

Lott/Jayaraman Animal Treadmill

Matlab/C/Serial SDK

Principle Engineer

Gus K Lott III, PhD

Neurobiological Instrumentation Engineer

Email: lottg@janelia.hhmi.org

Abstract

This document describes the included applications and the software interface, in C, to the treadmill system via the FTDI serial interface using the FTD2xx C driver instead of the virtual COM port driver. We found that the virtual COM port interface was not reliable for these high data rates. The SDK consists of a short description of the cross platform FTDI drivers, the command codes supported by the treadmill system, and a description of the data streams returned from the treadmill system during operation.

Note that this document also applies to using the device in virtual COM port mode as a serial device. All of the byte commands act identically whether you're interfacing via the virtual port mode or the C api. It all boils down to a byte stream.

The Matlab IMAQ code is tied to Windows implementations, but in general, the FTDI device interface is supported across all major platforms.

Revision	Date	Comment
0	3/25/10	
1	5/17/10	Added Example Code, descriptions of included code, Wrapup for release to public

SDK Documentation – Rev 1

Prepared by Gus K. Lott III, PhD

lottg@janelia.hhmi.org

May 17, 2010

HHMI Janelia Farm Research Campus
19700 Helix Dr.
Ashburn, VA 20147



Included Software

motionDisplay_imaq.m – Source Included

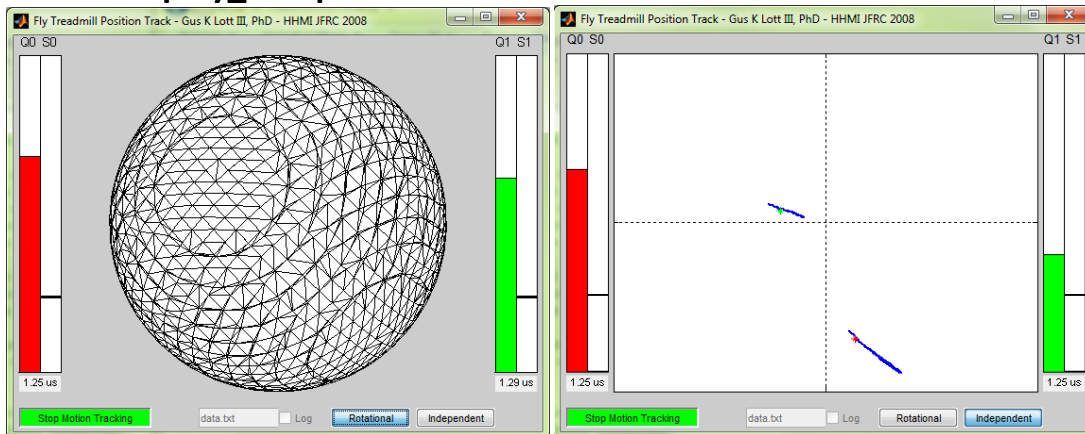


Figure: *motiondisplay_imaq.m* - Motion Display Software (Matlab)

- Displays the individual tracks of integrated x/y motion
- Status from the cameras (surface quality and shutter speed)
- Acquires data at the full 4 kHz data rate updating at 20Hz visually
- Use the motion vectors to translate a virtual ball under certain geometric assumptions
- This software will also log the raw data to disk (raw ASCII text file)

treadmillVideo_imaq.m – Source Included

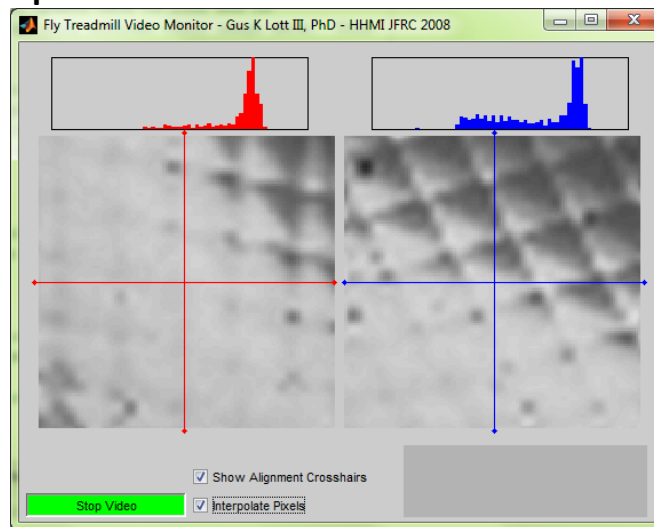


Figure: *treadmillVideo_imaq.m* – See Through the 30x30 pixel Cameras for focusing and lighting

- Subpixel interpolation for alignment and calibration to a target
- Histogram to assist in illumination setup
- Video update at 20Hz from each camera in parallel
- Matlab Based

Treadmill.dll – Matlab IMAQ Adaptor, Source Included

Included with the Matlab Software is a Matlab Image Acquisition Toolbox adaptor that acts as a data port between a multi-threaded application environment (polling from the FTDI driver) and Matlab. This is built according to the Matlab 2010a IMAQ Adaptor Kit located:

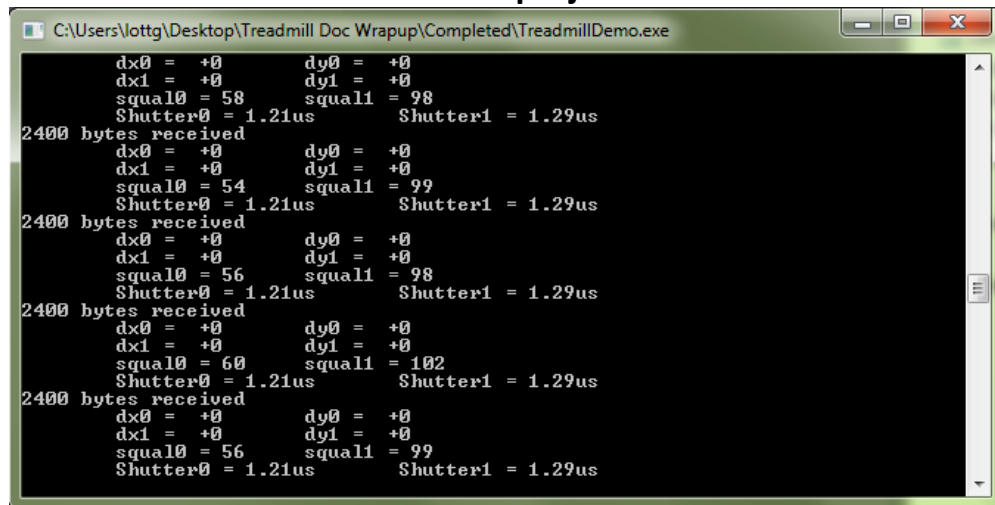
Matlab IMAQ Adaptor Kit:

http://www.mathworks.com/access/helpdesk/help/pdf_doc/imaq/adaptorkit.pdf

The adaptor functionality is illustrated in the above mentioned motion display and video viewing .m files. It basically polls and pre-formats the data and then hands it to Matlab. If you wish to modify this code to create a high performance closed loop solution, you will need to open the included visual studio project, point it to your location of the FTDI and Matlab libraries in the project setup, and link in your version of the matlab IMAQ adaptor kit project as described in the adaptor kit documentation.

This solution was created due to the lack of reliability in the data stream on the Virtual COM port in Windows (it seems to be treated as a low priority interrupt and thus loses data).

TreadmillDemo.exe – Visual Studio 2005 project source included

The image shows a screenshot of a Windows console application window titled "C:\Users\lottg\Desktop\Treadmill Doc Wrapup\Completed\TreadmillDemo.exe". The console displays a continuous stream of motion data in a structured format. Each data block consists of a header line "2400 bytes received" followed by two columns of data. The first column contains dx0, dx1, squal0, and Shutter0. The second column contains dy0, dy1, squal1, and Shutter1. The data values change between blocks, for example, squal0 goes from 58 to 54 to 56 to 60, and squal1 goes from 98 to 99 to 98 to 102. Shutter values are consistently 1.21us and 1.29us. The window has a standard Windows XP-style title bar and a scrollbar on the right side.

```
C:\Users\lottg\Desktop\Treadmill Doc Wrapup\Completed\TreadmillDemo.exe
dx0 = +0      dy0 = +0
dx1 = +0      dy1 = +0
squal0 = 58   squal1 = 98
Shutter0 = 1.21us  Shutter1 = 1.29us
2400 bytes received
dx0 = +0      dy0 = +0
dx1 = +0      dy1 = +0
squal0 = 54   squal1 = 99
Shutter0 = 1.21us  Shutter1 = 1.29us
2400 bytes received
dx0 = +0      dy0 = +0
dx1 = +0      dy1 = +0
squal0 = 56   squal1 = 98
Shutter0 = 1.21us  Shutter1 = 1.29us
2400 bytes received
dx0 = +0      dy0 = +0
dx1 = +0      dy1 = +0
squal0 = 60   squal1 = 102
Shutter0 = 1.21us  Shutter1 = 1.29us
2400 bytes received
dx0 = +0      dy0 = +0
dx1 = +0      dy1 = +0
squal0 = 56   squal1 = 99
Shutter0 = 1.21us  Shutter1 = 1.29us
```

Figure: *TreadmillDemo.exe* – Stand alone windows console application to test treadmill functionality without the requirement of a Matlab License (i.e. for integration into a VR system).

- Detects any number of connected FTDI devices
- Starts 4 kHz data stream from treadmill system
- Displays binned translation information in 50ms (20Hz) intervals
- Similar to motionDisplay.m, just text only

FTDI Driver interface

The system is communicated with via the FTDI D2xx Driver library. This C interface is supported across platforms including Windows, Linux, Mac OS X, and a variety of embedded platforms as well.

FTDI D2xx Driver Can be downloaded from:

- <http://www.ftdichip.com/Drivers/D2XX.htm>

This is a general purpose serial to USB bridge. There is not a unique ID associated with the device, so if you have several FTDI driven interfaces, you'll need to explore a connection (by sending query commands and reading responses such as the "dump register" command) until you find the treadmill.

The D2xx SDK Documentation is located:

- [http://www.ftdichip.com/Documents/ProgramGuides/D2XX_Programmer's_Guide\(FT_000071\).pdf](http://www.ftdichip.com/Documents/ProgramGuides/D2XX_Programmer's_Guide(FT_000071).pdf)

Functions of interest are basically limited to:

- FT_Open – Connect to a device
- FT_ResetDevice – Reset the FTDI chip and the serial interface
- FT_SetTimeouts – Define timeouts for blocking reads
- FT_SetDataCharacteristics – Configure Serial Data Stream
 - **FT_BITS_8**
 - **FT_STOP_BITS_1**
 - **FT_PARITY_NONE**
- FT_SetFlowControl – Configure no handshaking
 - **FT_FLOW_NONE**
- FT_SetBaudRate – Set data stream baud rate
 - **1250000 Baud**
- FT_GetQueueStatus – Determine number of bytes available for reading (optional)
- FT_Purge – Flush input and output buffers
- FT_Write – Write data to the treadmill
- FT_Read – Read data from the treadmill
- FT_Close – Close interface

The SDK programming guide for FTDI's serial interface is clearly written with example code. They also support an "event driven read" from the driver as well.

Byte Command Codes over Serial Interface

All control and data transfer to and from the treadmill system is achieved by raw multi-byte commands sent over the FTDI serial interface. The serial input will reset within 500ms of the last byte sent, so entire commands should be handed to the output buffer of the FTDI driver with a single call to FT_Write. Byte commands are summarized in the following table.

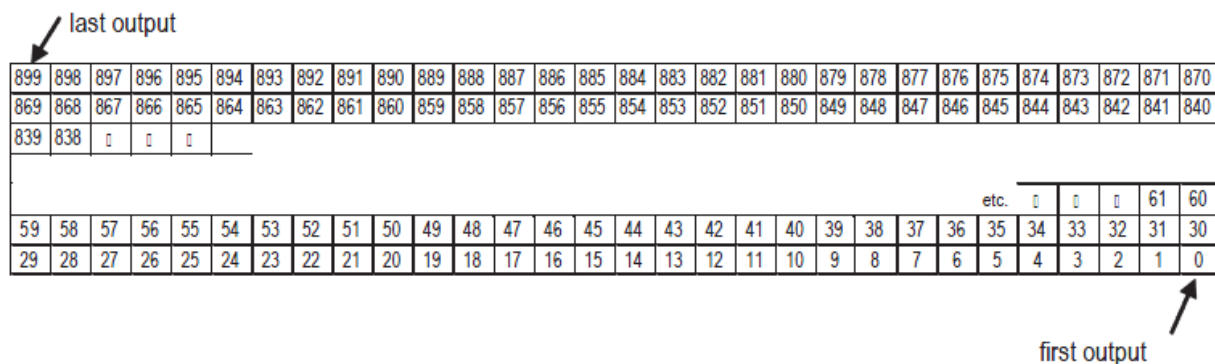
Serial Command Summary

[250 0]	Turn off Video Stream
[251 0]	Initiate video stream
[252 0]	Dump Internal Registers
[254 0]	Stop Motion Data Acquisition
[255 0]	Start Motion Data Acquisition

There are several other commands that allow the user to access the internal registers in the ADNS cameras and such, but they are not included in this document due to lack of thorough testing. A look at the firmware source code (included) will give a sense for what these commands are if desired.

Serial Command Detail

- **[251 0] – Initiate Video Stream from Cameras**
 - This command initiates a rapid dump of pixels from both cameras at 20Hz for visual display to a user for alignment and focusing purposes. The pixel streams are interleaved as received on the PC.
 - The stream begins with the bottom right pixel of each frame and continues across rows to the left and then up to the top row as illustrated in this figure from the data sheet:



Return data stream:

Data Stream: C0P0,C1P0,C0P1,C1P1,C0P2,C1P2,...,C0P899,C1P899

Frame Size per camera: 30x30 = 900 pixels

Data Stream chunk per 50ms video interval: 1800 pixels

The length of 1 frame is 900 pixels (30x30 image). So one complete frame from both cameras consists of 1800 bytes returned from the chip. No effort is made to indicate when the frame starts, it is assumed that no bytes are lost and that the PC software basically poll for 1800 byte chunks.

- **[250 0] – Stop Video Stream from Cameras**

- There is a bug in the firmware that requires the system to be physically reset after stopping the video stream in order to enter into motion tracking mode. A user should only have to enter video mode infrequently for setup and calibration purposes, so this is not a critical bug, but can be fixed in a future release.

- **[252 0] – Dump Internal Registers**

- This command may be used to poll a candidate FTDI interface to see if it is a treadmill and also returns status info from the internal registers in the ADNS camera chip
- This command should rapidly return 50 bytes read from registers within each camera chip (interleaved in the data stream).
- Byte order is:

1. Product ID	9. Resolution	17. Frame Period Max Bound L
2. Revision ID	10. Configuration Bits	18. Frame Period Max Bound U
3. Motion	11. Extended Config	19. Frame Period Min Bound L
4. Delta_X	12. Shutter Lower	20. Frame Period Min Bound U
5. Delta_Y	13. Shutter Upper	21. Shutter Max Bound L
6. SQUAL	14. Frame Period Lower	22. Shutter Max Bound U
7. Pixel Sum	15. Frame Period Upper	23. LP_CFG0
8. Maximum Pixel	16. Configuration II	24. LP_CFG1
		25. Observation

- **[255 0] – Start Motion Data Acquisition**

- This command initiates the 4kHz Motion data stream from the cameras. The data is returned in 12 byte packets per sample. Each sample consists of motion and status data from each of the cameras.
- A single sample packet consists of the following data:
 1. Zero (0)
 2. A counter that counts from 1 to 255 and then loops back to 1
 3. Delta_X from camera 0 – zero centered on 128
 4. Delta_Y from camera 0 – zero centered on 128
 5. Delta_X from camera 1 – zero centered on 128
 6. Delta_Y from camera 1 – zero centered on 128
 7. Surface Quality from Camera 0
 8. Surface Quality from Camera 1
 9. High Byte of Shutter Speed, Camera 0

- 10. Low Byte of Shutter Speed, Camera 0
- 11. High Byte of Shutter Speed, Camera 1
- 12. Low Byte of Shutter Speed, Camera 1
- Only the initial byte is ever allowed to be zero. This was designed to verify that the data stream never misses a byte and that packets remain aligned to the 12 byte width. It's essentially a byte header.
- Motion values are "movement since last measurement" and are in units of arbitrary counts derived from some level of pixel interpolation on the camera chip. Actual correspondence to physical displacement in millimeters must be calibrated as it is a function of optics.
- Turn motion values into signed displacements by subtracting 128 from the unsigned number.
- Subtract one from the quality numbers to get the reported value of "features detected" in the camera field of view. This number represents a confidence in the tracking result. This number is higher when the field of view has more structure. See ADNS6090 data sheet for more details. This is the readout of the SQUAL register.
- The shutter speed is a number of clock cycles that the shutter is held open for. The high and low bytes are prevented from ever being 1 (to preserve the packet header). The camera clock is 24MHz, so the actual shutter speed is nominally:
$$\text{ShutterSpeed} = ((\text{HighByte}-1)*256+\text{lowByte})/24\text{MHz}$$
- **The shutter speed must be faster than 250us** in order to prevent quantization error from appearing in the 4kHz data stream. If the shutter speed is too long, then motion will jitter across many samples. For example, a 500us shutter speed would return a zero and then the detected motion in the 4kHz data stream.
- The shutter speed is a function of the illumination and lens light gathering capacity and is automatically adjusted in the camera to lighting conditions. Typically, this is something that will be setup once and then ignored (similar to SQUAL).
- **[254 0] – Stop Motion Data Stream**
 - The system will complete the current motion packet and stop sending data.

Example C Code:

This example code connects to a treadmill device (at ID 0), starts motion data streaming in the default x/y mode at 4kHz, reads 200 motion vector pairs and associated status info from the system (see motion data stream definition above), and then stops the stream and closes the interface.

This code is mostly taken out of the acquisition thread of the included Matlab IMAQ adaptor code

```
//FTDI Variables
FT_STATUS ftStatus = FT_OK;
FT_HANDLE ftHandle;
unsigned char wBuffer[10];
unsigned char rBuffer[2400];
DWORD txBytes = 0;
DWORD rxBytes;

//Configure and Connect to Treadmill serial interface
ftStatus |= FT_Open(0,&ftHandle);
ftStatus |= FT_ResetDevice(ftHandle);
ftStatus |= FT_SetTimeouts(ftHandle,2000,2000);
ftStatus |=
FT_SetDataCharacteristics(ftHandle,FT_BITS_8,FT_STOP_BITS_1,FT_PARITY_NONE);
ftStatus |= FT_SetFlowControl(ftHandle,FT_FLOW_NONE,NULL,NULL);
ftStatus |= FT_SetBaudRate(ftHandle,1250000); //1.25MBaud Communication rate
if (ftStatus!=FT_OK) printf("Error")

//Stop any existing data stream from the treadmill
wBuffer[0]=254;
wBuffer[1]=0;
FT_Write(ftHandle,wBuffer,2,&txBytes);

FT_Purge(ftHandle,FT_PURGE_RX|FT_PURGE_TX);

//Motion Data @ High Speed
wBuffer[0]=255;
wBuffer[1]=0;
FT_Write(ftHandle,wBuffer,2,&txBytes);

//Read 200 packets of data (12 bytes per packet)
FT_Read(ftHandle,rBuffer,200*12,&rxBytes);

//Stop Acquisition
wBuffer[0]=254;
wBuffer[1]=0;
FT_Write(ftHandle,wBuffer,2,&txBytes);

//Close serial interface
FT_Close(ftHandle);
```