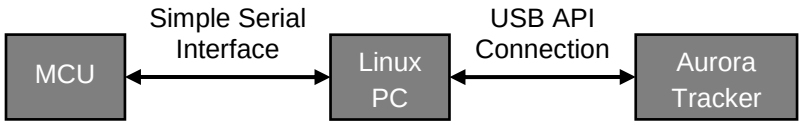
Simple Serial Interface to NDI Tracking Systems

Prepared By:

Michael Kokko
Dartmouth College
May 22, 2023

Objective

Northern Digital, Inc. (NDI) provides a "Common API" with C/C++ bindings for communicating with Polaris optical tracking devices and Aurora electromagnetic tracking devices. Although relatively well documented and easy to build on Linux (and presumably Windows) platforms, the API is not particularly well suited for use on small microcontroller systems (high default baud rate, relatively high overhead with several library files, written in C++ rather than C, etc.). To simplify the microcontroller-side implementation, we have developed a C/C++ interface (server) that communicates with the tracking system (specifically Aurora in the initial implementation) and streams transforms via a simple RS232 serial protocol suitable for consumption by a client program running on a microcontroller or other PC. A primary advantage of this approach is that the client (e.g. MCU) need not deal with tracker configuration, initialization, and communication and can thus simply consume a binary serial stream. Introduction of some latency and additional complexity in temporal synchronization are potential drawbacks that require further analysis.



Key Software and Dependencies

- **Tracker Serial Interface:**
  - Available from Mike's github
  - Includes **`mak_packet`** library for serial packet definition and construction
  - Installation and build (after dependencies below):
    - `git clone git@github.com:treelinemike/tracker-serial-interface`
    - `cd tracker-serial-interface`
    - `make`
- **NDI Common API:**
  - Available (with registration) from https://support.ndigital.com, but mirrored in private github repo
  - Installation
    - `git clone git@github.com:treelinemike/ndicapi`
    - `cd ndicapi`
    - `make sample`
    - Combine objects into static library and install it on system
      - `cd ./build/linux/obj/library/src`
      - `ar rc libndicapi.a *.o`
      - `ranlib libndicapi.a`
      - `sudo cp -p libndicapi.a /usr/local/lib/`
      - `sudo ldconfig -v`
    - Install header files
      - `sudo mkdir /usr/local/include/ndicapi`
      - `sudo cp -p ./library/include/* /usr/local/include/ndicapi`

- **Serial Communication Library:**
  - Cross-platform, available from https://github.com/wjwwood/serial
  - Installation
    - First install gstreamer with commands listed here
      - May need to remove gstreamer1.0-doc (necessary in Ubuntu 22.04)
    - `git clone git@github.com:wjwwood/serial`
    - `cd serial`
    - `sudo apt install catkin build-essential libgtest-dev`
    - `make`
    - `make test`
    - `make install`
    - For whatever reason it builds to `/tmp/usr/local/` so:
      - `sudo cp -pr /tmp/usr/local/include/serial /usr/local/include/`
      - `sudo cp -p /tmp/usr/local/lib/libserial.o /usr/local/lib`
    - Then, importantly, update the database that stores shared library locations:
      - `sudo ldconfig -v`

Serial Data Format

The following data packet definitions are used in the **`tracker-serial-interface`** server and client programs.

Physical Layer: RS232, no hardware flow control
Configuration: 115200 baud, 8-N-1, no software flow control

Note: Any byte subject to DLE stuffing will be followed by another identical byte if its value equals that of the DLE character (0x10). Stuffed DLE bytes are not to be included in byte counts or CRC8 calculation. Byte numbers shown in the tables below will change when DLE bytes are stuffed.

Single-Byte Command Data Packet (AS OF 22-MAY-2023 NOT YET IMPLEMENTED, PLAN TO IMPLEMENT NAK)

Byte*	Byte ID	DLE Stuff?	Value(s) / Type	Description
0	DLE	No	0x10	Data Link Escape
1	STX	No	0x02	Start Transmission
2	PKT_TYPE	Yes	0x06: ACK 0x11: TRK_START 0x12: TRK_STOP 0x15: NAK	Packet type
3	CRC8	Yes	uint8_t	CRC8 value based on byte 2, not including any stuffed DLEs
4	DLE	No	0x10	Data Link Escape
5	ETX	No	0x03	End Transmission

Transform Request Packet

Byte*	Byte ID	DLE Stuff?	Value(s) / Type	Description
0	DLE	No	0x10	Data Link Escape
1	STX	No	0x02	Start Transmission
2	PKT_TYPE	Yes	0x13: TFORM_REQUEST	Packet type
3	NUM_REQS	Yes	uint8_t	Number of transforms requested
4	TOOL_SN	Yes	uint32_t	Tool serial number (generally displayed in hexadecimal)
5				
6				
7				
…				Repeat bytes 4-7 for any additional transforms being transmitted
4+4*N	CRC8	Yes	uint8_t	CRC8 value based on bytes 2 through 3+4*N, not including any stuffed DLEs
5+4*N	DLE	No	0x10	Data Link Escape
6+4*N	ETX	No	0x03	End Transmission

Tracker Data Packet

Byte*	Byte ID	DLE Stuff?	Value(s) / Type	Description
0	DLE	No	0x10	Data Link Escape
1	STX	No	0x02	Start Transmission
2	PKT_TYPE	Yes	0x01: TRANSFORM_DATA	Packet Type
3	NUM_TFORMS	YES	uint8_t	Number of transforms included in this packet in bytes 8 through 7+36*N, not including any stuffed DLEs
4	FRAME_NUM	Yes	uint32_t	Frame number as reported by tracker, little endian (low byte first)
5				
6				
7				
8	TOOL_SN	Yes	uint32_t	Tool serial number (generally displayed in hexadecimal)
9				
10				
11				
12	Q0	Yes	float32	Quaternion, element 0
13				
14				
15				
16	Q1	Yes	float32	Quaternion, element 1
17				
18				
19				
20	Q2	Yes	float32	Quaternion, element 2
21				
22				
23				
24	Q3	Yes	float32	Quaternion, element 3
25				
26				
27				
28	Tx	Yes	float32	Translation, element 0 (x axis)
29				
30				
31				
32	Ty	Yes	float32	Translation, element 1 (y axis)
33				
34				
35				
36	Tz	Yes	float32	Translation, element 2 (z axis)
37				
38				
39				
40	ERROR	Yes	float32	Marker fit error, as reported by tracker
41				
42				
43				
…				Repeat bytes 8-43 for any additional transforms being transmitted
8+36*N	CRC8	Yes	uint8_t	CRC8 value based on bytes 2 through 7+36*N, not including any stuffed DLEs
9+36*N	DLE	No	0x10	Data Link Escape
10+36*N	ETX	No	0x03	End Transmission

*** NOTE: DLE stuffing will change byte numbers**

Serial Parsing Algorithm (implemented in `tracker-client.cpp`)

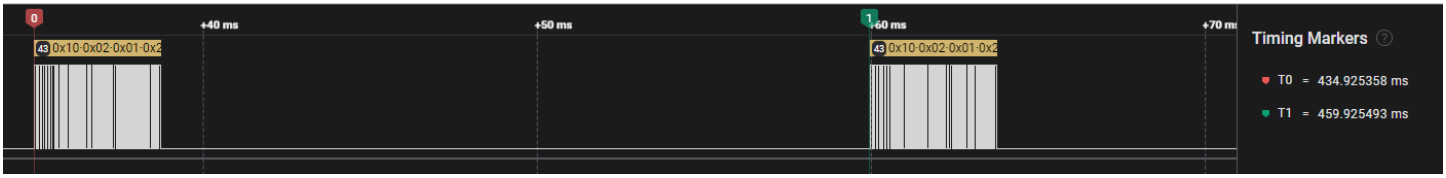
```
initialize byte buffer
initialize packet buffer
while( true )
    read all available bytes from serial port into end of byte buffer
    search from start of byte buffer for the STX byte preceded by an odd number of DLE bytes
    if( start sequence found )
        discard all bytes preceding DLE STX
        search from start of byte buffer for the ETX byte preceded by an odd number of DLE bytes
        if( end sequence found )
            copy bytes into packet buffer, eliminating unstuffing DLE bytes
            calculate CRC8 and check against received CRC byte, raise exception and/or discard packet if not a match
            extract packet type
            switch( packet type )
                case (command packet):
                    extract and execute command
                case (data packet):
                    extract, convert, and write/store/pass data
                default
                    invalid packet type
            end switch ( packet type )
        end if ( end sequence found )
    else if ( start sequence not found )
        reset byte buffer, keeping DLE if that is the last byte in current buffer
    end if( start sequence not found )
end while
```

Timing Analysis

(Performed at 115,200 baud with Saleae Logic USB Logic Analyzer)

It may be possible to run this faster than 115,200 bps, but this rate is certainly supported on Atmel MCUs and therefore likely on the Arduino platform. Currently the library is configured to send one packet per tool rather than one packet per frame. Thus, if in a single frame three tools are observed, three packets with the same frame number will be transmitted. Packet length will vary slightly depending upon number of stuffed DLE bytes, but a 43-byte packet will be transmitted in 3.77ms:

In streaming mode, packets arrive at 25ms intervals, consistent with Aurora 40Hz sampling frequency:



Thus, transmitting three packets (i.e. data for three aurora tools) should be possible at 115,200 bps while still allowing time for the NDI API calls (done at 921,600 bps) and processing both on the server and client sides. This timing could be relaxed significantly with parallel processing of serial data and/or a reduction in the sampling frequency.