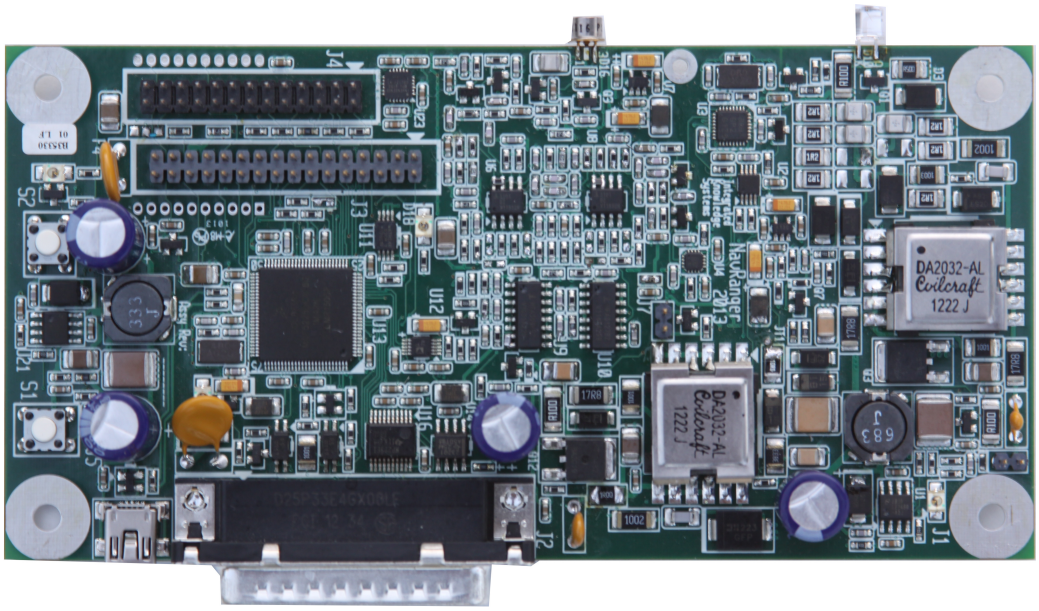
NAVRANGER-OEM



1/2/14

OEM Manual

NavRanger-OEM from Integrated Knowledge Systems is a single board laser ranging subsystem for integration into products requiring high-speed distance measurements. NavRanger uses the time-of-flight of short pulses from a class 1 laser. NavRanger also has a 9-Axis IMU and additional IO with USB, Serial, CAN or Analog output. This manual describes the default behavior using the USB port.

NavRanger-OEM

1. WARNINGS, CAUTIONS, AND DANGERS

Do not use NavRanger in applications where failure may cause injury or damage.

Warning

NavRanger-OEM contains no user serviceable components, requires no maintenance, and has no physical adjustments.

Modification may result in unsafe laser emissions and will void the warranty.

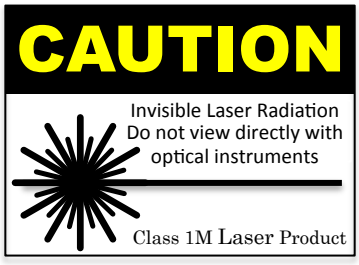
Caution - Laser Radiation

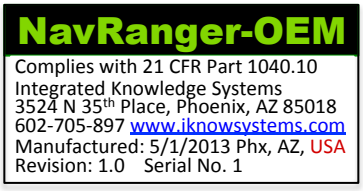
The emitted laser radiation is invisible!

The laser is class 1M due to the dispersion of the beam.

Viewing at less than 4 inches (100mm) or with a lens may result in eye damage.

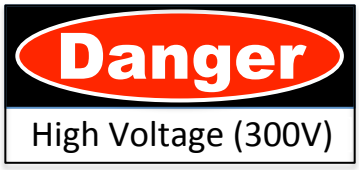
If a lens is added to the laser, a scanning interlock may be required to make the system class 1.





Danger - High Voltage

NavRanger has high voltage power supplies up to 300V. No not touch the board while power is supplied. Touching the board while power is supplied can cause damage to the board that is not covered by the warranty.



Warning

NavRanger is sensitive to static discharge and applying voltages beyond limits will cause damage to the board. This damage is not covered by the warranty.

2. QUICK START

Please perform the procedures outlined in this section to verify correct operation before installing into the final application. Each NavRanger board has been tested at the factory, but the failures can occur with software installs and various hardware configurations. Performing procedures in this section will verify the basic functionality.

2.1. Installing Software

The latest version of the NavRanger software package is available from Integrated Knowledge Systems web site at http://www.iknowsystems.com/navranger.html.

In the Resources section of the page, select the appropriate package for your operating system.

2.1.1. Mac OS X

Mac OS X 10.6.8 or newer is required.

1. Run a virus scan on the NavRangerMac.zip file.
2. Decompress the NavRangerMac.zip to a convenient location.
3. Locate the application NavRangerClient.app in NavRangerMac and go to section 2.2.

2.1.2. Windows

Windows XP, 7 or 8 is required. You also need the latest .Net updates.

1. Run a virus scan on the NavRangerWidows.zip file.
2. Decompress NavRangerWindows.zip to a convenient location.
3. Open the NavRangerDriver directory.
4. Execute InstallDriver.exe.
5. Note there may be some warning about using an unsigned driver. Please dismiss these warnings.
6. Click the Next button.
7. When the installer finishes, click the Finish button.
8. Execute vcredist_x86.exe for 32 bit systems or vcredist_x64.exe for 64 bit systems.
9. Follow the instructions to install the package.
10. If you have an AMD 64 bit processor, replace the file libusbK.dll in the NavRangerWindows directory with the libusbK.dll file from /NavRangerWindows/NavRangerDriver/amd64.
11. Locate the application NavRangerClient.exe in NavRangerWindows and go to section 2.2.

2.2. Connecting

1. Execute NavRangerClient.app on the Mac in the NavRangerMac directory or NavRangerClient.exe on Windows in the NavRangerWindows directory.
2. Power the NavRanger.
3. Connect a USB cable between the computer and NavRanger J5.
4. Verify data is updating in the NavClient window.
5. Set up a target about 12 inches from the NavRanger laser / photo diode.
6. Click the restore Parameters button to get the default parameters and activate the APD power.
7. Click the "Laser is Off" button to turn the laser On.
8. Verify the client is outputting reasonable distance data and it changes as the target moves.

2.3. Using the client software

The NavRanger client has two windows, NavControl and NavData. Clicking the Close button on either window exits the program.

2.3.1. Control Window

Figure 2.3.1 shows the NavRanger control window. The Control window has buttons for activating NavRanger Actions, setting Parameters, and shows the current Measurements in text fields and bars. Actions cause changes to the state of the NavRanger and are executed by clicking on the buttons indicated in the figure. The currently available Actions are listed in Table 2.3.1.1. Measurements indicate the state of the NavRanger sensors and are explained in Table 2.3.1.2. Parameters consist of text boxes and buttons. Enter the parameter value in the text box and click the associated button to write the value. Note that any unwritten parameter fields will be overwritten buy the current value when any of the parameters are written by clicking the button.

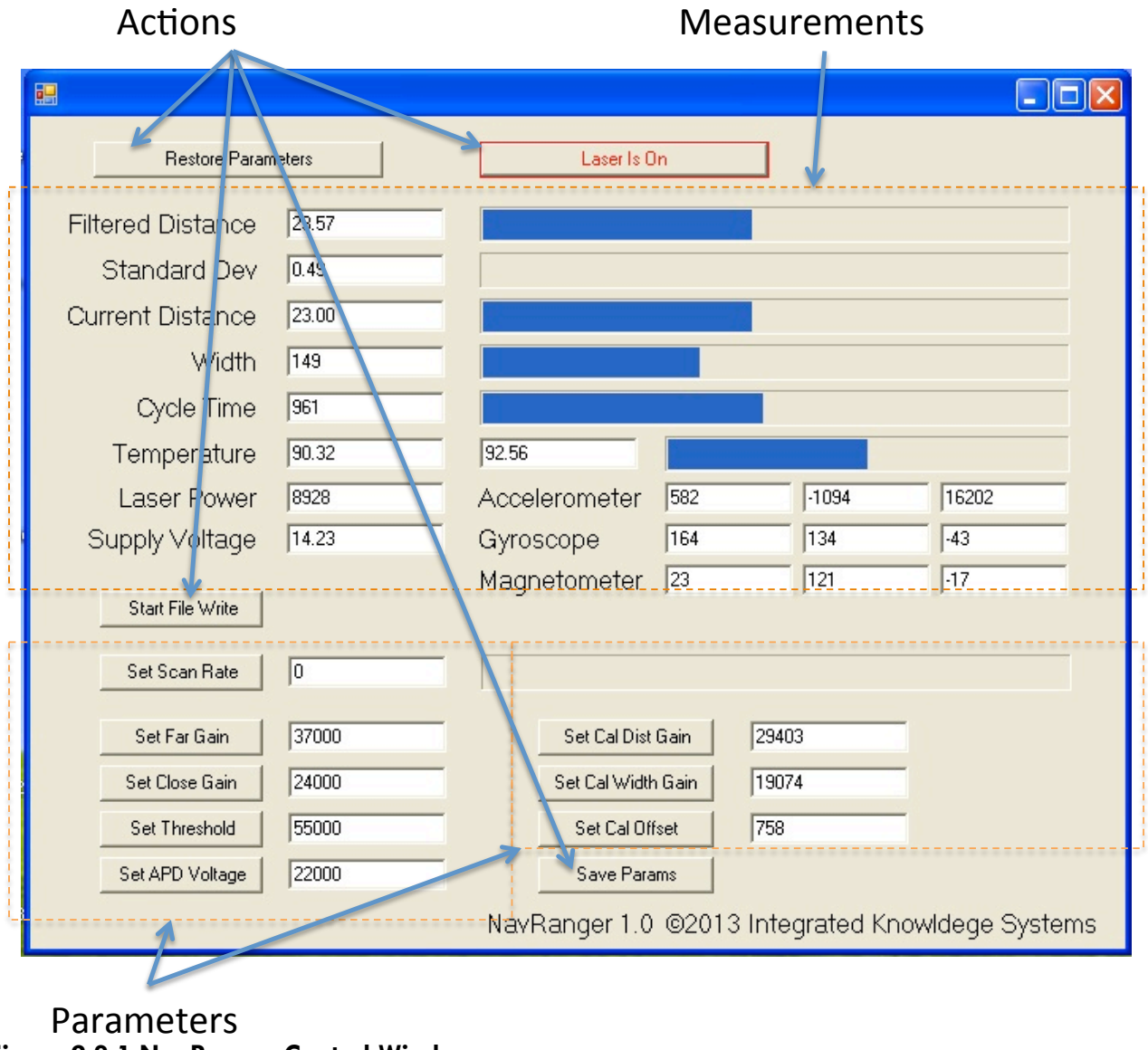


Figure 2.3.1 NavRanger Control Window

Table 2.3.1.1: Actions

Button Name	Action
Restore Parameters	Loads the configuration parameters from NavRanger flash memory to working space. If not already started, the photodiode voltage is controlled to the APD voltage.
Laser Is On Laser Is Off	Toggles the state of the laser and indicates its state.
Toggle Save Data	Starts recording of the continuous data to a file located in the same directory as the application in Comma Separated Text format. The first row of the file indicates the value contained in each column. The file name is LaserDistanceData-n.csv, where n is the file number. File numbers start at one when the application starts and will overwrite files with the same name. Note that large files can be created very quickly. There will be about 20K rows generated per second.
Save Parameters	Saves the current working space parameters to flash on the NavRanger. Note the previously saved parameters will be overwritten.

Table 2.3.1.2: Measurements

Name	Units	Description
Filtered Distance	Inches*	The output of a single pole low pass filter with a cutoff frequency of 0.5 Hz
Std Dev	Inches*	Sqrt [E2 [(E1[x] - x[t])^2]], where E[x] is the Filtered Distance and E2[v] is the low pass filtered value of v.
Current Distance	Inches*	The distance sampled at 10 / sec
Width	Raw TDC time 65ps/LSB	Indicates the strength of the returned signal or the dispersion of the pulse due to an angled surface.
Cycle Time	Microseconds	Time between internal data updates.
Temperature	Deg F	Left field shows the on-board sensor and Right shows the IMU.
Laser Power	ADC Counts	The raw ADC value of the rectified and filtered laser voltage. This is somewhat temperature sensitive and can vary from 9000 to 8000 when the laser is on.
Supply Voltage	Volts	The measured input power supply voltage.

Accelerometer	Raw IMU Value 32767/(2 G)	X, Y, Z low pass filtered accelerations. Filter cutoff frequency is 94Hz. Note these are updated at 400Hz.
Gyroscope	Raw IMU Value 32767/(250 deg/sec)	X, Y, Z low pass filtered rotation rates. Filter cutoff frequency is 98Hz. Note these are updated at 400Hz.
Magnetometer	Raw IMU Value 4095 / (1229 uT)	X, Y, Z Magnetometer measurements. Note these are updated at 400Hz.

*With default Cal parameters - See Table 2.3.1.3

Table 2.3.1.3: Parameters

Name	Units	Description
Scan Rate	RPM	The rotation rate of an optional scanner.
Far Gain	DAC Counts	The relative gain for far distances.
Close Gain	DAC Counts	The relative gain for close distances.
Threshold	DAC Counts (0-65535)	The relative received threshold.
APD Voltage	ADC Counts (0-65535)	Setpoint to the ADC voltage control loop. Max 30K ->240V, 20K->160V
Cal Dist Gain	distUnits (TDCRef / TDCCount)	RawDistance = TDCCount * CalDistGain / TDCRef + CalOffset
Cal Width Gain	distUnits (TDCRef / TDCCount)	RelDistance = RawDistance - Width * CalWidthGain / TDCRef
Cal Offset	distUnits	Distance = RelDistance + CalOffset

distUnits is the desired distance units. Inch by default.
TDCRef = (65ps / clockPeriod). Measured by the TDC and is approximately 3840

Note "Toggle Save Data" can be very useful for examining data from the NavRanger. The .CSV file can be read in Excel or like program for graphing the data. The top row in the file has labels for the columns and each row represents one measurement. Distance and width measurements run at about 20K per second and the IMU values update at about 400 per second. In the file, the IMU values are duplicated on each row until they are updated.

2.3.2. Graph Window

The Graph window shows a polar plot where the angle changes clockwise with time and the radius is the distance. A full circle occurs in 100ms or one scan which ever is first. Figure 2.3.2 shows the Windows version of the Graph Window. The Mac version should be nearly identical. The Max Width and Max Distance parameters control the scaling of the graphs.

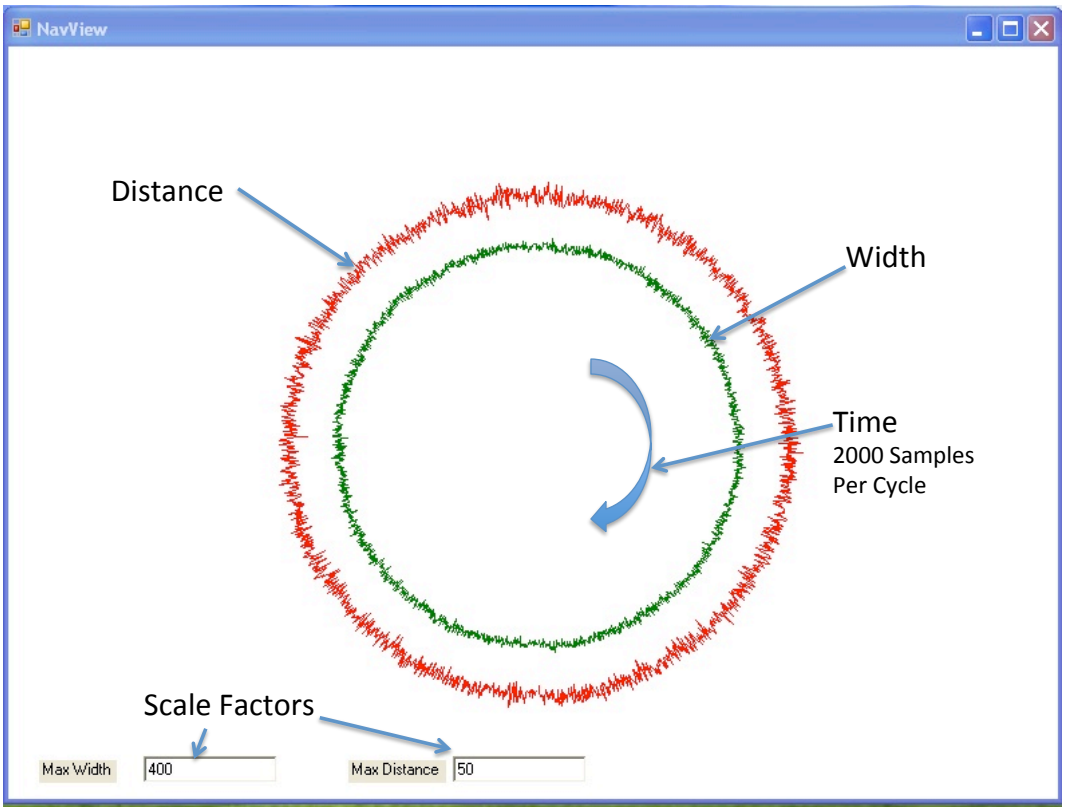


Figure 2.3.2 NavRanger Graph Screen

3. HARDWARE

3.1. Mounting

Figure 3.1 shows the NavRanger mounting, laser output, and sensor locations. The four main mounting holes are for #6 screws, 1/4 inch from the corners of the board. There is an additional mounting hole between the laser and photodiode for a #2 screw. This may be required in high vibration environments. Both the photodiode and laser need to be secularly mounted by a shroud around the devices. Note that the laser emitter is not in the center of the laser package.

Do not connect the photodiode case to chassis ground. The photodiode (Receiver) case is tied to the board ground. Use an insulated mount. See the NavRanger web page for example mounting drawings.

3.2. Optics

For best performance, a collimator needs to be added to the laser and a lens needs to be added to the photodiode receiver . Additionally, a 905nm optical bandpass filter must be added to the receiver for operation in sunlight.

Collimation Lens:	http://www.thorlabs.us/thorproduct.cfm?partnumber=A110TM-B
Receiver Lens	http://www.thorlabs.us/thorproduct.cfm?partnumber=LA1951-B
Filter:	http://www.thorlabs.us/thorproduct.cfm?partnumber=FL905-25

Note that both lenses need to be positioned so the focal point is on the device surfaces indicated in Fig 3.1 and the flat side of the piano convex lens is toward the receiver.

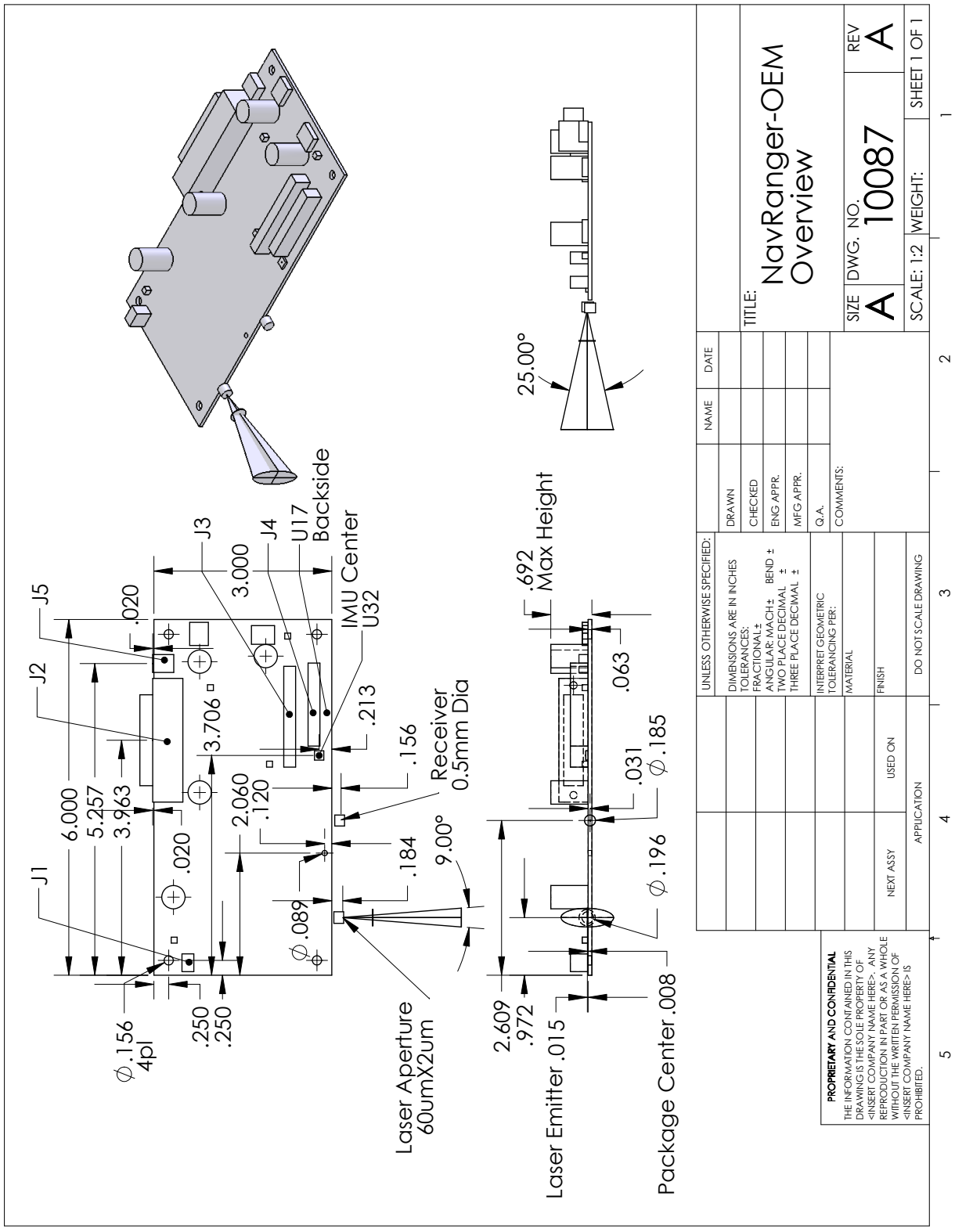


Figure 3.1: NavRanger mounting and component locations.

3.3. Main Interface J2

Table 3.2 shows the pinout of the 25 pin D connector J2. This connector supplies power to the board and provides robust IO connections including the CAN bus and RS-232.

3.3.1. Power

NavRanger requires power from 9V to 28V applied to pin 1 positive and pin 4 ground. The current required at 12V is 0.32A.

A separate power supply circuit is available to power the Auxiliary connector. Note that the AuxGnd line is connected to GND through a one-ohm resistor. AuxGnd and Gnd must be connected together at one point in the system.

Table 3.2.1: Power Pins

Pin	Name	Description	I/O	Min	Max	Units
1	Power	Main Supply for the board 3.3 Watts required Max ripple must be <5%	In	9	28	Volts
4	GND	Board supply ground. Max and Min Referenced to this input.	In			
16	GND2	Board Ground This is connected to GND	In			
2	AuxPow1	Pass through 1A PTC to pins 27,29, and 31 of J3	In	0	48	Volts
14	AuxPow2	Pass through 1A PTC to pins 27,29, and 31 of J3 Connected to AuxPow1	In			
3	AuxGnd1	Pass through to 28,30 and 32 of J3. 1Ohm connection to board ground (GND).	In	-0.2	0.2	Volts
15	AuxGnd2	Pass through to 28,30 and 32 of J3. 1Ohm connection to board ground (GND) Connected to AuxGnd1	In	-0.2	0.2	Volts

3.3.2. Analog and Digital I/O

The Main interface connector has one analog output, one optically isolated digital output, and one optically isolated digital input.

In the standard embedded software, the Analog Output is the current distance, where 5 volts is 50 meters. Different scaling or usage can be obtained by changing the calibration Parameters or embedded software.

Table 3.2.2: Analog and Digital I/O Pinout

Pin	Name	Description	I/O	Min	Max	Units
18	AO4	Analog Output 4 Output impedance 120 ohms	Out	0	5	Volts
5	AO4Gnd	Ground for AO4. Connected to GND	In			
6	AI1	Analog Input 1	In	0	3.3	Volts
17	AI1Gnd	Ground for AI1 Connected to GND				
19	PDO1	Digital Output1 Positive Isolated NPN collector Max Current 20mA	Out	-48	48	Volts
7	NDO1	Digital Output1 Negative Isolated NPN Emitter Max Current 20mA	Out	-48	48	Volts
20	PDI1	Digital Input 1 Positive Isolated Diode in series with 1K Max 18V between PDI1 and NDI1	In	-48	48	Volts
8	NDI1	Digital Input 1 Negative Isolated Diode in series with 1K Max 18V between PDI1 and NDI1	In	-48	48	Volts

3.3.3. RS-232 Port

The RS-232 port is optional and is open circuit in the GPS configuration. The RS-232 port uses standard levels and the default communication rate is 56K Baud. Note that the current embedded software uses only the RX and TX lines.

Table 3.2.3: RS-232 Port Pinout

Pin	Name	Description	I/O	Min	Max	Units
10	/RSTX	RS-232 Transmit Data Only with RS-232 Option	Out	-12	12	Volts
22	/RSRX	RS-232 Receive Data Only with RS-232 Option	In	-18	18	Volts
9	RS232Gnd	Ground for RS232 Port				

		Connected to GND				
11	RSRTS	RS-232 Request To Send Only with RS-232 Option Not Currently Used	Out	-12	12	Volts
23	RSCTS	RS-232 Clear to Send Only with RS-232 Option Not currently used	In	-18	18	Volts
21	RSShield	Shield for RS-232 Connected to GND				

3.3.4. CAN Bus

The CAN bus can provide communication up to 1 Mb/second. The current version of the embedded software does not use the CAN bus.

Table 3.2.4: CAN Bus

Pin	Name	Description	I/O	Min	Max	Units
25	CANH	CAN Bus High	In/Out	-2	7	Volts
13	CANL	CAN Bus Low	In/Out	-2	7	Volts
12	CANGnd	CAN Bus Ground Connected to GND				
24	CANShield	Shield for CAN bus Connected to GND				

3.4. Auxiliary Interface J3

J3 provides power, analog, and digital signals for scanning or simple robot control applications. Note that these signals do not have any static or over voltage protection.

3.4.1. Power

AuxSupply can provide power for a scanner or motor drive for a small robot. Max current is 1A and is limited by a self-resetting PTC device.

Table 3.3.1: Power

Pin	Name	Description	I/O	Min	Max	Units
27, 29, 31	AuxSupply	Auxiliary Power Supply 1A Max	Out			

28, 30, 32	AuxGnd	Ground for AuxSupply Connected to GND through 1 Ohm	Out			
22, 26	3.3V	Board 3.3V 100mA, 400uF Max	Out	3.2	3.4	Volts
21, 25	GND	Board Ground	Out			

3.4.2. Analog In

Analog inputs are 12 bit 0 to 3.3V board ground referenced.

Table 3.3.2 Analog Inputs

Pin	Name	Description	I/O	Min	Max	Units
23	AI_2	Analog Input 2	In	0	3.3	Volts
19	AI_2Gnd	Ground for AI_2 Connected to GND	In			
24	AI_3	Analog Input 3	In	0	3.3	Volts
17	AI_3Gnd	Ground for AI_3 Connected to GND	In			

3.4.3. Digital IO

The digital IO signals attach to some of the intelligent peripherals on the MCF52255. They can be programed to be general purpose or be controlled by their specified peripherals.

Table 3.3.2 Digital input and output pins

Pin	Name	Description	I/O	Min	Max	Units
2, 4, 6, 8	IRQ1-7	Interrupt Input or General purpose digital I/O	In/Out	0	3.3	Volts
10, 12	PTI0-1	General purpose digital I/O	In/Out	0	3.3	Volts
14, 16	DT0-1	32 Bit Timer or General purpose digital I/O Includes 274 Ohm Pull-up to 3.3V	In/Out	0	3.3	Volts
18, 20	PWM5, PWM7	PWM IO or General purpose digital I/O	In/Out	0	3.3	Volts
1, 3, 5, 7, 11, 13, 15	GND	Digital IO Ground Connected to GND	In/Out			

3.5. Laser interlock J1

J1 is the laser scanning interlock connector. A scanning interlock may be required to make the system a Class 1 laser product when a laser collimator is used with the NavRanger. The board comes with pins 1 and 2 shorted making it possible for the laser to be turned on any time power is supplied to the board. This jumper must be removed to use the interlock. A scanning interlock will allow current to flow from pin 2 to 1 only when scanning and no current to flow when the laser is stationary.

This interlock must be testable, such that the laser should be command to turn On and verified that there is no laser power when Interlocked Off.

Table 3.4 Laser Power Interlock

Pin	Name	Description	I/O	Min	Max	Units
1	LaserPowIn	Power to the laser diode power supply 100mA Max	In	0	28	Volts
2	LaserPowOut	Power from internal supply	Out	0	28	Volts

3.6. ColdFire BDM J4

The ColdFire BDM connector J4 is for updating the embedded software on the NavRanger. This requires the purchase of a Background Debug Module. PE Micro (www.pemicro.com) has a good collection of BDM modules and debug software. See the following link for more information:

http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=USBMLCF

3.7. Digi XBee Interface U17

The XBEE interface U17 is made for Digi XBee-Pro wireless modules. The XBee Pro S2B was tested, but other modules with the same pin-out will work. See the following link for more information:

http://www.digi.com/pdf/ds_xbeezbmodules.pdf

Note that the connector for U17 is on the backside of the board and is not installed in the standard NavRanger. The connector can be installed as a special order or installed by the user. The connector is available from several distributors like Newark.
It is a Samtec P/N: MMS-110-01-L-SV.

Table 3.4 XBee Interface

Pin	Name	Description	I/O	Min	Max	Units
1	3.3V	Power for the Radio 300mA Max	Out	3.2	3.4	Volts
10	GND	Connected to board Ground	Out			
2	SDOut	Serial Data Receive TTL	In	0	3.3	Volts
3	SDIn	Serial Data Transmit TTL	Out	0	3.3	Volts
12	/CTS	Clear to send TTL RS-232 Not currently used in software	Out	0	3.3	Volts
16	/RTS	Request to send TTL RS-232 Not current used in software	In	0	3.3	Volts
5	/Reset	Active low reset Not currently used in software	Out	0	3.3	Volts
9	/Sleep	Sleep command	Out	0	3.3	Volts

3.8. TTL Serial GPS Interface J6

J6 is an optional connection for a GPS module that uses a 3.3V TTL level serial interface for communication. The connection can supply 3.3V up to 100mA for the GPS. The connections also has a separate digital signal for a GPS wakeup signal. Note that the current version of the embedded software does not control the GPS interface.

Table 3.7 GPS Interface J6

Pin	Name	Description	I/O	Min	Max	Units
1	3.3V	Power for the GPS 100mA Max	Out	3.2	3.4	Volts
2	GND	Connected to board Ground	Out			
3	URX	Serial Receive Data	In	0	3.3	Volts
4	UTX	Serial Transmit Data	Out	0	3.3	Volts
5	GWake	GPS Wakeup signal or General purpose Digital I/O	Out / In	0	3.3	Volts

4. SOFTWARE INTERFACE

Projects for XCode and Visual C++ can be found in the NavRangerClient directory in the NavRangerMac or NavRangerWindows directories. You can get XCode from the Mac App Store and Visual C++ from http://www.visualstudio.com/en-us/downloads/. Select Visual Studio Express 2013 for Windows for the free version.

4.1. Using the Basic Application Interface

This section describes how to get range, signal strength, and IMU data from the NavRanger.

The NavComponets directory contains the source files required for communication. Copy this directory to your project and include the following files in your project:

CUSBNavCmdPipe.cpp
CUSBNavDataPipe.cpp
CUSBPort.cpp
CUSBPipe.cpp
NavDriveIO.cpp

In your project or makefile set the NavComponets to be an include and a library directory. Set libusbK.lib to be a link dependency.

CNavDriveIO is the application's interface to the NavRanger. It performs most of the calculations to get and condition data for the application. Your custom application should communicate through CNavDriveIO. If you need to perform additional calculations on the NavRanger data you should modify or replace CNavDriveIO. See tables 4.1 to 4.3 for a descriptions of the methods used in the procedures below.

1. **Perform the following once to initialize the NavRanger interface.**

    1.1. Include NavDriveIO.h
    1.2. Create one instance of CNavDriveIO. Note that if you are using managed code in Windows, you must create a static instance of CNavDriveIO. In this case the instance is called `gNavDriveIO.`
    1.3. Call `CNavDriveIO.Initialize (int msPerSample) where msPerSample = 0;`

2. **Establish a connection to the NavRanger, load the parameters, and turn-on the laser.**

    2.1. Call `int gNavDriveIO.UpdateContiniousData ()` until it returns an integer greater than zero.
    2.2. Call `gNavDriveIO.RestoreParams ().`
    2.3. Call `bool gNavDriveIO.HandleStatusPipe ()` until it returns True.
    2.4. Call `gNavDriveIO.LaserOn ()`
    2.5. Call `bool gNavDriveIO.HandleStatusPipe ()` until it returns True.

3. Repeat the following to get range, signal strength, and IMU data.

3.1. Call `int gNavDriveIO.UpdateContiniousData ()` until it returns an integer greater than zero.
3.2. Call `int gNavDriveIO.GetDataSize ()`.
Use the returned value as a maximum index-1 into the data arrays.
3.3. Call `double* gNavDriveIO.GetDistArrayPtr ()`.
3.4. Call `double* gNavDriveIO.GetWidthArrayPtr ()`.
3.5. Call `T_IMUData* gNavDriveIO.GetIMUArrayPtr ()`.
3.6. Call `UInt16* gNavDriveIO.GetDeltTimeArrayPtr ()`.

4. Verify that the laser is ON.

4.1. Call `bool gNavDriveIO.LaserIsOn ()`. If the laser turns off you can assume that the power to the NavRanger failed and you should reload the parameters, and turn-on the laser as described in the section above.

5. Set a parameter.

5.1. Call `void gNavDriveIO.SetCloseGain (int valueToSet)` with a value from 0 to 2^8-1 to set the close gain. Note that there are many set functions with the same format. A list of these functions can be found in NavDriveIO.h and the units for these functions can be found in Table 2.3.1.3.
5.2. Call `bool gNavDriveIO.HandleStatusPipe ()` until it returns True.

Table 4.1: CNavDriveIO Startup and Shutdown Methods

Method Name In CNavDriveIO	Returns	Params	Description
Initialize	void	int Unused	Sets up the client to connect to the NavRanger when Attached
`Shutdown`	void	None	Stops communications and closes the USB port

Table 4.2: CNavDriveIO Command Methods

Method Name In CNavDriveIO	Returns	Params	Description
`HandleStatusPipe`	bool	None	Handles sending a command and updating local status data. True indicates the previous command has been processed and all the local parameter values have been updated.
`RestoreParams`	void	None	Sends a command to copy parameter data from NavRanger's eeprom. `HandleStatusPipe` should be called after sending this command until it returns True.
`LaserOn`	void	None	Sends a command to turn on the Laser. `HandleStatusPipe` should be called after sending this command until it returns True.
`LaserOff`	void	None	Sends a command to turn off the Laser. `HandleStatusPipe` should be called after sending this command until it returns True.
`SetAPDVoltage`	void	int value	Sends a command and associated value to set the Avalanche Photo Diode voltage. Max 30000 gives 240V, 20000 gives 160V or Vout = ((value - 20K)/125)

			+160. `HandleStatusPipe` should be called after sending this command until it returns True.
`Set....`	void	int value	There are many Set.... commands that send an integer value to the NavRanger. These all work the same way as `SetAPDVoltage`. See NavDriveIO.h for a complete list.

Table 4.3: Basic CNavDriveIO Continuous Data Methods

Method Name In CNavDriveIO	**Returns**	**Params**	**Description**
`UpdateContiniousData`	int	None	Updates the USB connection status and gets data. Returns a value greater than zero if a data block is available. Must be called at greater than 10Hz to avoid loosing data with default block size. In the current version a data block is a complete scan or 2000 data points. The maximum number of points is defined as `kMaxScanSize` in CNavDriveIO.h. It may be appropriate to have a timeout invoke special actions that may be required if communication with the NavRanger is lost.
`GetDataSize`	int	None	Returns the number of samples in the receive buffer. Use the value returned by this function as a maximum index+1 for the arrays retrieved using the functions described below.
`GetDistArrayPtr`	Pointer to double array	None	Gets a pointer to an array of distances in units determined by the calibration values. Use GetDataSize to determine the number of elements in the array. Access this data using the standard 'C' array operators or other pointer arithmetic.
`GetWidthArrayPtr`	Pointer to double array	None	Gets a pointer to an array of doubles with relative signal strength data. The values are the width of the received light pulse in units of 65ps. Use GetDataSize to determine the number of elements in the array.
`GetIMUArrayPtr`	Pointer to array of `T_IMUData` structures.		Gets a pointer to an array of structures that contain measurements from the Inertial Measurement Unit. `T_IMUData` structure is an array of signed 16 bit integers. Table 4.4 contains a description of elements in this array. Note that constants for indexing this data are defined in CNavDataPipe.h. Use GetDataSize to determine the number of elements in the array.
`GetDeltTimeArrayPtr`	Pointer to array of UInt16	None	Gets a pointer to an array with the time since the previous data sample in microseconds. Use GetDataSize to determine the number of elements in the array.

Table 4.4: T_IMUData indices.

Index	Value	Units	Sampling
0 1 2	Acceleration X Acceleration Y Acceleration Z	Raw IMU Value 32767/(2 G)	Low pass filtered accelerations. Filter cutoff frequency is 94Hz. Note these are updated at 400Hz.
3	RawTemperature	T in Deg C= (RawTemperature * 0.1162) - 27	Updated at 400Hz.
4 5 6	Gyro X Gyro Y Gyro Z	Raw IMU Value 32767/(250 deg/sec)	X, Y, Z low pass filtered rotation rates. Filter cutoff frequency is 98Hz. Updated at 400Hz
7 8 9	Magnetometer X Magnetometer Y Magnetometer Z	Raw IMU Value 4095 / (1229 uT)	X, Y, Z Magnetometer readings. Note these are updated at 200Hz

4.2. Exception Handling

The NavRanger Client interface uses C++ exceptions to notify the application of failures. The unmodified version of CNavDriveIO prints the error result to the console. If the application can handle the exceptions at a higher level, then the catch sections in `CNavDriveIO.Initialize` and `UpdateContiniousData` must be modified to throw the exception to a higher level.

4.3. The NavRanger Client Application

The directory NavComponents contains files for receiving and interpreting data from the NavRanger. The main classes, CUSBPort, CUSBPipe, and CNavDriveIO and their derived classes are shown in Fig. 4.1. The CUSBPipe classes work with CUSBPort to get data and manage the USB port. You need to change the CUSBPipe objects if you change the data sent by NavRanger. You will probably never need to change CUSBPort. There needs to be one CUSBPipe for each USB endpoint. Derived classes CUSBInteruptPipe and CUSBIsocPipe support Interrupt and Isochronous USB endpoints types. The classes CUSBNavCmdPipe and CUSBNavDataPipe provide specific data handling for the NavRanger. CUSBNavCmdPipe is a CUSBInteruptPipe and CUSBNavDataPipe is a CUSBIsocPipe.

CNavDriveIO instantiates three pipes, mCommandOutPipe, mStatusInPipe, and mDataPipe. The instances mCommandOutputPipe, and mStatusInPipe are CUSBNavCmdPipes for sending commands and getting status. The instance mDataPipe is a CUSBNavDataPipe, which gets the continuous measurements such as distance and IMU values.

CNavDriveIO can work with two threads as shown in Fig 4.2. One is the Continuous data thread, handled by NavView and the other is the asynchronous command/status thread handled by NavCommand.

Each periodic timer tick in NavView calls UpdateContiniousData (), which returns the number of records received. Functions are provided to get the size of the data available (GetDataSize ()) and get pointers for accessing the continuous data (Get...ArrayPtr()). This data owned by the CNavDriveIO instance and stored as a 'C' array with the number of elements specified by GetDataSize (). Data access using these pointers must be after UpdateContiniousData () and be in the same thread. Note that UpdateContiniousData () also updates member variables that are used by the command/status thread through the atomic Get...() methods. A Mutex blocks access to this data until UpdateContiniousData () finishes. The method UpdateContiniousData () should be modified as required to provide application specific processing of NavRanger data.

Periodic ticks of NavCommand call HandleStatusPipe (), which checks if mStatusInPipe has data and gets the data if it is available. HandleStatusPipe () returns true when the response number matches the last sent command indicating the parameter values represent what was requested. If the response number does not match, mStatusInPipe.StartNextRead () is executed again unless there is a timeout. If a timeout occurs HandleStatusPipe () performs the write of a dummy command with a new command number.

CNavDriveIO provides a set of methods to sending commands for Actions and setting Parameters. The setting commands (Set... ()) call the method WriteCommand (), which sends a USB Interrupt message using mCommandOutPipe. WriteCommand (), also initiates a StartNextRead () in mStatusInPipe, so the next HandStatusPipe () call can get the new state of NavRanger.

Parameters are updated when HandleStatusPipe () is called and data is available indicated by a True return value. Methods for getting measurements and parameters can be called at anytime, but may not return until UpdateContiniousData () or HandleStatusPipe () complete.

4.3.1. Windows specific details

Note that the file libusbK.dll must be in the same directory as the executable.

In the Windows version, CNavDriveIO is instantiated as gNavDriveIO on the heap in Main.cpp and Initialized as shown in Fig. 4.2. The application consists of two .Net Forms. These Forms are setup and run in two separate threads. See Figure 4.2 where main sends the message "Run" to the NavControl and NavData Forms.

The Form NavControl waits for two types of events, a timer Tick and button Click. When the timer expires it calls ControlTimer_Tick (), which updates the fields and sets the button states. Updating the fields consists of calling the various Get functions in gNavDriveIO to get the measurements and filling in the Text boxes. The Form NavControl also calls the gNavDriveIO HandleStatusPipe. () If the count identifiers match, HandleStatusPipe updates the mParametersValues structure from the data in mStatusInPipe and returns True. When the NavControl Form gets True back from HandleStatusPipe it populates the parameter text boxes with the values from mParametersValues.

The Form NavView has a periodic timer that calls IsocEvents_Tick () each tick. This calls the NavDriveIO UpdateContinious () method and updates the plot if more than two data points were found.

4.3.2. Mac OSX specific details

The Mac client is a mixed ObjectiveC / C++ application. AppController.m does most of the work and interfaces with NavDriveIO. The method TimeFireMethod () is called periodically at 10ms intervals. TimeFireMethod () first calls UpdateContiniousData () and computes the filtered measurements and updates the plot if data is found. TimeFireMethod then calls HandleStatusPipe () and updates the parameter fields if it receives an update. The TimeFireMethod () also updates the buttons based on the current parameter values.

5. EMBEDDED SOFTWARE

The embedded software is available by special request from Integrated Knowledge Systems. Modification instructions are available in that package.

Note that a Coldfire Background Debug Module is required to update the NavRanger embedded software. See section 3.4. Additionally, the GCC complier is required to build the embedded application.

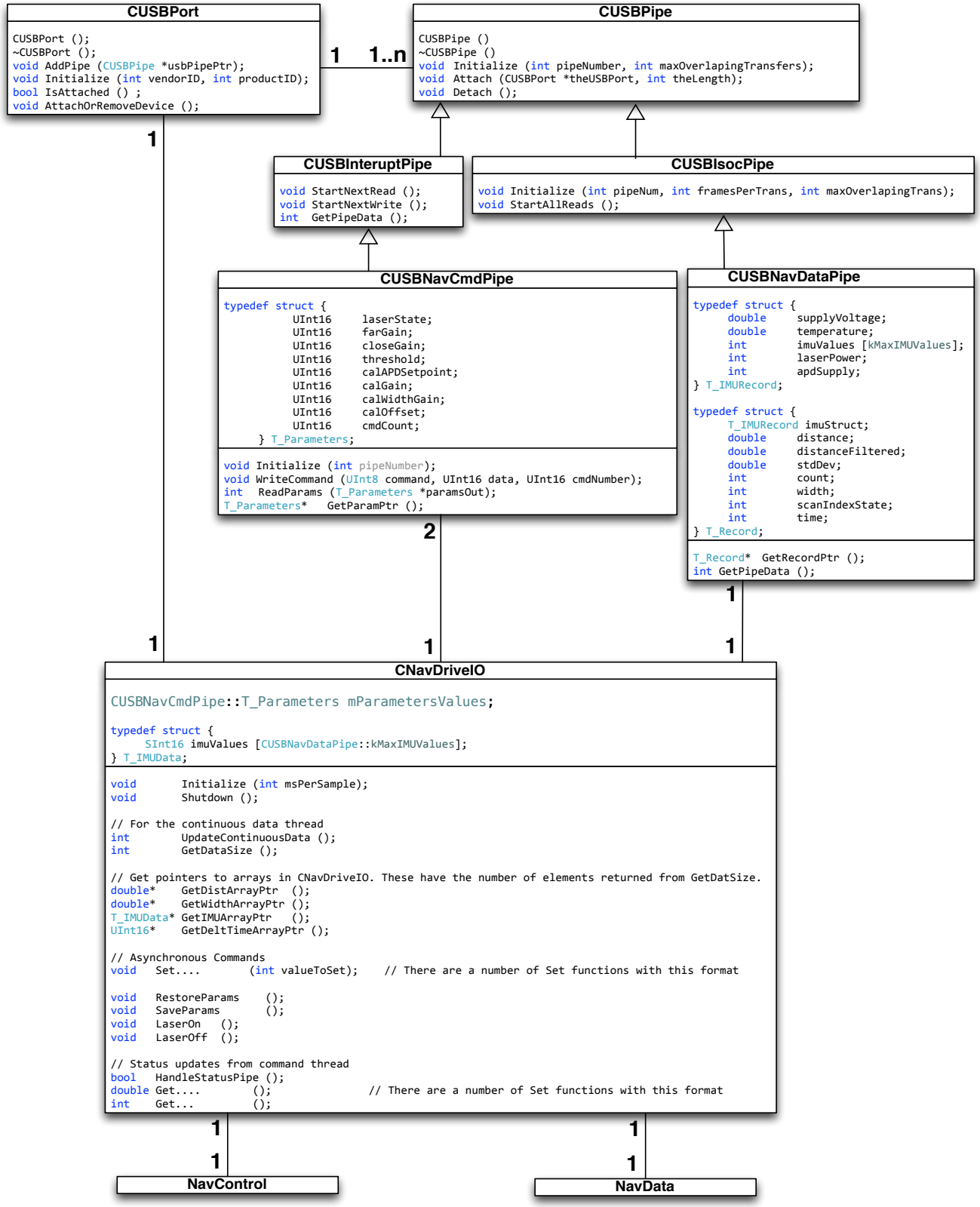


Figure 4.1 NavClient Class Diagram

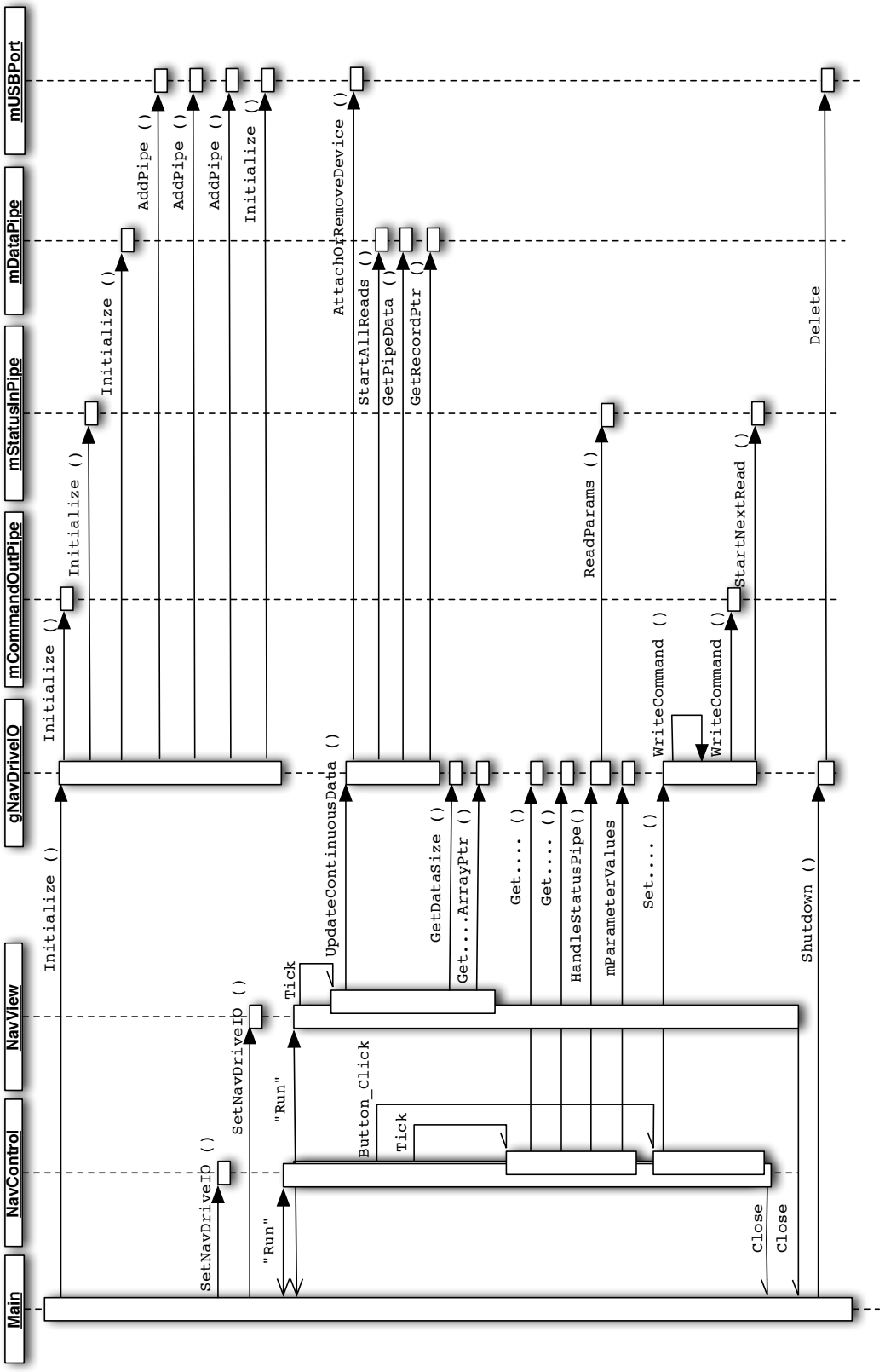


Figure 4.2 NavClient Sequence Diagram