How to: **OPTOTRAK**

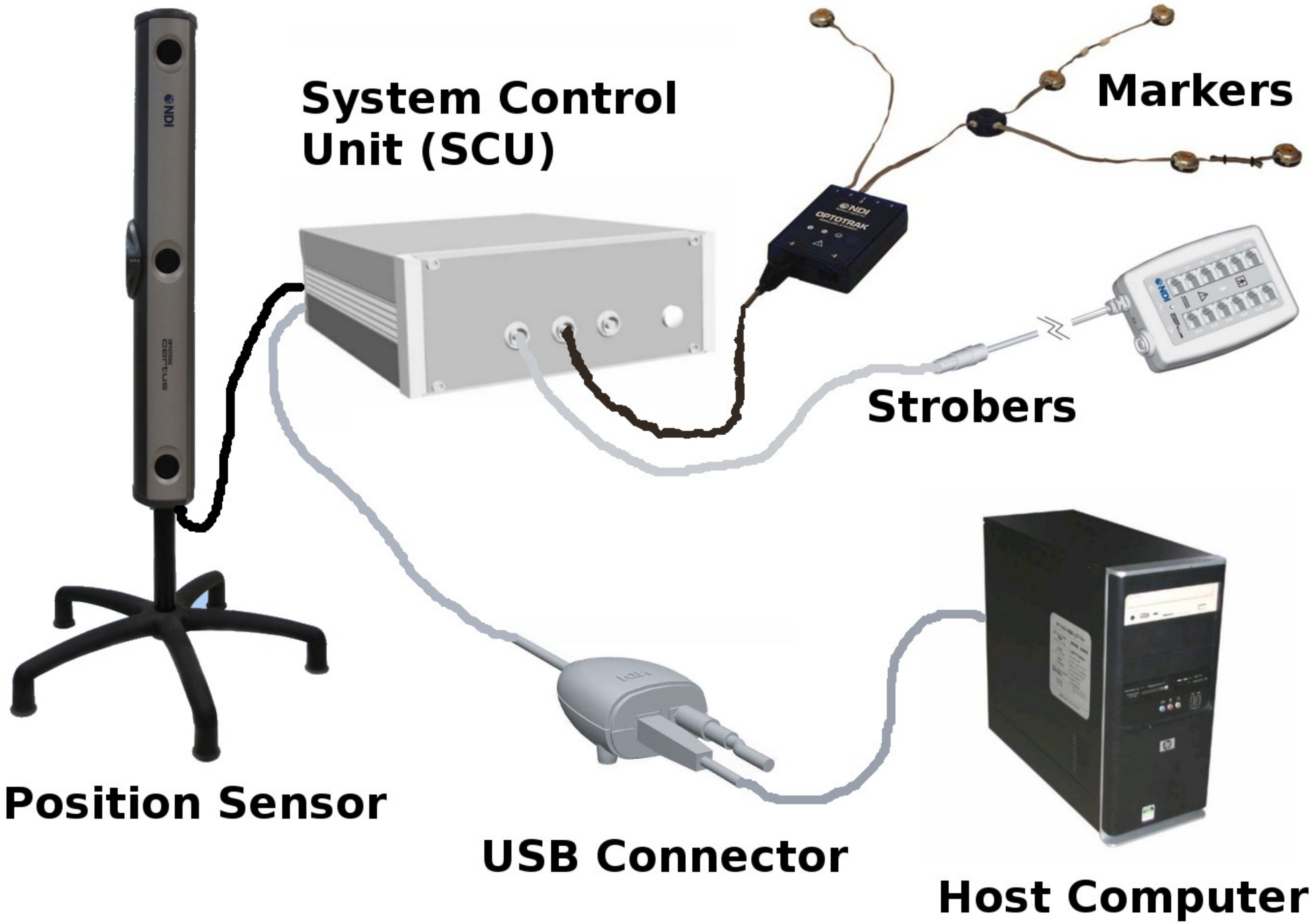
≠

Richard Schweitzer, September 9th 2016

How to: **OPTOTRAK**

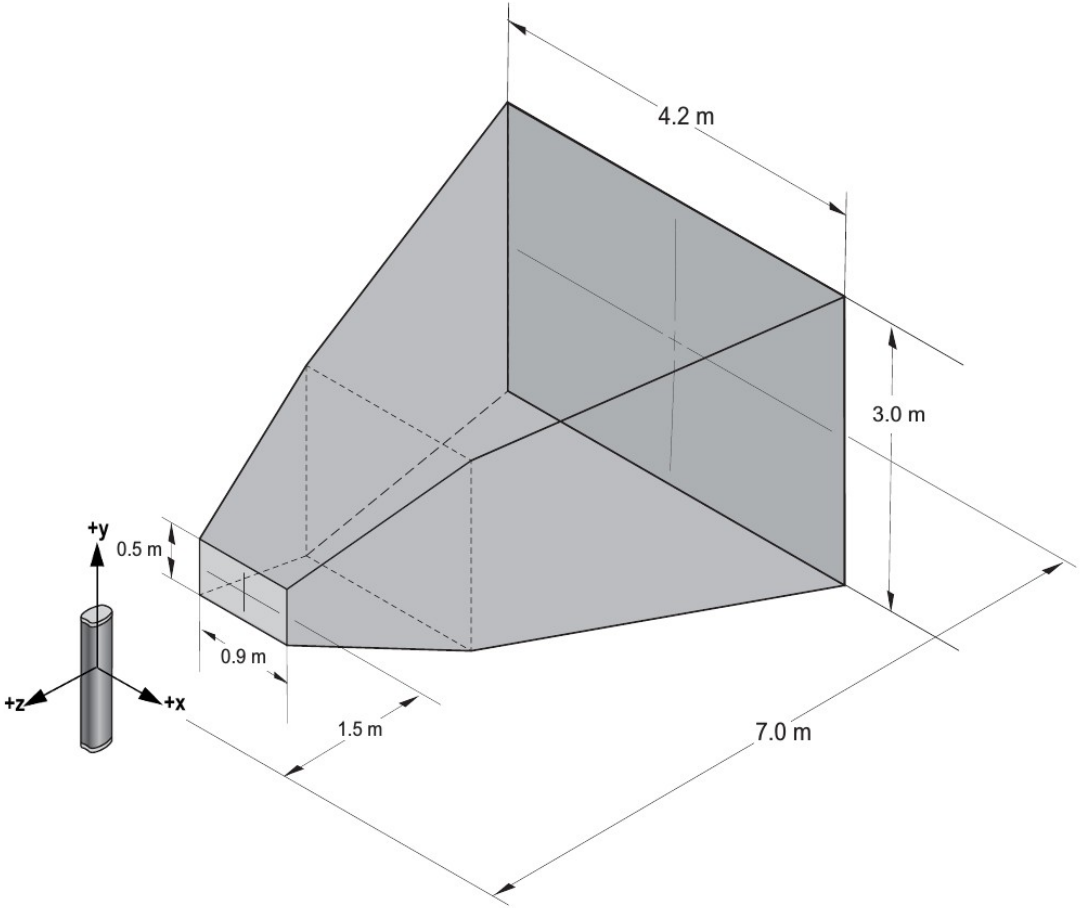
1) System Overview
2) Matlab_to_Optotrak()
   a) Blocking / non-blocking routines
   b) 3D data / Rigid Objects
   c) Sample program
3) Spatial Accuracy and Timing
4) Experimental Setup
   a) External synchronization via TTL using Datapixx
   b) Request samples from Optotrak Host via TCP
5) Air-drawing Demo (“roundtrip.m”)

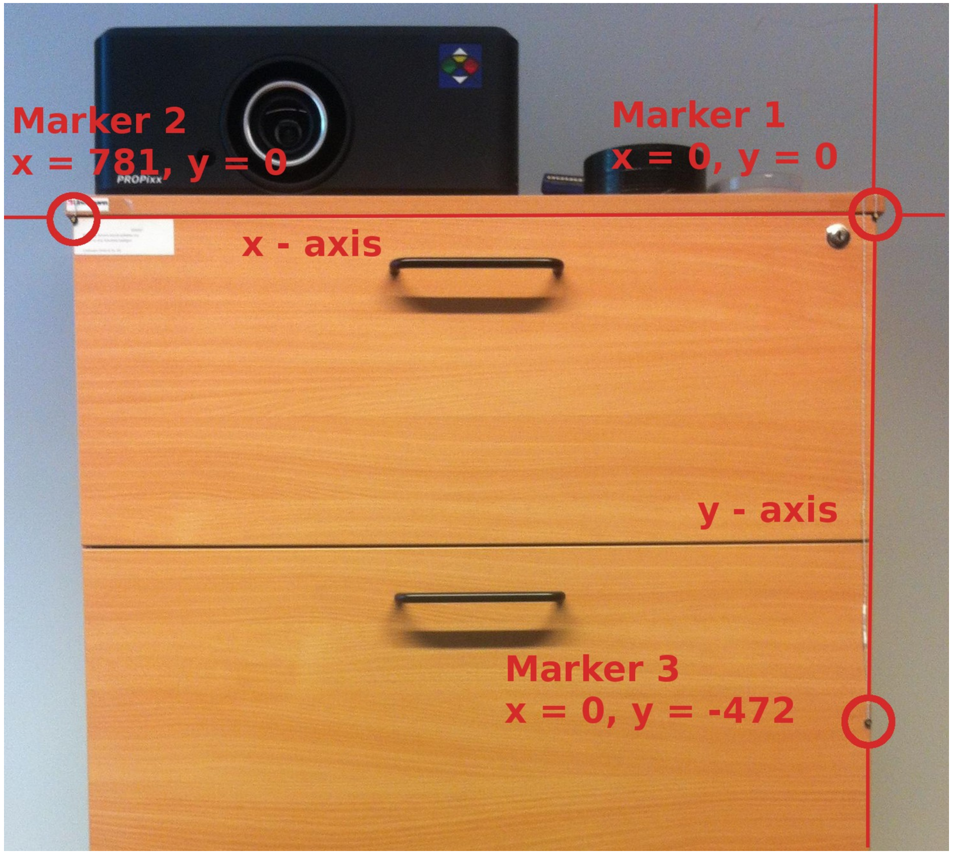
1) System Overview



1) Coordinate System

- custom alignment of coordinate system possible
- stored in .cam files (default: system.cam)





2) Matlab_to_Optotrak()

- MEX link function to send commands to the Optotrak, via the Optotrak API installed on the Optotrak Host, named *kastori*
- Based on Jarrod Blinch (08/2015), extended by R.S. (07/2016)
- edit *mexFunction.cpp* to make changes to it
- Call the function from Matlab to do things:

```
[return_code, return_data, tstart, tend] =
Matlab_to_Optotrak(FunctionName, Parameter1,
                   Parameter2, []);
```

- Example:

```
[error_code, transform_data, ts, te] =
Matlab_to_Optotrak('ReceiveLatestTransforms',
                   1, 3);
```

2) Collection with Matlab_to_Optotrak()

Check out: **`optotrak_sampling_server.m`**

- Specify all important parameters (e.g., sampling frequency, #Markers, results file name, camera alignment etc.)
- Run this function on Optotrak Host to collect data during an experiment on the Experiment Host
- Request latest samples at almost real-time (~1ms) via TCP
- To come: Look at resulting data while recording

2a) Blocking / non-blocking routines

- **Blocking**: The execution of the script is suspended until data is ready and retrieved.

```
Matlab_to_Optotrak('DataGetLatest3D')
```

- **Non-blocking**: Data is first requested and only retrieved when it is ready.

```
Matlab_to_Optotrak('RequestNext3D')
% here we do something else
… … …
Matlab_to_Optotrak('ReceiveLatest3D')
```

2b) 3D data and Rigid Objects

3D Data

- Coordinates of three markers
  (x1, y1, z1, x2, y2, ...)
- Depend on definition of coordinate system

Rigid Object

- To be defined prior to experiment
- Computed based on 3D data and definition file
- *Translation* of object origin (Tx, Ty, Tz)
- *Rotation* of object (q0, qx, qy, qz)

2c) Sample Program - Setup

```
% important parameters
collection_num_markers_1 = 3;            % markers on optotrak port 1
collection_num_markers_2 = 6;            % markers on optotrak port 2
collection_frequency = 300;              % Hz
collection_duration = 2;                 % s
cam_filename = 'Aligned20160728';        % .cam file, coordinate system
save_here = [];                          % where we save data

% setup the collection. 'err' is non-zero if an error occurred.
[err, err_code] = Matlab_to_Optotrak('TransputerLoadSystem', ...
    collection_num_markers_1, ... collection_num_markers_2, ...
    collection_frequency, collection_duration, cam_filename, 0);
if (err ~= 0)
    disp(err_code);
    error('TransputerLoadSystem did not succeed. Aborting!');
end

% load a rigid object, i.e. the shelf (id=0, markers=[1,2,3]).
% It's defined in 'schrank_4.rig'.
Matlab_to_Optotrak('RigidBodyAddFromFile', 0, 1, 'schrank_4');

% activate Optotrak Markers
Matlab_to_Optotrak('OptotrakActivateMarkers');
```

2c) Sample Program - Collection

```
% request a first sample
Matlab_to_Optotrak('RequestNextTransforms');

% start a sample loop.
tstart = tic;
while toc(tstart) < collection_duration
    % check for new sample
    [data_status, data, ts, te] =
        Matlab_to_Optotrak('DataReceiveLatestTransforms', ...
        1, collection_num_markers_1+collection_num_markers_2);
    % check whether sample has been received
    if strcmp(data, 'DataNotReady') ~= 1 && data_status == 0
        % save data in a non-preallocated file
        save_here(i,:) = data;
        % request new sample
        Matlab_to_Optotrak('RequestNextTransforms');
    end
end
```

2c) Sample Program - Shutdown

```
% wait for a second and then retrieve a sample, if there is one
pause(1);
[data_status, data, ts, te] =
    Matlab_to_Optotrak('DataReceiveLatestTransforms', ...
    1, collection_num_markers_1+collection_num_markers_2);

% Deactivate Optotrak Markers
[a, b] = Matlab_to_Optotrak('OptotrakDeActivateMarkers');
if (a ~= 0)
    display('Problem with: OptotrakDeActivateMarkers!');
    b
end

% Shutdown Optotrak
[a, b] = Matlab_to_Optotrak('TransputerShutdownSystem');
if (a ~= 0)
    display('Problem with: TransputerShutdownSystem!');
    b
end
```

2d) What the data looks like

3D Data

Frame	#Markers	uFlags	Marker_1.x	Marker_1.y	Marker_1.z	Marker_2.x	...	Timestamp
606	9	0	0.7698	-0.3516	-0.2225	-780.2203	...	8.1570709
607	9	0	0.7654	-0.3552	-0.2075	-780.2210	...	8.1570714

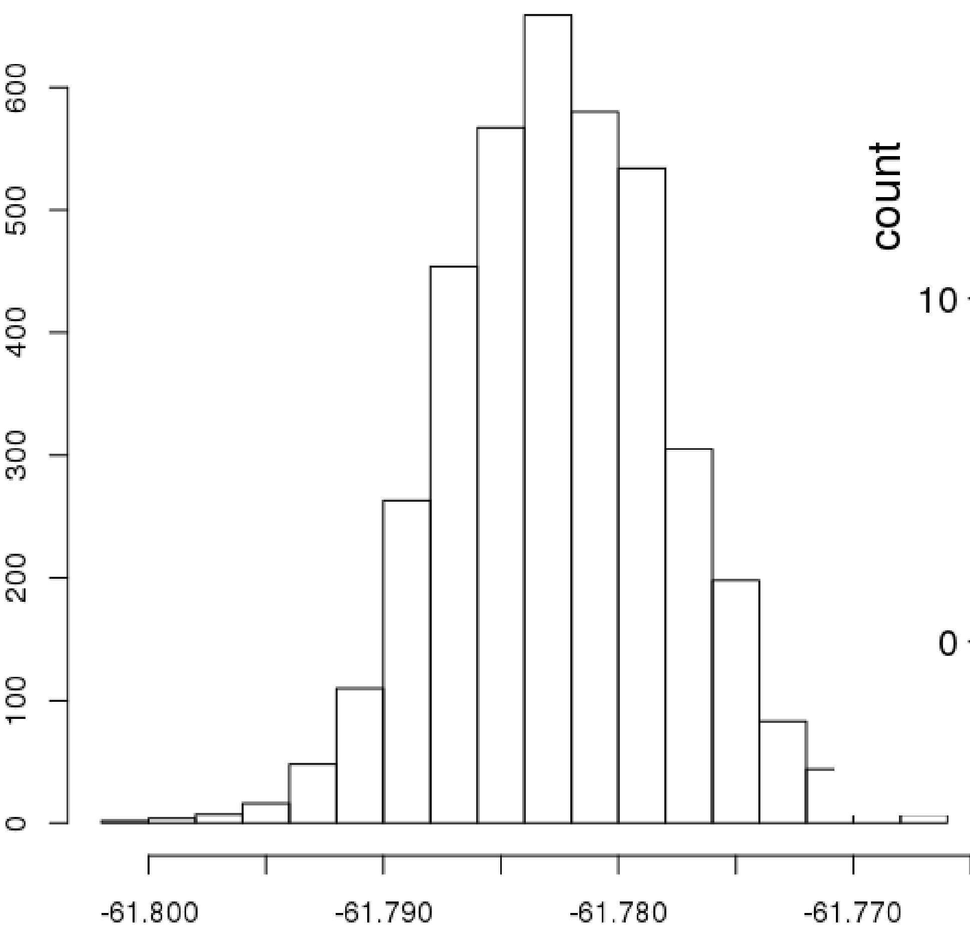
Transform (Rigid Body) Data

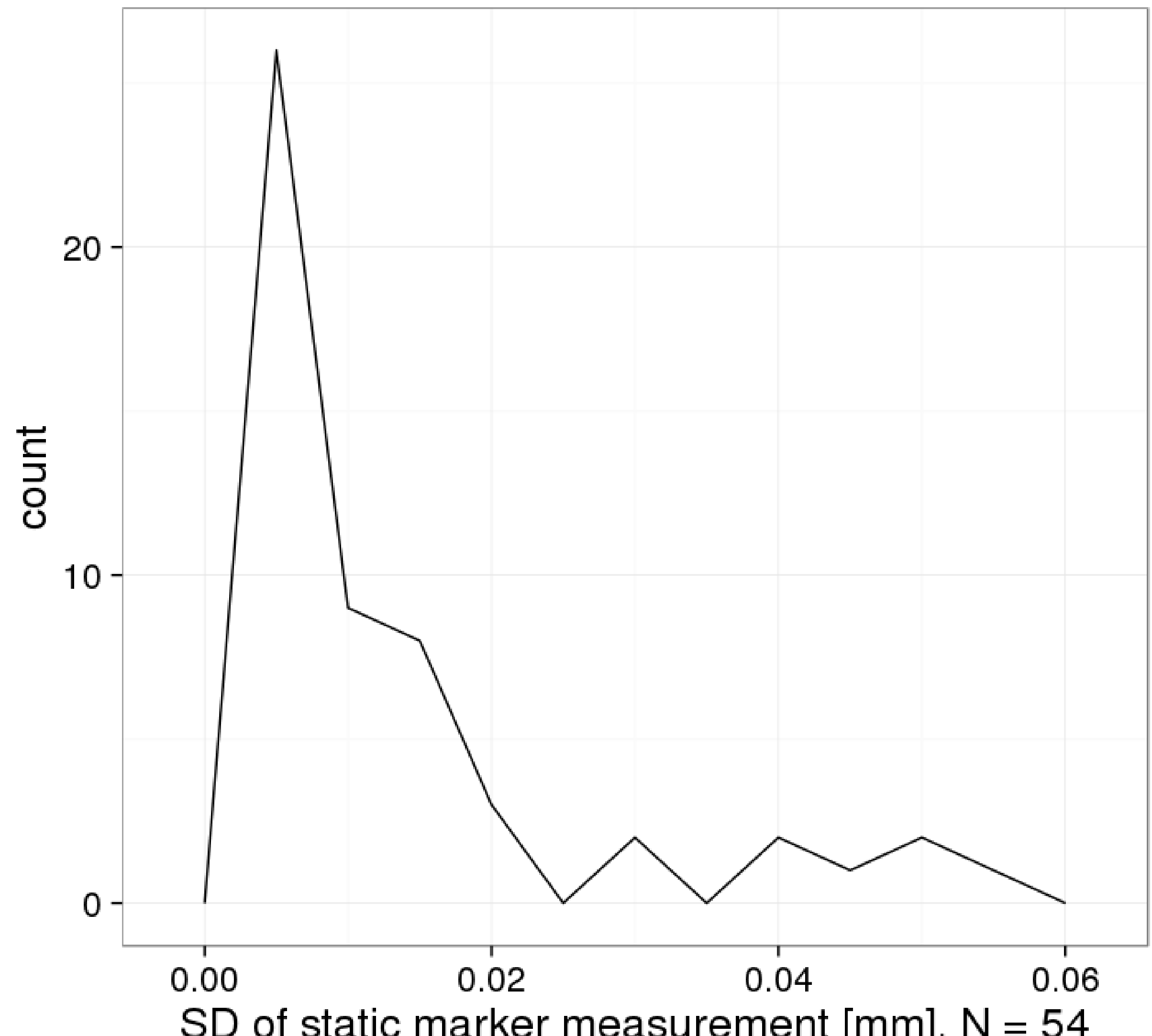
Frame	#Rigids	uFlags	RigidID.1	Flags.1	Quat.1.err	Quat.1.trans.x	Quat.1.trans.y	Quat.1.trans.z	...
406	3	0	0	4192	0.0314	-273.7896	-141.9251	16.9464	...

Quat.1.rot.q0	Quat.1.rot.qx	Quat.1.rot.qy	Quat.1.rot.qz	...	marker_start	marker_end	Marker_1.x	...	Timestamp
-0.7457	0.0006	-0.1850	0.6401	...	1	6	-279.8243	...	8.1660986

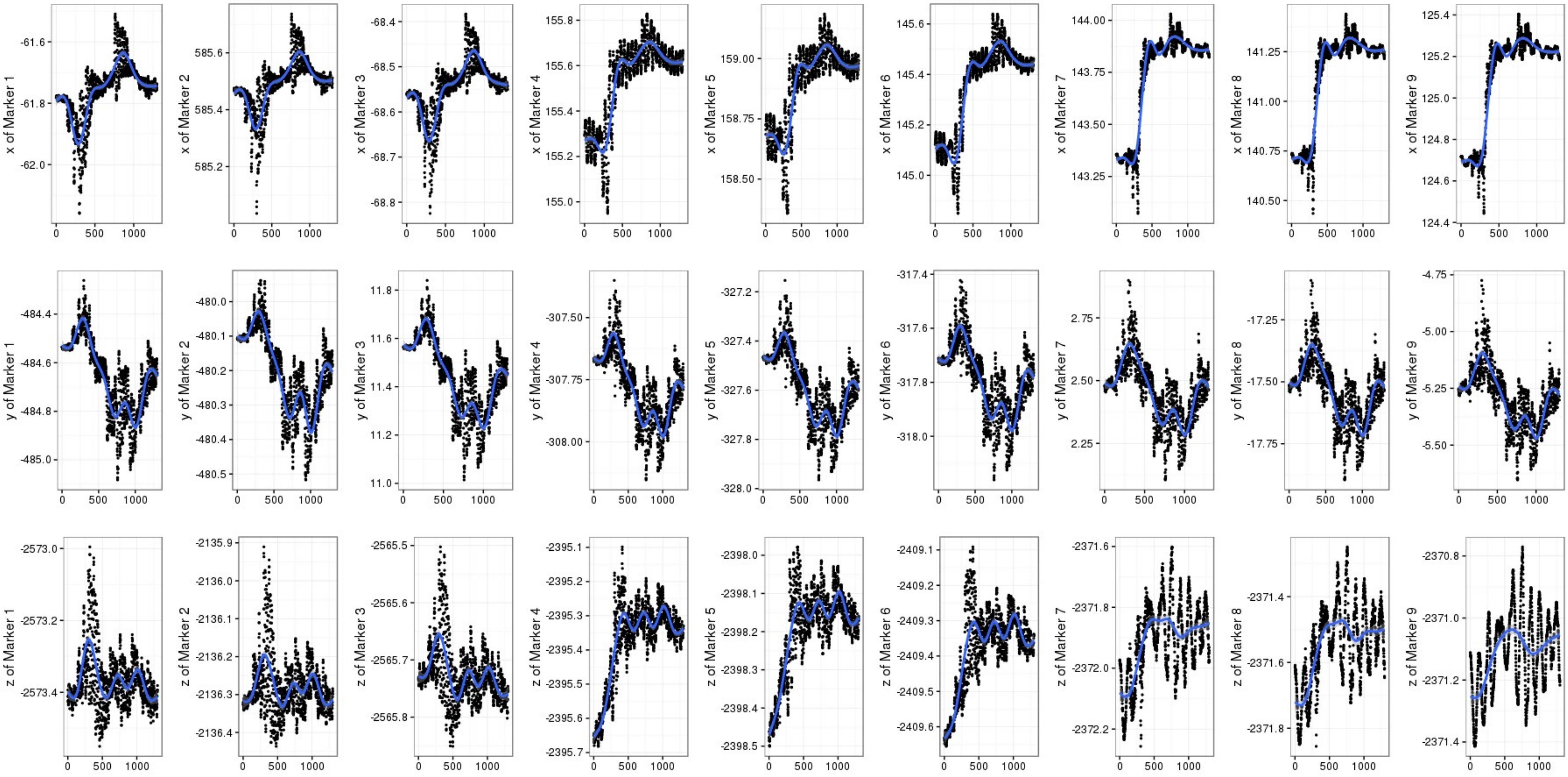
3a) Spatial Accuracy

- Mean standard deviation of: **0.01 mm**





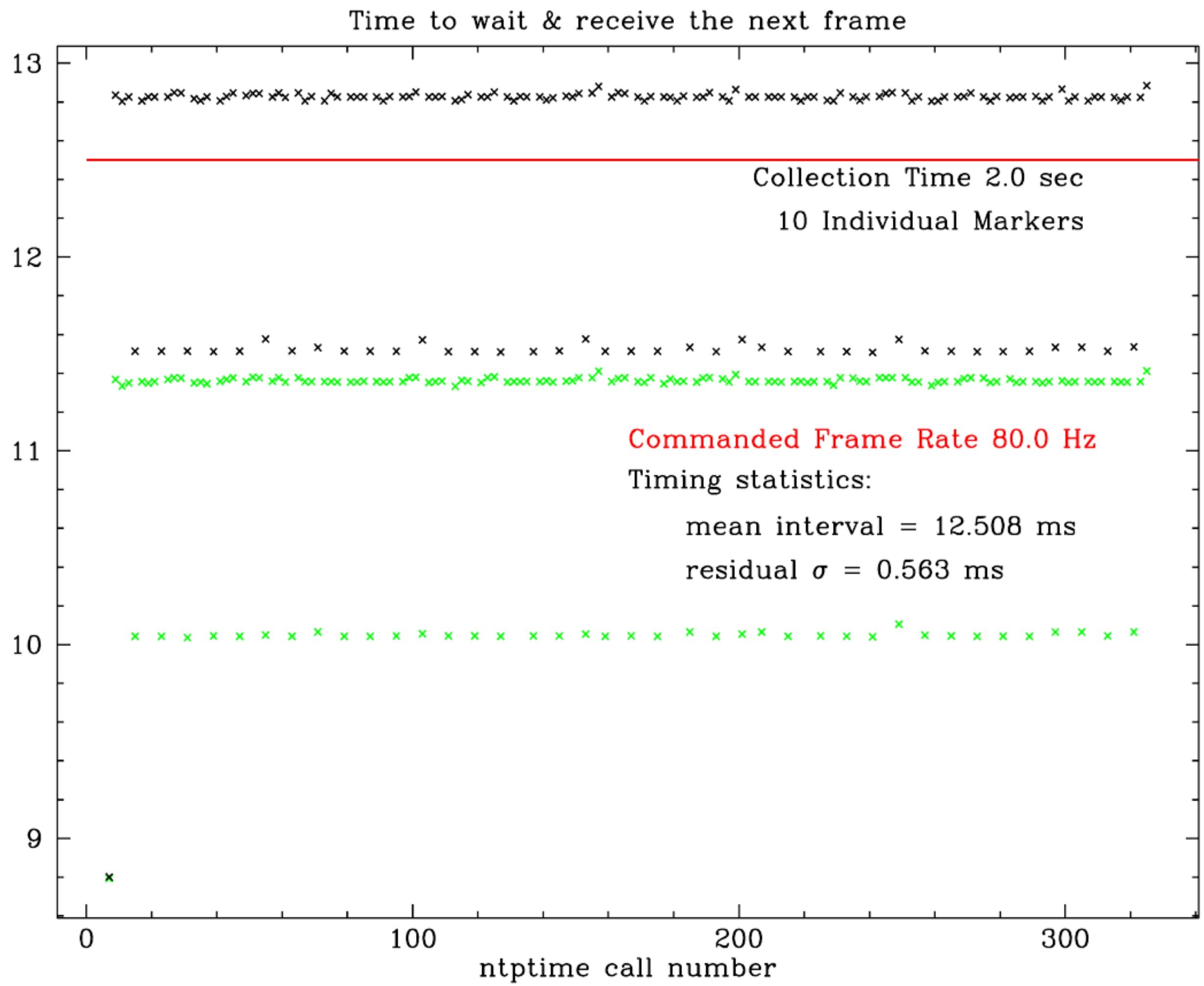
3a) Earthshaking Effects (for Martin)



3b) Temporal Accuracy

“We find that the Certus’ positional accuracy is very high, around ***20 μm*** *at a distance of 2.8 m.”*

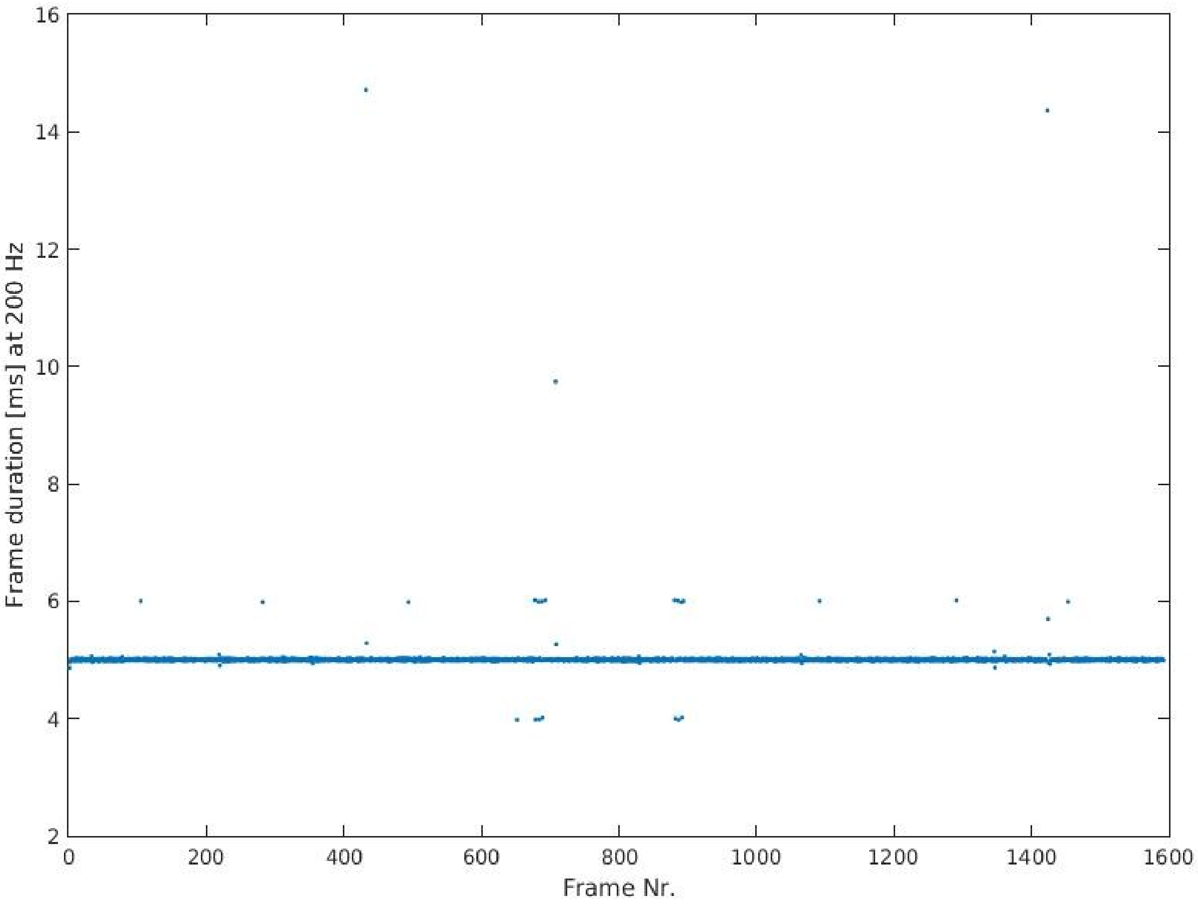
“In contrast, we find that its timing accuracy is typically no better than around ***5–10%*** [...]”



Barnes, P. J., Baldock, C., Meikle, S. R., & Fulton, R. R. (2008). Benchmarking of a motion sensing system for medical imaging and radiotherapy. *Physics in medicine and biology, 53(20)*, 5845.

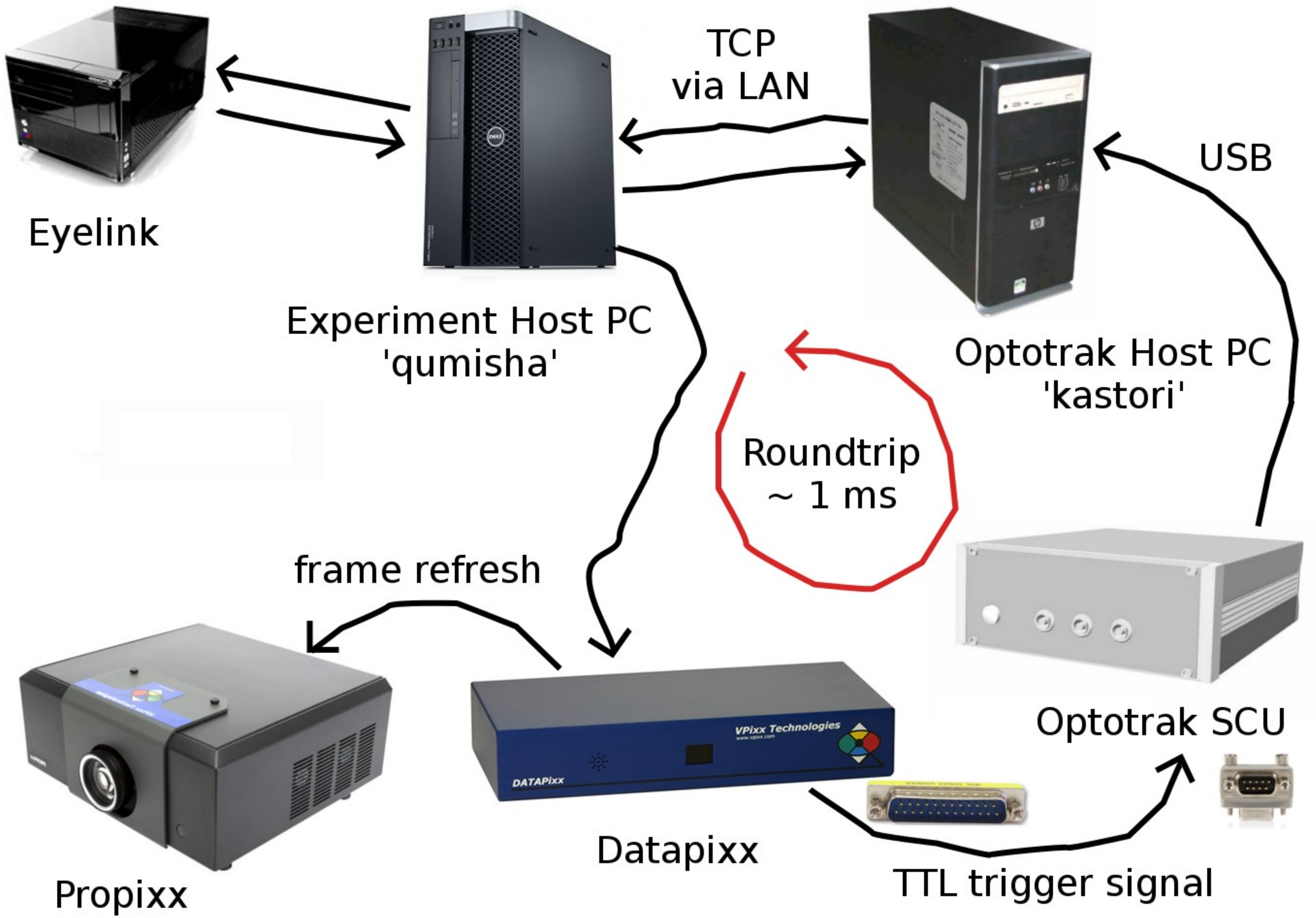
3b) Temporal Accuracy

Timing of 3D investigator slightly better,
still some extreme outliers persist.
RMSD = 0.3773 at 200HZ



Solutions:
– use low frame rates
– external triggering
– timestamps from host PC

4) Experimental Setup



4a) TTL Triggers using Datapixx

Background: Unreliable sampling timing

Problem: Synchronization between experiment and Optotrak sampling possible ?

Solution: Synchronization of a *screen refresh* and *frame production cycle* using Datapixx trigger system

Optotrak Trigger In:
– Pin 7 to start a pre-defined collection
– Pin 3 to trigger one *frame production*

4a) TTL Triggers using Datapixx

```
trigger_pin = 2^2; % serial pin 3 <- parallel pin 2 <- digital out 2
bufferAddress = 8e6; % address of the digital out buffer of datapixx
seconds = 25;          % how many seconds of triggering do we want?
refresh_rate = 120;    % the refresh rate of the screen in Hz

% create an outwave, that is composed of individual on/off signals.
doutWave = [trigger_pin, zeros(1,99)];
Datapixx('WriteDoutBuffer', doutWave, bufferAddress);

% specify the output sequence...
samplesPerFrame = size(doutWave, 2);                 % length of outwave
samplingRate = refresh_rate * samplesPerFrame;       % output frequency
framesPerTrial = refresh_rate * seconds;                % number of
Frames
samplesPerTrial = samplesPerFrame * framesPerTrial;  % total #signals

% now schedule the output sequence
Datapixx('SetDoutSchedule', 0, [samplingRate, 1], samplesPerTrial,...
    bufferAddress, samplesPerFrame);

% finally, sync this schedule with the next screen refresh
Datapixx('StartDoutSchedule');
PsychDataPixx('RequestPsyncedUpdate') % Mario Kleiner proposed this
Datapixx('RegWrVideoSync');
```

4b) Request Samples via TCP

Background: Experiment Host (Linux) and Optotrak Host (Windows) are different

Problem: Make data available to the Experiment Host without re-installing the Optotrak API

Solution: Transmit sampled data via a bilateral TCP on our local network

<u>Algorithm</u>:

1. Experiment Host writes a request code

2. Optotrak Host reads request code

3. Optotrak Host encodes latest sample in UTF and sends it back to Experiment Host

4. Experiment Host reads and decodes UTF to matrix

(must use fast mex-functions *splitstr.mexa64* and *str2doubleq.mexa64* !!!)

4b) Request Samples via TCP

```
request_code = 'request'; % write this in order to request a sample
stop_code = 'stopCollection'; % stop the Collection on the server

% this is the command to set up the connection to kastori. Insert the
% right parameters (ip, port_in, port_out) in accordance with kastori.
[in_socket, in_stream, d_in_stream, client_socket, out_socket, ...
out_stream, d_out_stream] = ...
    connect_to_optotrak_sampling_server(50, 2012, '172.29.7.127', 2011);

% abort the script if the connection failed
if isempty(in_socket) || isempty(client_socket) || isempty(out_socket)
    error('Error while connecting to optotrak server! Aborting now!');
end

% request optotrak sample from kastori
d_out_stream.writeUTF(request_code);

% wait for optotrak sample to return and decode it as soon as it arrives
data_available = 0;
while ~data_available
    [data_available, sample_data] = ...
    receive_from_optotrak_sampling_server(in_stream, d_in_stream);
end
```

5) Air-Drawing Demo (roundtrip.m)

qumisha - datapixx - (TTL) - optotrak - kastori - (TCP) - qumisha

Goal: A motion-contingent presentation on the Propixx.

Steps:

0. Start the Optotrak sampling server on Optotrak Host *kastori*
1. Connect *qumisha* and *kastori* via TCP
2. Start the prepared collection on *kastori* via a Datapixx TTL to the Optotrak SCU. (serial pin 7)
3. Setup a DoutSchedule on the Datapixx and sync it with the first frame refresh. (serial pin 3)
4. Start request-display-loop (i.e., painting)
    4.1. Request and get the sample from *kastori* via TCP
    4.2. Display the position of a marker in the screen using Psychtoolbox