, /*MSCAN 2 Receive*/
      DUMMY_ENTRY, /*MSCAN 2 Error*/
      DUMMY_ENTRY, /*MSCAN 2 Wake-up*/
      DUMMY_ENTRY, /*MSCAN 1 Transmit*/
      DUMMY_ENTRY, /*MSCAN 1 Receive*/
      DUMMY_ENTRY, /*MSCAN 1 Error*/
      DUMMY_ENTRY, /*MSCAN 1 Wake-up*/
      DUMMY_ENTRY, /*MSCAN 0 Transmit*/
      _can0_isr, /*MSCAN 0 Receive*/
      DUMMY_ENTRY, /*MSCAN 0 Error*/
      DUMMY_ENTRY, /*MSCAN 0 Wake-up*/
      DUMMY_ENTRY, /*Flash*/
      DUMMY_ENTRY, /*EEPROM*/
      DUMMY_ENTRY, /*SPI2*/
      DUMMY_ENTRY, /*SPI1*/
      DUMMY_ENTRY, /*IIC Bus*/
      DUMMY_ENTRY, /*DLC*/
      DUMMY_ENTRY, /*SCME*/
      DUMMY_ENTRY, /*CRG Lock*/
      DUMMY_ENTRY, /*Pulse Accumulator B Overflow*/
      DUMMY_ENTRY, /*Modulus Down Counter Underflow*/
      DUMMY_ENTRY, /*Port H Interrupt*/
      DUMMY_ENTRY, /*Port J Interrupt*/
      DUMMY_ENTRY, /*ATD1*/
      DUMMY_ENTRY, /*ATD0*/
      DUMMY_ENTRY, /*SCI1*/
      _sci0_isr, /*SCI0*/
      DUMMY_ENTRY, /*SPI0*/
      DUMMY_ENTRY, /*Pulse Accumulator A Input Edge*/
```

```
        DUMMY_ENTRY, /*Pulse Accumulator A Overflow*/
        _incO_isr, /*Timer Overflow*/
        DUMMY_ENTRY, /*Timer Channel 7*/
        DUMMY_ENTRY, /*Timer Channel 6*/
        DUMMY_ENTRY, /*Timer Channel 5*/
        DUMMY_ENTRY, /*Timer Channel 4*/
        DUMMY_ENTRY, /*Timer Channel 3*/
        DUMMY_ENTRY, /*Timer Channel 2*/
        DUMMY_ENTRY, /*Timer Channel 1*/
        DUMMY_ENTRY, /*Timer Channel 0*/
        _rti_isr, /*Real Time Interrupt*/
        DUMMY_ENTRY, /*IRQ*/
        DUMMY_ENTRY, /*XIRQ*/
        DUMMY_ENTRY, /*SWI*/
        DUMMY_ENTRY, /*Unimplement Intruction Trap*/
        DUMMY_ENTRY, /*COP failure reset*/
        DUMMY_ENTRY, /*Clock monitor fail reset*/
        _start, /*Reset*/
        };
#pragma end_abs_address
```

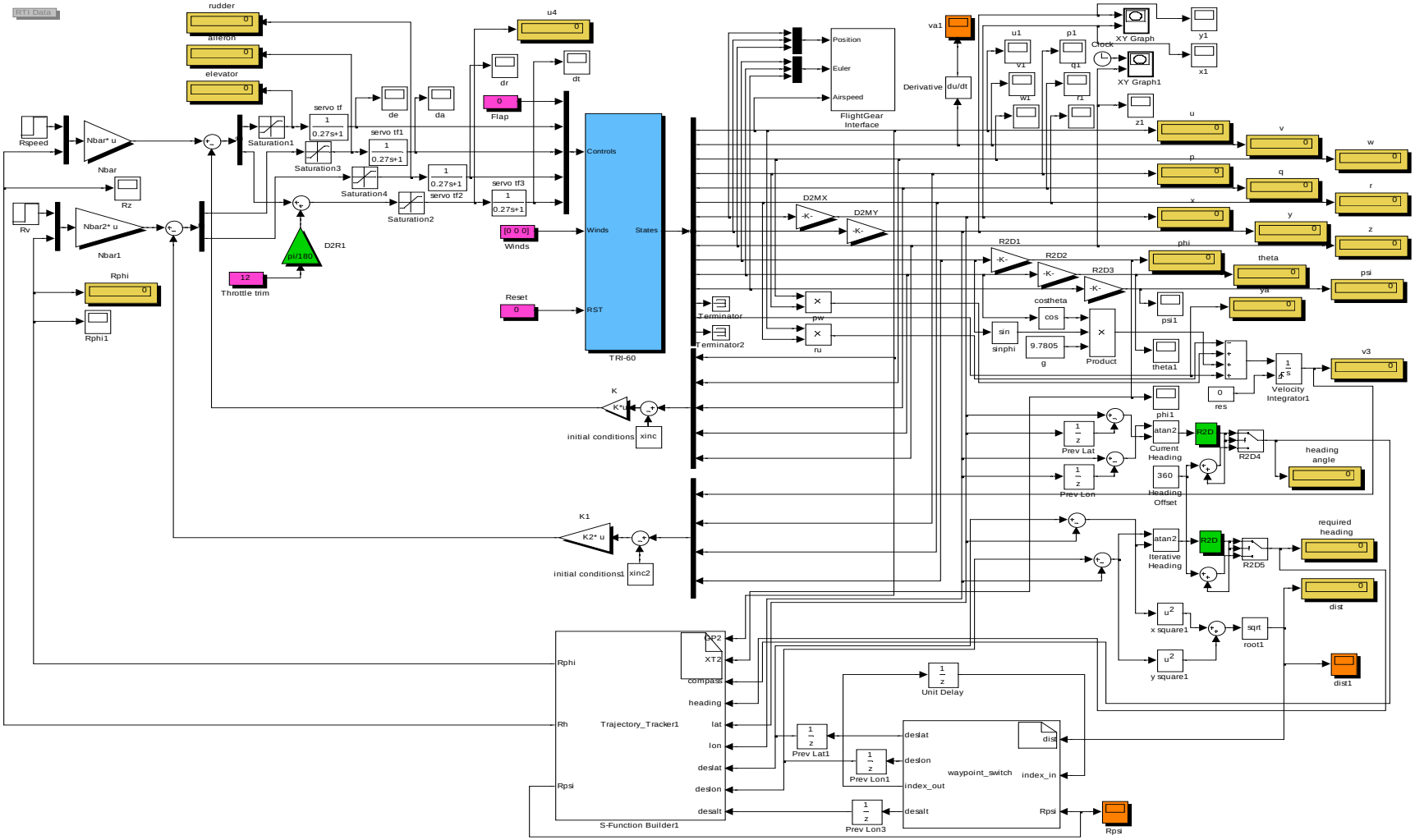
APPENDIX C

SIMULINK MODELS

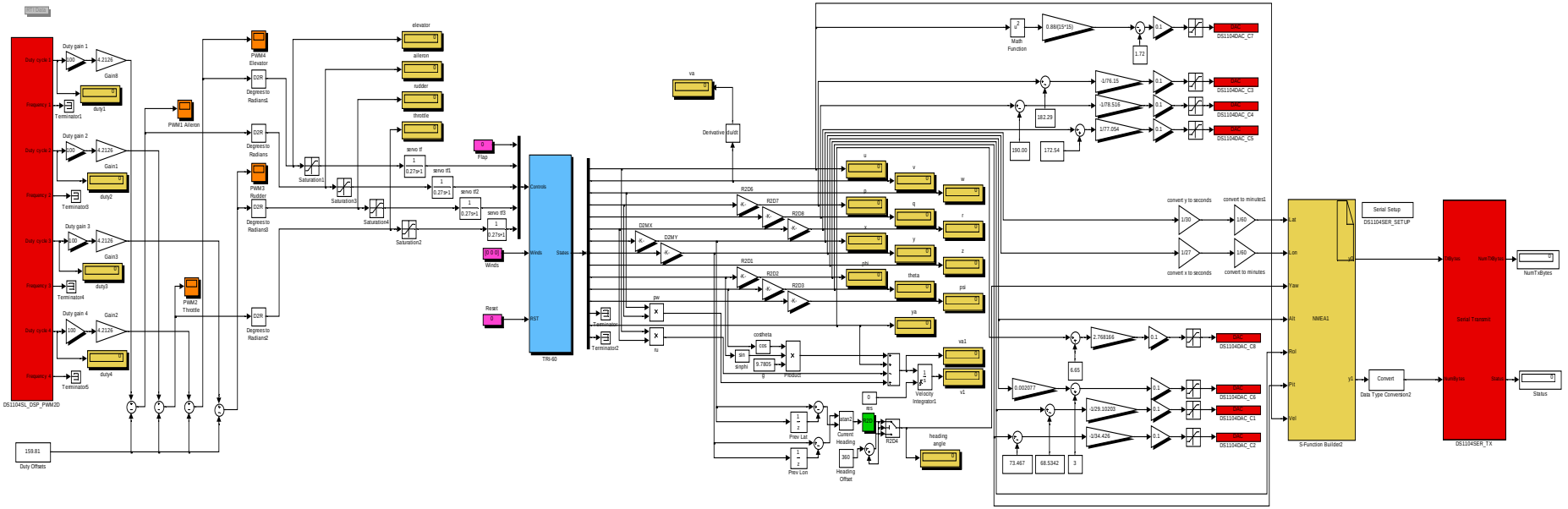
C-1: Simulation Only Model PID



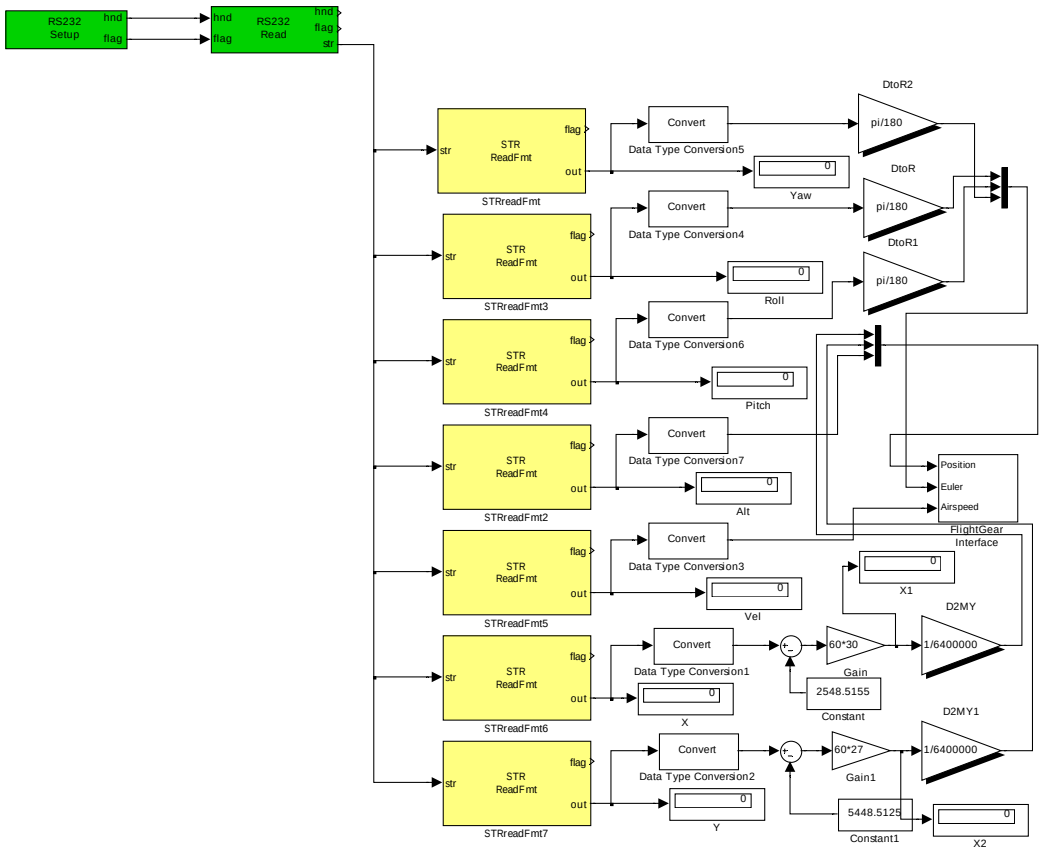
C-2: Simulation Only Model LQR



C-3: dSpace Model



C-4: Simulink Serial Model



C-5: Simulink Resource Files

This section consists of the C-code used to write the S-Functions.

Trajectory Tracker

```
double radius, angle, crsprod, clockwiseturn, anticlockwiseturn,
m_pi;
       double dirXvec, dirYvec, dirX1vec, dirY1vec, dirX2vec,
dirY2vec, Ra;
       signed int xc1, xc2, yc1, yc2, xc1diff, yc1diff, xc2diff,
yc2diff;
       unsigned int hyp1, hyp2;

       clockwiseturn = 0.20071286397934790134622443837619;
       anticlockwiseturn = -0.20071286397934790134622443837619;
       m_pi = 3.14159265358979323846264338327795;

       radius = pow(GP2[0],2)/(9.81*tan(fabs(XT2[0])*m_pi/180.0));
       if (radius > 255.0)
          radius = 255.0;

       // Find the difference between our current and the required
heading
       angle = fabs(compass[0] - heading[0]);

       // Calculate current direction vector
       dirXvec = cos(compass[0]*m_pi/180.0);
       dirYvec = sin(compass[0]*m_pi/180.0);

       // Calculate the two direction vectors for the two different
circles
       dirX1vec = dirYvec;
       dirY1vec = -dirXvec;

       dirX2vec = -dirYvec;
       dirY2vec = dirXvec;

       // Calculate the center of the two circles
       xc1 = (signed int)lat[0] + (signed int)(radius*dirX1vec);
       yc1 = (signed int)lon[0] + (signed int)(radius*dirY1vec);

       xc2 = (signed int)lat[0] + (signed int)(radius*dirX2vec);
       yc2 = (signed int)lon[0] + (signed int)(radius*dirY2vec);

       // Calculate distance from center of two circles to next
waypoint
       xc1diff = deslat[0] - xc1;
       yc1diff = deslon[0] - yc1;
       hyp1 = (unsigned int)sqrt (pow((float)xc1diff,2.0) +
pow((float)yc1diff,2.0));

       xc2diff = deslat[0] - xc2;
       yc2diff = deslon[0] - yc2;
```

```
       hyp2 = (unsigned int)sqrt (pow((float)xc2diff,2.0) +
pow((float)yc2diff,2.0));

       if (hyp1 < hyp2)
       {
          if (hyp1 >= radius)
             crsprod = (dirXvec*dirY1vec) - (dirX1vec*dirYvec);
          else
             crsprod = (dirXvec*dirY2vec) - (dirX2vec*dirYvec);
       }
       else
       {
          if (hyp2 >= radius)
             crsprod = (dirXvec*dirY2vec) - (dirX2vec*dirYvec);
          else
             crsprod = (dirXvec*dirY1vec) - (dirX1vec*dirYvec);
       }

       // If the heading error is more than 10 degrees, then we turn
 //      if (angle > 10.0  && angle < 350.0)
       if (angle > 0.0 && angle < 360.0)
       {
          if (crsprod < 0)
          {
             Ra = anticlockwiseturn;
          }
          else
          {
             Ra = clockwiseturn;
          }
       }
       else
       {
          Ra = heading[0] - compass[0];
          if (Ra >= 350.0)
             Ra = Ra - 360.0;
          if (Ra <= -350.0)
             Ra = Ra + 360.0;
          Ra = Ra*1.125*m_pi/180.0;
       }

       angle = heading[0] - compass[0];
       if (angle > 180.0)
          angle = angle - 360.0;

       if (angle < -180.0)
          angle = angle + 360.0;

       Rphi[0] = Ra;
       Rh[0] = desalt[0] - 50;
       Rpsi[0] = angle;
```

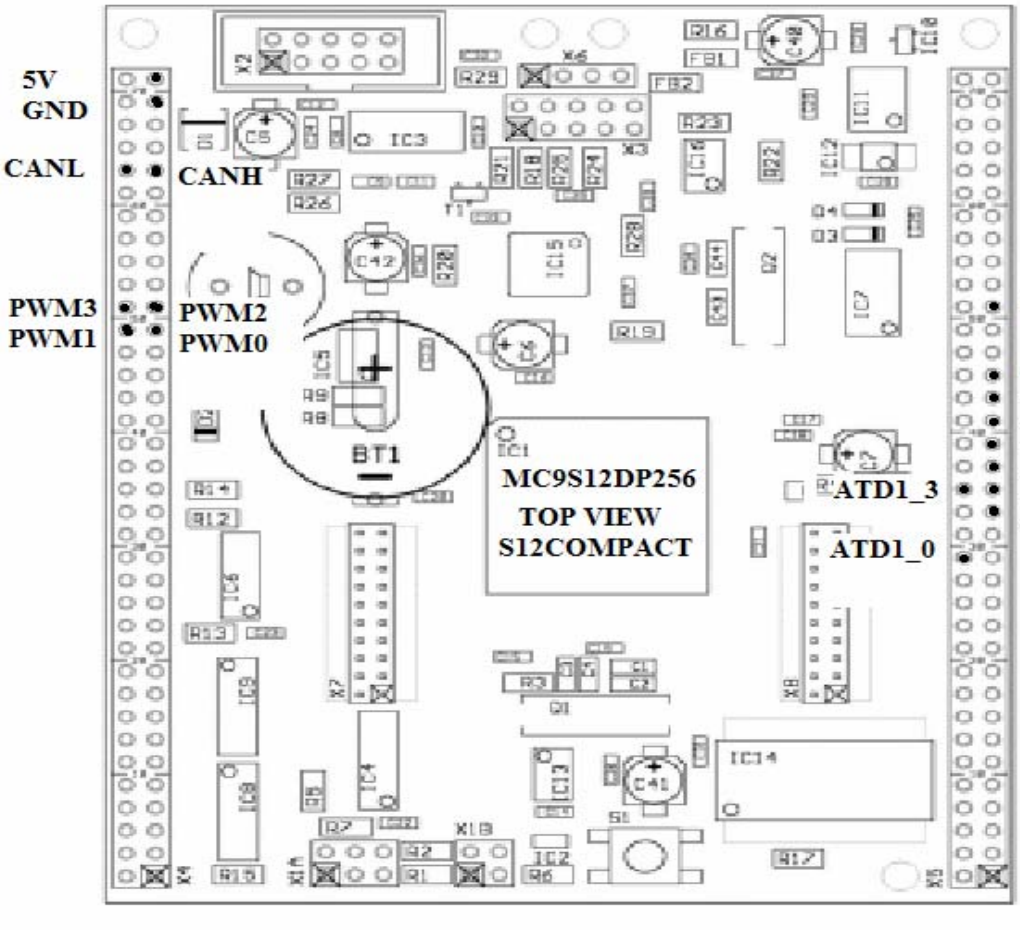
Waypoint Switch

```
double lat_array[6], lon_array[6], alt_array[6];

              lat_array[0] = 0;
              lon_array[0] = 500;
              alt_array[0] = 40;

              lat_array[1] = 1000;
              lon_array[1] = 0;
              alt_array[1] = 50;

              lat_array[2] = 0;
              lon_array[2] = 500;
              alt_array[2] = 60;

              lat_array[3] = 1000;
              lon_array[3] = 1000;
              alt_array[3] = 50;

     //         if (dist[0] <= 0.105) // heu
              if (dist[0] < 7) // pid
     //         if (dist[0] < 2.5) // lqr
              {
                 if (index_in[0] < 3)
                 {
                    deslat[0] = lat_array[index_in[0] + 1];
                    deslon[0] = lon_array[index_in[0] + 1];
                    index_out[0] = index_in[0] + 1;
                 }
                 else
                 {
                    deslat[0] = lat_array[0];
                    deslon[0] = lon_array[0];
                    index_out[0] = 0;
                 }
              }
              else
              {
                 deslat[0] = lat_array[index_in[0]];
                 deslon[0] = lon_array[index_in[0]];
                 index_out[0] = index_in[0];
                 if (fabs(angle[0]) < 0.5)
                    desalt[0] = alt_array[index_in[0]];
              }
```

APPENDIX D

HARDWARE UTILIZED

D-1: S12 Compact Pin Used

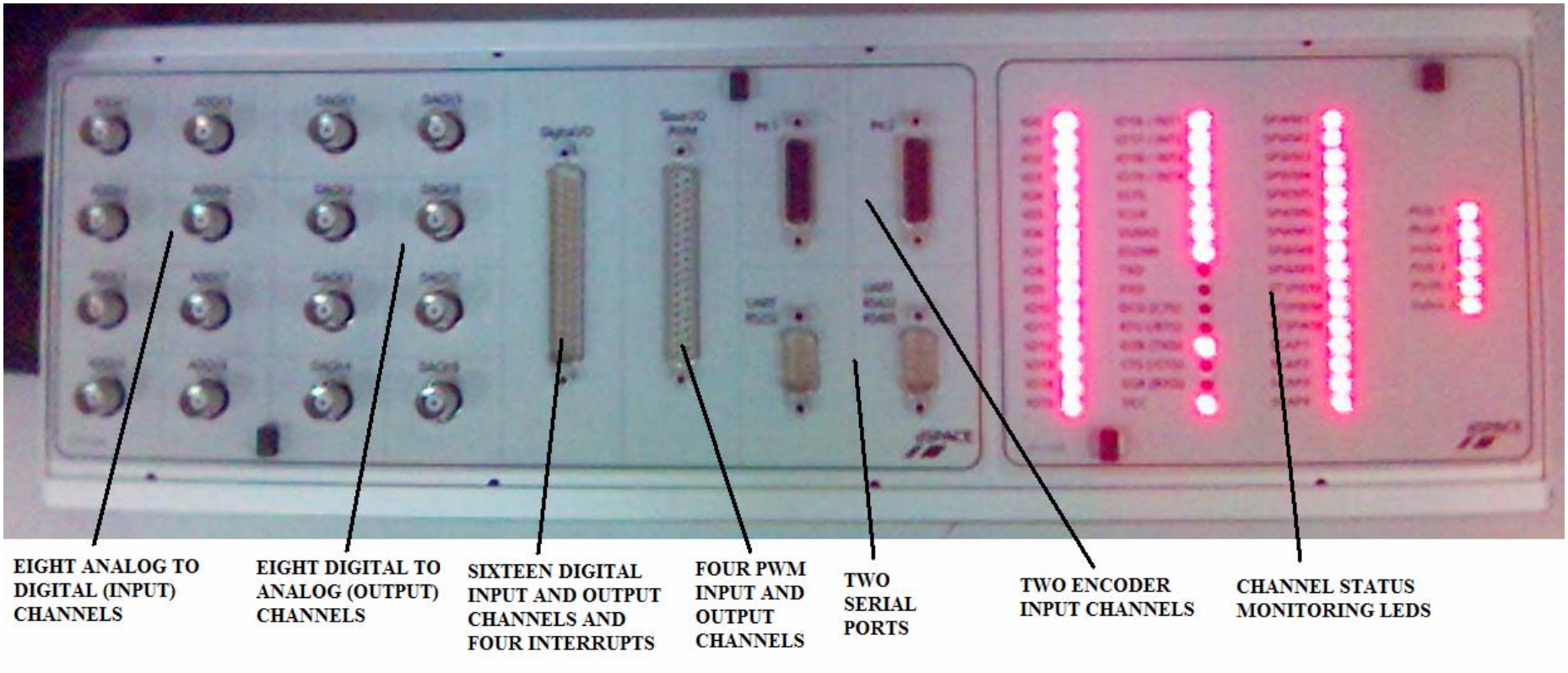


ATD0_2: ALTITUDE
ATD0_3: XRATE
ATD0_4: YRATE
ATD0_5: ZRATE
ATD0_6: ROLL
ATD0_7: PITCH

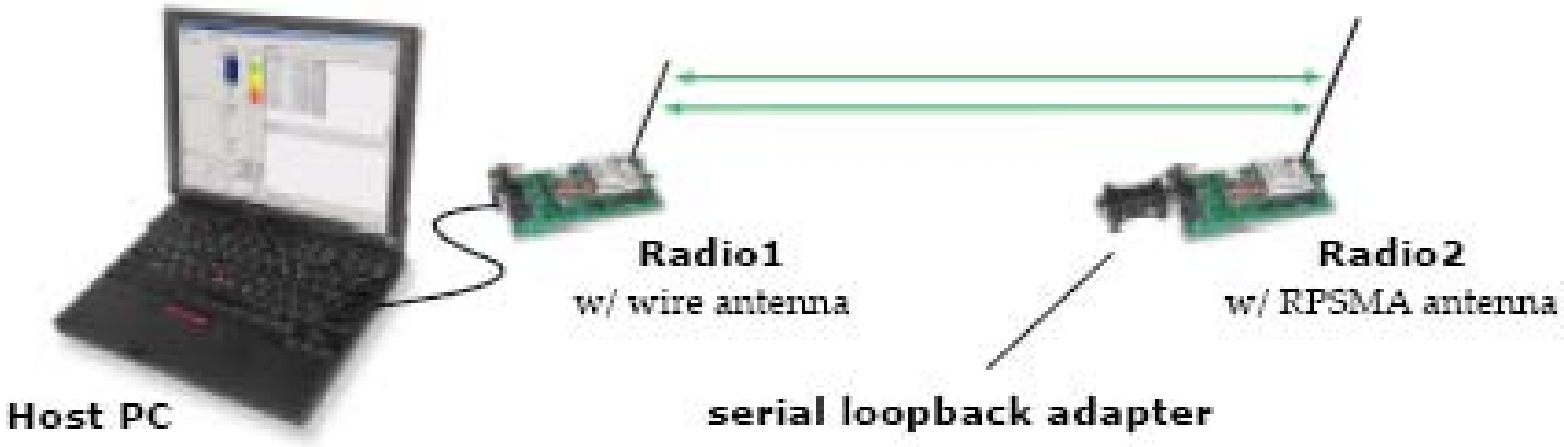
ATD1_0: Y ACCELERATION
ATD1_3: SPEED

PWM0: AILERON
PWM2: RUDDER
PWM3: ELEVATOR
PWM4: THROTTLE

D-2: dSpace Controller Board Pins



D-3: XSTREAM RF TRANSCEIVER



VITA

Mohammed Mahedi Hasan was born on April 25, 1980, in Dhaka, Bangladesh. He moved to United Arab Emirates at a very young age, and completed his high schooling there. He graduated from Arab Unity School as class valedictorian in 1996, He sat for his IGCSE O-Level Cambridge Board examinations in 1996. He graduated from St Mary Catholic High School in 1998. He sat for his GCE A-Level London Board examinations in 1997 and 1998. He received a Bachelors of Science Degree in Electrical Engineering in 2002 from Georgia Institude of Technology in Atlanta, Georgia, where he graduated with Honors.

Mr. Hasan worked in the United States as a Software Developer for a startup company called Core Concept Incorporated between 2002 and 2004, after which he moved baack to the United Arab Emirates. Mr. Hasan began his Master's Program in Mechatronics Engineering in American University of Sharjah in 2004. He was awarded the Masters of Science Degree in Mechatroincs Engineering in 2006.