3D at Depth

3D at Depth, Inc

TCP/IP Packets for CCI Communication

Version C.7

Nov 2021

Change Record

Version	Date	Notes	Author
C.1	11 Feb 2019	Initial Draft	J. Jeong
C.2	28 Jun 2019	Fixed pulse intensity size from UINT32 to UINT16. Point data size is 50.	D. Florin
C.3	11 July 2019	Checksum added	J. Jeong
C.4	26 July 2019	Checksum char to UINT bugfix	J. Jeong
C.5	18 Sep 2019	The Ack and Resend values were not correct; fixed	D. Florin
C.6	15 July 2021	Added InsTimePack, NavigationLong definitions. Fixed pulse time to make it clear that is has units of microseconds, not seconds. Added more on calculation of time of pulse.	D. Florin
C.7	22 Nov 2021	Small Text Changes and cleanup. Updated table (was off).	D. Florin

Table of Contents

1 Introduction

3D at Depth SL sensors deliver world class 3D point clouds. The real time scanning data can be delivered through TCP/IP protocol to any external equipment. This document explains how to communicate with the system and how to parse the data. The goal is for the 3D at Depth system to provide 3D data to the CCI (Command and Control Interface) software acquisition package to enable real time mapping in global coordinates.

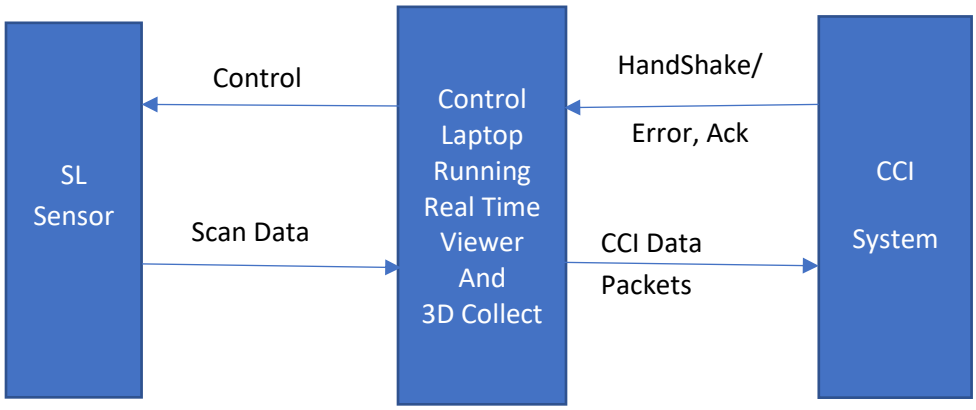
Offline import of logged data to CCI software users is also requested. For applications where CCI is not available online this will allow 3D data to enter the user's processing environment and follow a consistent workflow.

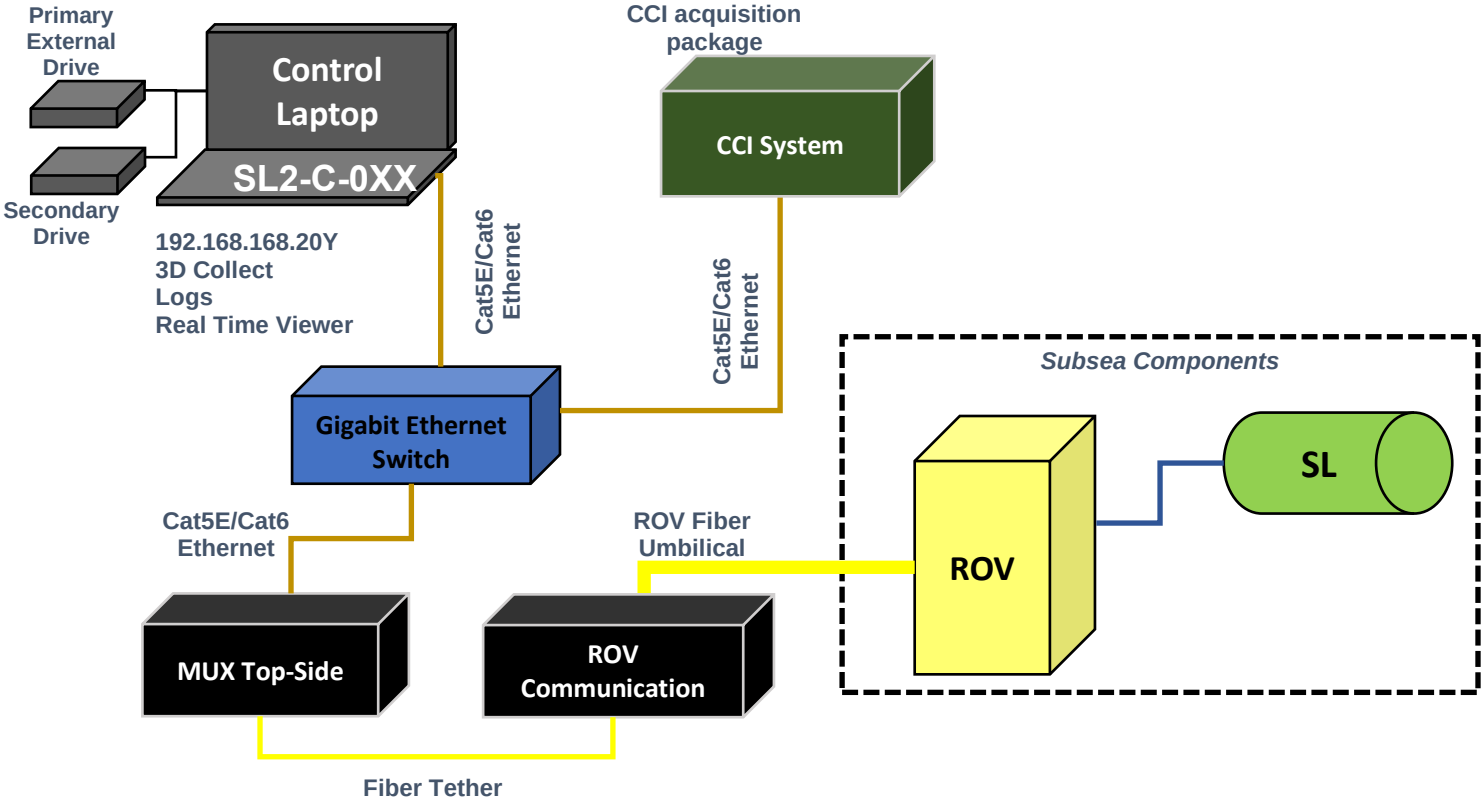
2 Scope

This document describes the TCP/IP software interfaces to the 3D at Depth Subsea Lidar topside system. It details the necessary communications protocols between the topside Real Time Viewer and the CCI software acquisition package. This document is considered a *working* document, and a specific version will be released when agreed to with software users.

3 Ethernet Interface

Communication packets between the SL Sensor and the Real Time Viewer (RTV) are sent and received via the TCP/IP Ethernet Protocol. The RTV opens a TCP/IP port, and waits for external communication as a server. The RTV Laptop has a static IP address for the hard line used to talk to the SL sensor and connection over this network is allowed. The same subnet should be used (192.168.168.xxx) for any network connections. A particular send-receive, handshake acknowledgment standard is also provided per packet usage. Simplistic diagrams are shown below.





3.1 Number Representations

The byte order for multi-byte packet fields will comply with the little-endian format (Intel). The table below describes the number of bytes per types. All structures and typedefs must adhere to a data pack, single byte scheme, as some packets may align to 1 byte size.

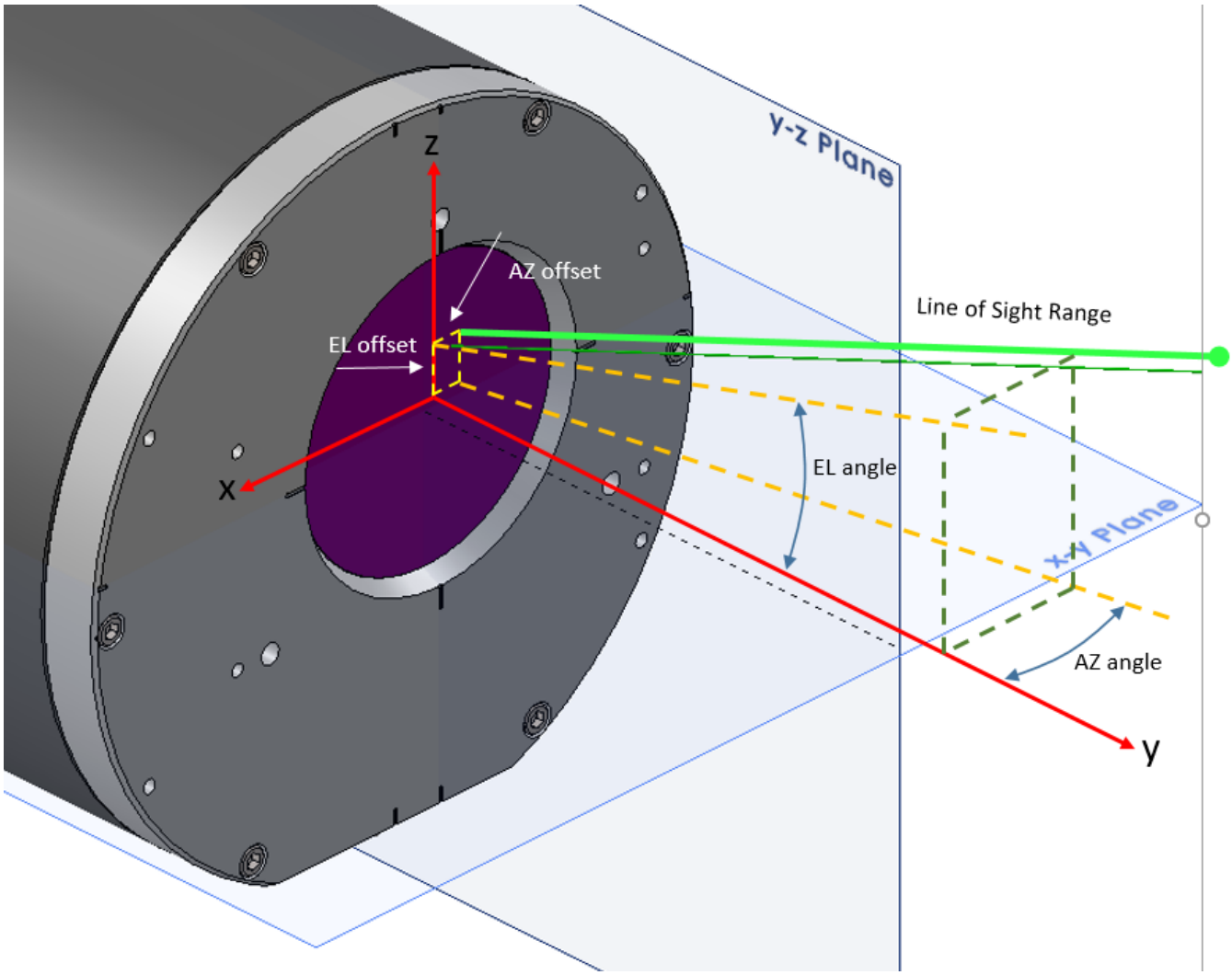
Type	Number of bytes
char, unsigned or signed 8 bit integer	1
short, unsigned or signed 16 bit integer	2
long, unsigned or signed 32 bit integer	4
float, 4 byte IEEE floating point representation	4
double, 8 byte IEEE floating point representation	8

Please reference the "utils.h" listed in the appendix for C++ definitions of the various types.

C++ type	Definition	Number of bytes
INT8	signed char	1
INT16	signed short	2
INT32	signed int	4
UINT8	unsigned char	1
UINT16	unsigned short	2
UINT32	unsigned int	4
INT64	signed __int64	8
UINT64	unsigned __int64	8

4 RIAAT Data Format

The 3D at Depth SL sensor utilizes a pulsed laser to take 3D range measurements. The laser emits a pulse of light to measure Line of Sight (LOS) range. While scanning in an Azimuth (AZ) and Elevation (EL) fashion, each LOS range is measured along with the current AZ-EL position, AZ-EL offsets, Intensity, and time. Each LOS range is measured from the front of the output window on the SL sensor as shown in the diagram below.



The Range, Intensity, AZEL Angle, and Time (RIAAT) data format is used to record the above measurements. In addition, advanced processing allows for multiple LOS range measurements for each AZ-EL position. Consider the case where some of the light is clipped on one surface and then hits another more distant surface. In this case, two valid LOS returns would be measured. The SL sensor provides five LOS range measurements for each laser pulse. Such a capability allows for expanded use typically in cloudy water.

The Range data is measured from the glass at the front of the sensor. In order to use the motion data, the relationship between the sensor and the inertial navigation unit is needed. The values of “Offset” and “Rotation” in the header provide that relationship so that the data can be placed in the navigation coordinate system. The offset is the xyz vector to add to the sensor to go from the sensor to the INS sensitive point. The Rotation is the

matrix needed to rotate from the sensor coordinate system into the INS system. In order to use the navigation data that is part of the packet, the data must be transformed into the INS coordinate system. Time is used to match when the point was measured to the INS positional data. The equation to change range into an xyz real world coordinate is as follows:

$[X_w,Y_w,Z_w]$ = [Rotation matrix] *[x, y, z] + Offset

Where [x, y, z] is calculated by

dN = 1/sqrt(1+tan(Az)*tan(Az) + tan(El)*tan(El))

x = range * dN * tan(Az) + Az_Offset

y = range * dN

z = range * dN * tan(El) + El_Offset

Range, Az, El, Az_offset and El_offset are part of the Scan Line Data packet.

The RIAAT data is arranged into a complete scan line of data, or grouping of data depending on the scan pattern. The actual size and format of a RIAAT data packet is configurable, based on the current SL configuration. In particular, the LOS range, number of measurements per LOS, and the number of AZ points determine the actual data size and format of the RIAAT scan line data. To conserve memory, the RIAAT data is stored in a binary format. The following table describes the general RIAAT data structure.

Range Intensity Angle Angle Time data format (binary)		
Item	**Bytes**	**Data Type**
ScanCFG structure, meta data	*	*
Timestamp		
Timestamp year (true year)	2	1 UINT16
Timestamp month (1-12)	1	1 UINT8
Timestamp day	1	1 UINT8
Timestamp days since Jan 1	2	1 UINT16
Timestamp hour	2	1 UINT16
Timestamp minutes	1	1 UINT8
Timestamp seconds	1	1 UINT8
Timestamp nano seconds	4	1 UINT32
Laser Pulse Scan LineData 1 (1 to m pulses per scan line)		
AZ, Cross track angle (deg)	4	1 float32
EL, Forward track angle (deg)	4	1 float32
AZ, Cross track offset from center (m)	4	1 float32
EL, Forward track offset from center (m)	4	1 float32
Pulse time offset from timestamp (usec)	4	1 float32

LOS Range 1 (from glass front) meters	4	1 float32
…		
LOS Range 5 (from glass front) meters	4	1 float32
Intensity LOS 1 / peak of signal	2	1 UINT16
…		
Intensity LOS 5 / peak of signal	2	1 UINT16
Laser Pulse Scan Line Data 2		
…		
Laser Pulse Scan Line Data m		

It should be noted that the SL sensor synchronizes its clock to the INS clock subsea once per second. Therefore, all times reported in the data files are SL sensor time, but this time is closely synchronized to the subsea INS time.

Time of a pulse is calculated as a combination of the timestamp on the individual pulse in microseconds (`time_offset_usec`) which is added to the timestamp on the header (sTimeStamp) for the time that the pulse occurred. This is the outgoing time to correct for motion.

5 CCI Packet Definitions and Protocol

The Real Time Viewer application on the 3D at Depth laptop will convert standard RIAAT packets into custom packets to send directly to the CCI software acquisition package for data mapping in real world coordinates in the user application.

The packets that Real Time Viewer produces are listed below

Packet Types		
Packet	Size Bytes	
Header	63	Defines scan line size
ScanLineData	*	One Scan line of data
Status	6	

5.1 Packet Header

The packet header fields are described in the table below. A Packet Count is embedded in the header and will increment on each Packet sent. This can be useful in determination of "stale" data. Number of Scan Lines is the number of Scan Line Data Packets that are going to be sent before the next Header packet is sent. For a fast scan image type, the header will define the number of lines that are in the scan image. For motion, the data is continuous and the size is more arbitrary. Data Size defines the size of a Scan Line Data Packet that follows. Since the number of points in a scan line can and do change from time to time, the overall size will also change. The size could also be calculated from the first part of the Scan Line Data Packet as well which is the recommended processing method to determine read size.

Header Packet		
Header Item	Value	Bytes
Alignment Code	0xC0DE	2 (1 UINT16)
Packet Count	counts	2 (1 UINT16)
Version	2	2 (1 UINT16)
Number of Scan Lines	N	2 (1 UINT16)
Data Size (bytes)	varies	4 (1 UINT32)
Offset (float[3])		12 (3 float)
Rotation Matrix (float[9])		36 (9 float)
Reference Type		1 (1 UINT8)
Checksum		2 (1 UINT16)

5.2 Scan Line Data

A Scan Line Data Packet is generated for each scan line. Scan Line Data has header information that applies to the pulse data. nScanLineNumber is a count of the current scan line and it will increment on each one until the count gets to the number indicated in the Header Packet, then it starts over. sNavSet is the navigation data captured at the start, middle and end of a line scan, however, depending on network lag, the data may apply to a different scan line that what is in this packet. The navigation data provides attitude during the scan so real word location can be determined. The ID that is part of the NavSet Scan Line Data header indicates the number of scan lines back that this data is applicable to. For example, a 1 would be the previous packet while a 0 would indicate this packet. TimeStamp is the large time stamp captured at the start of the scan. Normal operating procedure would have the system time synchronized to the Inertial Navigation Unit and accuracy should be better than 2 ms. nPtsPerScanLine is the number of points in this scan line and it can be used to determine how large the rest of the packet will be, although the Header Packet also provides a size for the entire Scan Line Data Packet.

Scan Line Data Packet		
Header Item	Value	Bytes
Struct Scan Data for CCI Communication		
Alignment Code	0x3D3D	2 (1 UINT16)
nScanLineNumber	1~N	2 (1 UINT16)
nPtsPerScanLine	M	2 (1 UINT16)
nChecksum		2 (1 UINT16)
sNavSet		283 (sizeof(StructNavLineSet))
sTimeStamp		14 (sizeof(StructTimeStamp))
PulseData		50 (structure) * M

5.3 Status Packet

Status Packet		
Header Item	Value	Bytes
Alignment Code	0x1000	2 (1 UINT16)
Status Type		2 (1 UINT16)
Packet Sent Count		2 (1 UINT16)

A status packet will be sent when asked for.

An Error will result in reset value of 0xFFFF, and will be sent to notify the CCI system that there is an error and the interface needs to be reset in order to recover. After reset, the next packet will be a Header Packet unless another command is sent by CCI.

If a Status packet is requested then Reset will have a value of 0x0000 and the packet sent count will be sent to the CCI system. This message is intended to be used as a handshake or as a keep-alive type of message verifying

round trip communication is occurring. If packet sent count does not increment when the request for Status is sent, that is an indication of a problem and reset should be commanded by CCI.

Status Type		
Status Type	Value	Bytes
Error	0x1000	2 (1 UINT16)
Status	0x0001	2 (1 UINT16)

5.4 sNavSet (StructNavLineSet)

sNavSet has a nNavID for matchup with the proper scan line that the data applies to and 3 packets of Navigation data for each scanline. Navigation Data is recorded at the beginning of scan, middle of scan, and end of scan. Since a period of scanning takes time dependent upon the number of scan pulses per each scan line, the movement of the scanner while taking a scan line is important to reconstruct the image. For example, if nNavID is 0, then the data applies to the same packet as it was delivered in. If the value is 1 then the data applies to the previous scan line, 5 would be five scan lines back.

sNavSet (StructNavLineSet)		
Header Item	Value	Bytes
nNavID		1 (1 UINT8)
sRealTimeNavStart		94 (sizeof(StructRealTimeNav))
sRealTimeNavMid		94 (sizeof(StructRealTimeNav))
sRealTimeNavEnd		94 (sizeof(StructRealTimeNav))

5.5 StructRealTimeNav

StructRealTimeNav structure is used for sRealTimeNavStart, sRealTimeNavMid, and sRealTimeNavEnd. The system can be configured where the INS outputs navigation packets and this data is saved so that everything is together to reconstruct the image. The data is the output of the INS at the time and is the direct forward estimation of the system attitude. The SL sensor reads this data and linearly estimates the system variables to exactly hit the start time, middle time and end time. It is possible that all three sets could be identical if no new navigation data comes in before the scan line is finished. The data is embedded into each scanline, however due to network lag, the data may apply to a different scan line than the packet it is delivered in. CCI can choose to use this navigation data, or use its own navigational data.

StructRealTimeNav		
Item	Value	Bytes
fHeading_deg		4 (1 float)
fRoll_deg		4 (1 float)
fPitch_deg		4 (1 float)
fNorthSpeed_mps		4 (1 float)
fEastSpeed_mps		4 (1 float)
fVerticalSpeed_mps		4 (1 float)
dLat_deg		8 (1 double)
dLon_deg		8 (1 double)
fDepth_m		4 (1 float)
nNavTimeStamp	InsTimePack	4 (1 UINT32)
structTimeStamp		14 (sizeof(StructTimeStamp))
Unused(spare to grow)		32*1

5.6 StructTimeStamp

StructTimeStamp is used in StructRealTimeNav and StructScanCFG. It is stamped by the SL Sensor time. The tm_yday is counted from the 1st of the year. The tm_mon starts with 0 as January. This time is synchronized to the INS system during operation with a max delta time of 2 ms offset before the clock is syncing up again. The sync is a jump in time so it is possible that it can go either forward or backwards in time.

StructTimeStamp		
Item	Value	Bytes
tm_year		2 (1 UINT16)
tm_mon		1 (1 UINT8)
tm_mday		1 (1 UINT8)
tm_yday		2 (1 UINT16)
tm_hour		2 (1 UINT16)
tm_min		1 (1 UINT8)
tm_sec		1 (1 UINT8)
m_nanoseconds		4 (1 UINT32)

5.7 Pulse Data

On every scanline, there are scan points as defined previously. Each scan point has angle, offsets, time offset from the reference, Range and Intensity data. For each pulse, there is range and intensity data. There are 5 ranges and intensities per pulse, but not all may be useful data. Time of pulse is the pulse time offset added to the structTimeStamp in the header. This is the time used for navigation correction, usually linearly estimated between the start, mid and end navigation data, whichever set is just before and after the time calculated.

Pulse Data		
Item	Value	Bytes
AZ, Cross track angle(deg)		4 (1 float)
EL, Forward track angle(deg)		4 (1 float)
Az offset from center(m)		4 (1 float)
El offset from center(m)		4 (1 float)
Pulse time offset from timestamp(usec)		4 (1 UINT32)
LOS Range 1 of 5		4 (1 float)
…		
LOS Range 5 of 5		4 (1 float)
Intensity 1 to 5		2 (1 UINT16)
…		
Intensity 5 of 5		2 (1 UINT16)

5.8 Ack and Error responses

Commands that the user may send to the interface:

Command Packet		
Header Item	Value	Bytes
Alignment Code	0xF00D	2 (1 UINT16)
Command		2 (1 UINT16)
nScanLineNumber		2 (1 UINT16)

Commands are Error/Resend, Ack, status request, and reset request.

On an Error, if the client wants the Scan Line Data packet to be sent again, return an error packet with the scan line number that is desired. The request must be made before the next scan set flushes the data out but it should still be available for approximately a minute after the initial send. The data will come in immediately after the request is received however it is possible that the data would be out of sequence. Use the scan line number to identify what data is actually showing up.

There is an (optional) ack with ScanLine Number. If client provides ack packets, then software will report successful connection status. The acks can be sent on each scan line or at some decimated rate, either is acceptable.

Status request for the server to send a status packet with the packet sent count. Since this request generates a packet, the count should increment on each request. If the packet count does not change, then that is an indication of error and need for system reset.

Reset request causes the packet sent count to start over and the interface will be shut down and restarted. The next packet will be a Header packet unless CCI sends another command before scan data comes in.

nScanLineNumber value will be ignored for Reset and Status requests.

Command Type		
Command type	Value	Bytes
Error/Resend	0x1000	2 (1 UINT16)
Ack	0x0001	2 (1 UINT16)
Status report	0x0100	2 (1 UINT16)
Reset	0xFFFF	2 (1 UINT16)

5.9 CheckSum

Header Packets and Scanline Data packets have a CheckSum. When calculated, the checksum is set to zero for the calculation, and then set to the calculated value after. Thus, the receiver needs to set the checksum field as zero before calculating checksum value over the packet size. The routine for calculating CheckSum is below, where `m_pBuffer` is the byte packet buffer to transfer. Note that odd packets will add the last byte in as an UPPER byte with the lower byte padded with zeros. (ie: last byte << 8, rather than as a lower byte).

```
UINT16 CalCheckSum()
{
	UINT16 n16Checksum=0; UINT16 n16Check;
	for ( UINT32 i=0; i < m_n32PacketSize ; i+=2 )
	{
		n16Check = (UINT8)m_pBuffer[i];
		n16Check = n16Check<<8;
		if ( i+1 != m_n32PacketSize ) n16Check += (UINT8)m_pBuffer[i+1];
		n16Checksum += n16Check;
	}
	return n16Checksum;
}
```

6 General Packet Usage in C++

The data structures and packet definitions were designed for TCP/IP socket usage. The Laptop will be configured with a static IP address, and the CCI will connect to it via standard TCP/IP socket methods through a Gigabit Ethernet switch.

Socket Connections		
RTV IP Address	**Description**	**Port Number**
192.168.168.2XY	TCP/IP Scan Data packet	2019
192.168.168.2XY	TCP/IP Navigation packet (on replay)	2020

In general, socket communications should follow this protocol:

1. A socket connection is made
2. A packet is sent
3. The receiver first reads the header (fixed byte size portion)
    a. receiver verifies valid header items such as the alignment code is correct
    b. receiver calculates the size of the remaining packet in the case of Scan Line Data Packet and reads that.
    c. receiver processes data and sends (optional) acknowledgement
4. If invalid header, error on read, or error on send
    a. If receiver wants the data to be sent again, send resend request
    b. If the interface needs to be cleaned up or on several back to back errors, send reset request, close the socket and re-initialize the socket connection.
    c. Optionally the receiver could scan for next alignment code and continue without asking for the data again.

In some instances, a “read timeout” error is acceptable per the desired protocol.

To meet protocol time deadlines, it is recommended that each socket connection should be controlled via a separate thread of execution. The socket data size can be fairly large, (620 rows x 200 views max scan size) so at least 7,650,000 bytes of buffer should be allocated and is possible to fill very quickly on a “Fast Scan” where the data becomes available all at once. This amount of data can be delivered at ~10 Hz, but typical operation is much slower. Processing of the data in chucks is acceptable however, the data will be sent to the socket all at once for a fast scan. On normal scans, the header packet will be sent, then each scan line will be sent as the data comes in. The size of the packet will change depending on the scan but the header packet will define the size for the packets that follow. If CCI does not keep up with the data generation, the socket will overfill and the extra data will be dropped. This could result in a packet that is not complete and a resync would be needed to resync the communications.

7 Appendix

Definitions for standard structures used in the simulator software (subject to change).

```
//struct TimeStamp
typedef struct
{
       UINT16 tm_year;
       UINT8  tm_mon;
       UINT8  tm_mday;
       UINT16 tm_yday;             // days since Jan 1 - added for struct tm compatibility
       UINT16 tm_hour;
       UINT8  tm_min;
       UINT8  tm_sec;
       UINT32 m_nanoseconds;
} StructTimeStamp;


typedef union {
      UINT32 word;
      struct {
       UINT32 empty :5;
       UINT32 ms    :10;
       UINT32 s     :6;
       UINT32 min   :6;
       UINT32 hours :5;
      } bits;
} InsTimePack;


typedef struct
{
       Float               fHeading_deg;
       Float               fRoll_deg;
       Float               fPitch_deg;
       Float               fNorthSpeed_mps;
       Float               fEastSpeed_mps;
       Float               fVerticalSpeed_mps;
       double              dLat_deg;
       double              dLon_deg;
       float               fDepth_m;
       InsTimePack         nNavTimestamp;
       StructTimeStamp     structTimestamp;
       UINT8               unused[32];
} StructRealTimeNav;


typedef struct
{
   UINT8               nNavId;              // Offset of Nav in Number of scan lines
   StructRealTimeNav   sRealTimeNavStart;   // Start of scan nav information
   StructRealTimeNav   sRealTimeNavMid;     // Middle of scan nav information
   StructRealTimeNav   sRealTimeNavEnd;     // End of scan nav information
} StructNavLineSet
```

```
typedef struct
{
	UINT16			nAlignmentTag; // fixed value
	UINT16			nScanLineNumber;
	UINT16			nPtsPerScanLine;
	UINT16			nCheckSum;
	StructNavLineSet	sNav;
	StructTimeStamp		sTimeStamp;
	StructPulse4CCI		sPulseData[nPtsPerScanLine];
} StructScanLineData4CCI;

typedef struct
{
	UINT16			nAlignmentTag;  // fixed value
	UINT16			nPacketCnt;
	UINT16			nVersion;
	UINT16			nNumScanLines;
	UINT32			nDataSize;	// ScanLineDataPacket Size
	float			fOffset[3];	// Distance arms to the INS sensitive pt
	float			fRotation[9];	// Rotation to INS coordinate system
	UINT8			nReferenceType; // for coordinate conversion
	UINT16			nCheckSum;
} StructPacketHeader4CCI;


typedef struct
{
	float		AZ_deg;
	float		EL_deg;
	float		Az_offset_m;
	float		El_offset_m;
	UINT32		time_offset_usec;// Pulse offset from main time stamp, microseconds
	float		Range_m[5];	// range from window
	UINT16		Intensity[5];	// scaled so that 0xFFFF is max value (saturated)
} StructPulse4CCI;


typedef struct
{
	UINT16			nAlignmentTag; // fixed value
	UINT16			nMessageCode;
	UINT16			nData;
} StructStatusPacket;


typedef struct
{
	UINT16			nAlignmentTag; // fixed value
	UINT16			nCommandCode;
	UINT16			nData;
} StructCommandPacket;
```

For port 2020, the Navigation Long definition:

```
// Define time structures: NAVIGATION_LONG - 90 bytes
// All multi-byte fields are sent MSB first,
// singed integers are two's complement.
typedef struct
{
   UINT16 sync;                // 0x24AA
   UINT32 SystemStatus;        // User Status
   UINT64 AlgoStatus;          // INS algorithm status
   float Heading_rad;          // [    0: 2pi ) + is East
   float Roll_rad;             // [  -pi:  pi ) + is port side up
   float Pitch_rad;            // [-pi/2: pi/2) + is bow down
   float NorthSpeed_mps;       // meters per second
   float EastSpeed_mps;        // meters per second
   float VerticalSpeed_mps;    // meters per second, + is up
   INT32 Lattitude;            // [-180:180) cal = 180 degree / 2^31
   INT32 Longitude;            // [-180:180) cal = 180 degree / 2^31
   float Altitude_m;           // meters + is up
   InsTimePack Time;           // Time
   float HeadingStddev_rad;    // radians
   float RollStddev_rad;       // radians
   float PitchStddev_rad;      // radians
   float NorthSpeedStddev_mps; // meters per second
   float EastSpeedStddev_mps;  // meters per second
   float VerticalSpeedStddev_mps;// meters per second
   float LattitudeStddev;      // meters
   float LongitudeStddev;      // meters
   float AltitudeStddev_m;     // meters
} StructNavigationLong;
```