Journal of Neuroscience Methods 308 (2018) 129–134

Contents lists available at ScienceDirect

Journal of Neuroscience Methods

journal homepage: www.elsevier.com/locate/jneumeth

MOTOM toolbox: MOtion Tracking via Optotrak and Matlab

Zoltan Derzsi*, Robert Volcic

Department of Psychology, New York University Abu Dhabi, United Arab Emirates

ARTICLE INFO

Keywords:
Northern digital
NDI
Optotrak Certus
Optotrak 3020
Matlab
Toolbox
Visuomotor
Virtual reality

ABSTRACT

We present a Matlab toolbox that allows the user to control and collect data using Northern Digital's Optotrak system. The Optotrak is a modular motion capture system, which tracks the positions of infrared markers. It also supports grouping markers together as a single body. The body's position, orientation as well as all the marker position data can be obtained simultaneously. The installation, set-up and alignment procedures are highly automated, and thus require minimal human interaction. We provide additional scripts, functions, documentation and examples to help experimenters integrate the Optotrak system into experiments using recent 64-bit computers and existing Matlab toolboxes.

1. Introduction

The Optotrak Certus is a highly accurate motion capture system manufactured by Northern Digital, that has been used in both industry and academia for almost two decades. It is a camera-based tracker, which captures the positions of proprietary markers consisting of infrared light emitting diodes. The three sensors in a camera are placed slightly apart, so an active marker will be seen by the sensors at slightly different angles. For each marker, these 2D sensor data are used to calculate the 3D position coordinates. While the 3D position data is useful to the user, the raw data containing the location of the brightest point in each sensor is available as well their peak sensor values. The position calculation can only be done for a single marker at a time, so the Optotrak system uses temporal multiplexing to allow many markers to be tracked. The markers are being flashed sequentially by the Strober, which also serves as an interface between the markers and the System Control Unit (SCU). The SCU is also connected to the cameras, and it sends data to the host computer for further processing. Every Optotrak system has at least one camera with three sensors, one SCU, at least one strober, and many markers. A typical set-up and its operation is shown in Fig. 1, but it can be scaled up to use up to the use of 10 sensors and up to 512 markers.

An additional interesting feature of the Optotrak system is the capability to assign markers that have constant distances between each other. When the markers are permanently attached to an object, it is possible to track the entire marker group as a single, 'rigid body'. In this case, depending on the nature of the marker assignment, the system returns the calculated centroid coordinates and orientation angles in addition to the marker coordinates. This can be used to enhance location accuracy of the object itself, or to track something else that is impossible to attach a marker to, but tied together with the rigid body.

To control and gather data from the Optotrak system, Northern Digital recommends using their own proprietary software, First Principles. This software, however, does not support automated hardware configuration and data collection: manual interaction is necessary to configure the system and to start/stop position recording. Northern Digital released an Application Program Interface (API) written in C language which allows third-party software to interact with the Optotrak system, but it requires advanced programming skills to make use of it. In the field of academic research, Matlab (The MathWorks, Inc.) is a popular universal tool that allows easy and convenient data analysis and visualization. Unfortunately, it is our experience that collecting data via an experiment written in Matlab is particularly difficult when it is required to interface with proprietary hardware. A possible way of getting the data into Matlab is to organize collection separately in an independent system, and write a data importer script that can be used with the analysis. This approach might be adequate in applications when no hardware configuration is required during the experiment, however, in many applications, for example when Matlab and Psychtoolbox (Kleiner et al., 2007) are mainly used, we perceive this as a great source of inflexibility, as hardware cannot be accessed programmatically within an experiment. The continuous interaction with the Optotrak system has been shown to be very effective in a number of past studies: for example, when a perturbation needs to be triggered when the hand is at a certain distance from the starting position (Hesse and Franz, 2009; Franz et al., 2009; Camponogara and

* Corresponding author.
E-mail address: ha5dzs@gmail.com (Z. Derzsi).

https://doi.org/10.1016/j.jneumeth.2018.07.007
Received 4 March 2018; Received in revised form 6 July 2018; Accepted 7 July 2018

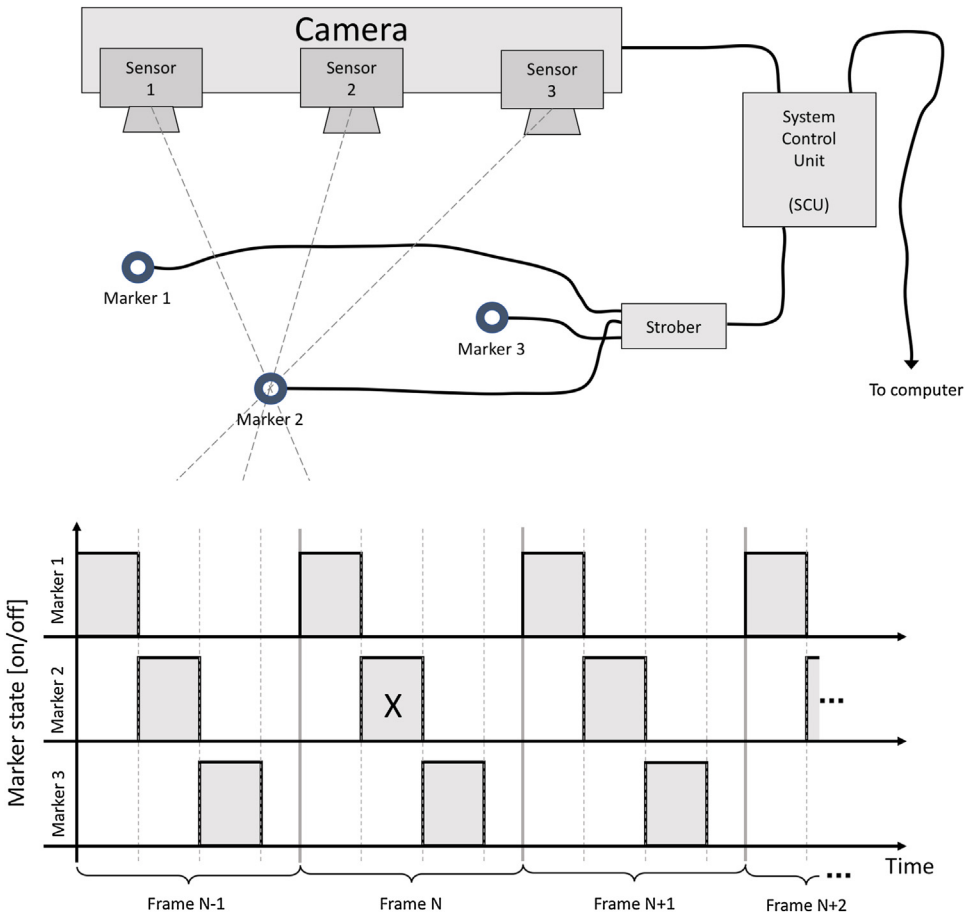

Fig. 1. The top drawing shows a typical Optotrak set-up: the camera with three sensors and the strober are connected to the SCU. Currently Marker 2 is activated, which is seen at different angles by the sensors. Bottom: the markers are temporally multiplexed, showing that Marker 2 is active right now. The marker strobing sequence finishes slightly before a new frame is being captured.

Volcic, 2017), or when positional information needs to be altered online (van der Kooij et al., 2013; Volcic et al., 2013; Volcic and Domini, 2016), or in the implementation of head position-contingent displays (Fantoni et al., 2010). This paper presents a Matlab toolbox that allows communication with the Optotrak system through Matlab's interface. With it, users can conveniently initialize and automate data collection without having to leave their Matlab environment, and all hardware handling is taken care of within its internal functions, transparent to the user. A requisite to use the toolbox is that the API has to be purchased from Northern Digital.

2. Why write another Optotrak toolbox?

We are aware of three Optotrak-Matlab interface implementations: One by Volker Franz (2004), whose approach was to integrate external C functions into Matlab through the use of Matlab's compiled binary executable files. A known limitation of this toolbox is that it was primarily written for the older 32-bit systems, and compiling the code requires being familiar with programming in C as well as being familiar with the API and a third-party Integrated Development Environment (IDE). We also found a collection of Matlab scripts available on the University of British Columbia's Research on Embedded Attention Lab website, which is written by Craig Chapman (2012), which uses Matlab's shared library feature to access the API functions directly. However, it is not possible to access all the data in the form the API prepares it due to a well documented limitation in Matlab's shared library facilities. These limitations prevent direct access to the Optotrak data. The third software we found, which also includes all functions in a single 'mex' file was written by Jarrod Blinch, who published his work in his blog (Blinch, 2011). Similarly to Volker Franz's toolbox, this also requires the knowledge of an external IDE. However, a tutorial is given that shows how to import and compile the code using a specific version of IDE. Furthermore, the last release of Matlab which worked on a 32-bit system was the R2015b, and subsequent releases are not capable of running existing 32-bit binary files. Since Northern Digital have released the 64-bit Optotrak drivers, and the most up-to-date version of Psychtoolbox requires 64-bit Matlab, we decided to write a new toolbox that takes the best ideas from the approaches described above, in the hope to be able to maximize the number of accessible features of the Optotrak system. We also took care to ensure reverse-compatibility: an experiment developed on a 32-bit system that uses our toolbox also runs on a 64-bit system, without having to modify the Matlab code. With the additional documentation and examples we include, setting up and automate data collection with the Optotrak system requires less programming skills and it is thus far more convenient than before.

3. Methods

Interfacing with the Optotrak system from Matlab is done via a recent (3.14 or newer) version of Northern Digital's Optotrak API, which can be obtained as separate library files for Linux and Windows, and for 32 and 64 bit systems. During the set-up process, the toolbox determines the user's operating system and computer architecture, and the appropriate library file is selected for use. The toolbox presented in this paper uses a hybrid approach: every documented API function is available through Matlab's shared library support, but not all of them provide meaningful results due to Matlab's documented limitations. For example, additional functions were written in C to convert data organized in nested structures to Matlab arrays. During the set-up process, the toolbox will compile all the necessary C code automatically, without having to manually edit the C code included. However, the automated compilation process can only be executed once certain requirements are

fully met; these requirements include having additional software installed, and the latest version of the library files copied to the appropriate location. The exact steps are included in the documentation of the toolbox. If there is a problem, the set-up script informs the user what steps are necessary to resolve it. The toolbox uses external configuration files, which are written in the Windows '.ini' syntax. All details of data acquisition and the rigid body definitions are specified in these configuration files, which can be tailored to individual experiments. Should incorrect or impossible settings be given in these files, the toolbox provides meaningful error messages and advises the user how to resolve them. The configuration files are not limited to the toolbox: they are compatible with the proprietary software supplied by Northern Digital.

3.1. Global coordinate system manipulation

By changing the global coordinate system, the same raw sensor data obtained by the Optotrak system will be converted to different 3D positions. The cameras in the Optotrak system use their own factory-set global coordinate systems, which is useful to have aligned to fit the needs of an experiment. We repeatedly align it to our experimental table at which our participants are sitting. To reduce the chance of marker occlusion, it is possible to add many cameras to track a single marker. In such cases, the coordinate systems of each camera has to be registered together to a single coordinate system first, and once the registration process is done, the newly generated common global coordinate system can be aligned as well. The coordinate system definitions are stored in camera parameter files, which are generated during each new registration and alignment. Northern digital use their own terms in their documentation, a brief glossary is included in Table 1. The registration and alignment processes work by tracking the positions of a known rigid body. In our implementation, we use the 'Cubic Reference Emitter' that is available from Northern Digital, but we have also included 3D-printable objects and their rigid body definitions that might be used for this purpose. We have created a script that detects the number of cameras used in the system, and guides the user through the registration and alignment process. The user is informed about the name of the newly generated camera file and the magnitude of the introduced tracking error in the process.

3.2. Controlling the Optotrak system with the toolbox

While every documented API function is directly available through the toolbox, we decided to make numerous scripts that are intended to make everyday tasks easier. These functions are separately documented, and have names that are easy to remember, some of them are shown in Appendix A. If only real time data is required, receiving data is now possible with a single line of code. Functions are also present to detect marker or coordinate proximity, and to handle data buffering, or decode and interpret status flags of the Optotrak system. We introduced simplified data formatting: the functions we wrote in C return the marker position coordinates and, if selected to do so, the rigid body transformation data in the form of matrices. Invisible markers and unsuccessful rigid body transformations are annotated with Matlab's a not-a-number (NaN) entry, instead of the Optotrak API's less practical default number (-3.697314×10^{28}). This way, 3D data can be visualized directly without the need for manual plot boundary adjustments. Furthermore, we added error management: it is now possible to retrieve the error messages directly from the SCU, and many functions have built-in sanity checks against user error. The toolbox can store raw data files, and it is possible to convert the raw data to 3D positions either immediately after file creation, or at a later date. It is also possible to execute the conversion within a different coordinate system, thereby porting data between set-ups, or to convert the raw data within a completely different coordinate system alignment, which may be useful when a re-alignment of the global coordinate system is necessary. The toolbox returns the 3D coordinates in millimeters, and the orientation Euler-angles (roll, pitch, yaw) are in radians. All numbers are in Matlab's double precision formats. There are functions provided to visualize marker positions. The marker position plotter function is also capable of displaying many frames as well, creating spaghetti-like 3D plots of marker data. This function also allows the update of the figure real-time, allowing the monitoring of marker coordinates directly. A simpler function is provided to show which markers are visible, which can be useful during marker set-up or when designing an experiment.

Table 1
The glossary of some key Optotrak terms.

Term	Description
Registration	The process of assigning many cameras into a common global coordinate system.
Alignment	The process of assigning the global coordinate system to a rigid body's local coordinate system.
Raw data	Sensor data directly available from the cameras.
3D data	The raw sensor data converted to *X–Y–Z* marker coordinates in the global coordinate system.
6D data	Rigid body data defined in two triplets: the *X–Y–Z* coordinates of the rigid body's location, and the Roll–Pitch–Yaw rotation angles.

3.3. Rigid bodies, warp detection

It is possible to create rigid body definition files programmatically with the toolbox, even during an experiment, provided that the assigned set of markers makes a rigid body transform possible. Following a successful transform, the system will return the body's centroid coordinates and the orientation angles, which the Optotrak documentation refers to as '6D Transform' and '6D data'. The toolbox also has a function to calculate the centroid coordinates and the orientation angles without a rigid body definition. Functions are also provided to detect warp, where the distances between the assigned set of markers and the centroid changed over time due to a loosely attached marker, for example. Just like the data acquisition settings, the generated rigid body definition can be stored in external configuration files, and it is used to determine the orientation angles. Should the user decide to proceed without the use of the rigid body tracking facility in the Optotrak system, the toolbox will calculate the orientation angles with respect to the centroid and a specified marker. In this latter case, it is the user's responsibility to detect warp, and that the marker from which the orientation is calculated is always visible to a camera.

3.4. Virtual markers

It is not always possible or practical to attach a marker to a desired surface. For example, if the position of the fingertips is required, it is impossible to attach markers to them as it would spoil the sense of touch and thereby compromise accuracy. One solution is to track a rigid body that is attached to the fingernail, and then calculate the location of the fingertip with respect to the rigid body. At the beginning of an experiment, we create a virtual marker definition using a function in the toolbox, and we use this definition to calculate the new positions of the virtual marker. While the toolbox functions are primarily intended to work with the Optotrak system's rigid body transforms, it is also possible to use virtual markers in applications where the centroid and the orientation are manually calculated, without the use of the Optotrak system's rigid body tracking facility.

4. Installation, documentation, source code availability and examples

The toolbox is located in our GitHub repository where direct download option is also available. We have included a detailed wiki-

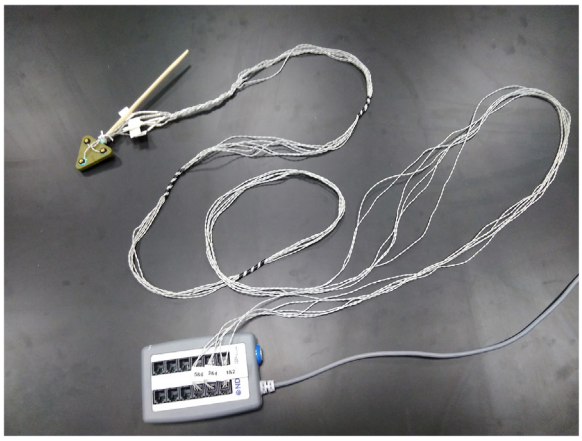

Fig. 2. The 6-marker VolcicLab Rigid Body, with all markers installed and connected to the Strober. In one of the examples included with the toolbox, we assigned a virtual marker to the tip of the chopstick attached to the marker holder.

style documentation. At the time of writing, the toolbox can be downloaded from: https://github.com/volcic/motom-toolbox.

In addition to the included documentation, there is a help section for every function in the toolbox. Furthermore, there is a brief introduction for newcomers to the Optotrak system, introduction to rigid bodies, virtual markers, and to the internal workings of the toolbox. For experts, there is added documentation for each built-in API function, including all the status flags and data management. Examples are provided on how to configure the system, how to handle real-time and buffered data, rigid bodies, and virtual markers. For rigid body and virtual marker tracking, we have also added a 3D-printable model which is shown in Fig. 2. Contributions from other users are welcome too. Some brief information about the prerequisites are listed along with set-up instructions in the README file as well: the user needs to set up Matlab so it is able to compile C code and copy the latest version of both the 32- and 64-bit Optotrak API files to the directories shown in the instructions. In order to make sure that the local header files are used, the user is also required to follow a simple tutorial that briefly edits the C header files. When all is done, the user is required to run the set-up script, which will detect the user's environment, and in case a required condition is not met, guides the user about what to do. We have included fully working examples and configuration files. There are two code snippets in the appendix: Appendix A.1 shows how to repeatedly fetch a single frame of real-time position data from the system, and Appendix A.2 shows an example implementation of the use of the data buffer, which allows position tracking at higher frame-rates. Our typical use of the Optotrak system and the toolbox is to align the coordinate system prior to the experiment, use the buffer to store data which is saved for each trial, and monitor a the real-time data while recording. At the end of the trial, we convert the raw files to 3D positions, which we analyze further. It is possible to integrate the toolbox with other environments. Fig. 3 shows the flow-chart of an example visuomotor experiment.

4.1. How to install the toolbox

It is important to note that the toolbox has two prerequisites. First, Matlab must have a supported C compiler installed that the toolbox can make use of, and the Optotrak API files that have been purchased from Northern Digital must be copied to their respective locations: the .dll and .lib files to 'bin' directory of the toolbox, and the .h files to the 'source' directory within the toolbox. Second, the .h files need to be edited following the instructions, which results in three lines of code added. The exact instructions are available as a step-by-step guide in the toolbox documentation. For legal reasons, we are unable to provide the modified .h files ourselves. Once downloaded from our repository, unzip the package to a directory and execute RUNME.m. Should a dependency be missing, the script will provide meaningful error messages and instructions.

4.2. Basic usage

There are working code and configuration files and their documentation included with the toolbox for a number of different mock experiments. The registration and alignment procedure is shown in detail separately. We suggest performing the registration and alignment process prior to an experimental session, to create a camera file which contains the accurately aligned global coordinate system. We recommend the creation of a new data acquisition configuration file for each new experiment. This may be done by modifying one of the examples we provided with the toolbox code. Depending on the desired

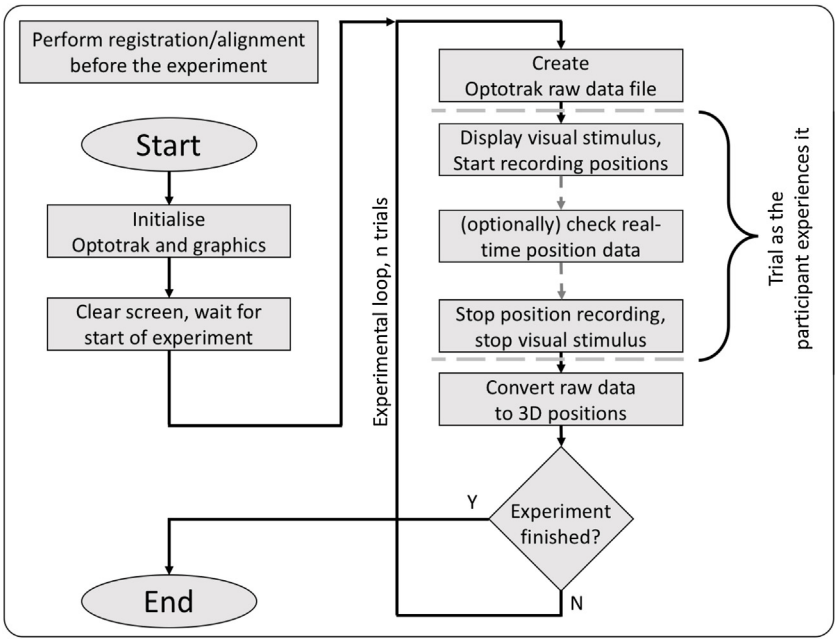

Fig. 3. The flow-chart of an example visuomotor experiment. The MOTOM toolbox can be integrated with other software, such as Psychtoolbox. While the real-time data functions take about 10 ms to execute during which time the a render loop will be held, recording position data to the Optotrak's buffer is done concurrently with the rendering. This makes it possible record marker positions at a higher sampling rate than the frame rate of the stimulus.

frame rate, we believe that it is best practice to access real-time data below the frame frequency of 80–100 Hz, and use the Optotrak system's buffering data acquisition facility for frame rates above 100 Hz. During the initialization of the Optotrak system, the toolbox will provide a warning or an error message when there is a risk of buffer overflow. The buffer usage depends on the length of the data collection time, number of markers, and the frame rate.

5. Limitations

5.1. Performance

We found that fetching a single frame of real-time position data in our set-up where we use two cameras and the SCU is connected to the host computer via Ethernet, took around 10 ms. Therefore, with real-time data, frame skips will occur above frame rates of approximately 100 Hz. This can be detected by examining the returned values of the serial number (or, frame-counter) of the captured frame. For applications that require high-speed tracking, we recommend using the data buffer, and keep real-time data for monitoring purposes only.

5.1.1. Error management and features

Please note that while core functionality such as marker and rigid body tracking has been tested thoroughly, other features such as the functions regarding the Optotrak Data Acquisition Unit (ODAU) have not been implemented. The rigid body orientation is only available in the Euler angle format. Minor features such as switch state detection, beeping, or the blinking of visible LEDs on devices have been implemented, but not rigorously tested. We have compiled a list of API functions in the online documentation to show in particular which functions do work in the toolbox, and which functions do not.

6. Conclusions

We have presented a Matlab toolbox that allows users to drive the Optotrak system without having to program in C or to leave the Matlab environment. At the time of writing, it is the only toolbox that allows the realization of a 64-bit Matlab-Psychtoolbox-Optotrak set-up. We wrote several data handling functions in C that handle data correctly. We also have added extra features, such as the creation of rigid bodies and the tracking of virtual markers. The toolbox also has data visualization functions. We decided to share the source code in the hope that other experimenters will benefit from an improved Matlab-Optotrak interface.

Acknowledgements

We would like to thank NYU Abu Dhabi for funding the project, Ivan Camponogara for the valuable information and testing during development, and Carlo Nicolini for his version of virtual marker tracking.

Appendix A

Link to download the toolbox: https://github.com/volcic/motom-toolbox. For more information and examples, please refer to the online documentation and the `examples` directory included with the toolbox. Further examples of data acquisition configuration files are included as well. If necessary for single-camera set-ups, the `'standard'` camera files may be used, which gives positions in the factory-supplied global coordinate system. However, we recommend aligning the coordinate system to suit the user's environment. Here, we have included two usage scenarios. Please note that in order to run this code, a data acquisition configuration file is required. The documentation of the toolbox includes information on how to create one. The included working examples contain these configuration files, which can be tailored to the experimenter's needs.

A.1 Obtaining 100 frames of real-time data

The following is a code snippet that shows a simple way of initializing the Optotrak system and then collect data.

```
% This example reads 100 frames of marker positions
% Frame rate, number of markers are in the config file

optotrak_startup; %this detects what hardware is available.
%The camera file contains the coordinate system alignment settings.
%The camera file may be set to 'standard' for single-camera set-ups.
%The 'config file' contains data acquisition settings.
optotrak_set_up_system('path_to_camera_file', 'path_to_config_file');

%Optionally, read back how the system was initialised.
data_acquisition_settings = optotrak_get_setup_info;
  %This is in a structure, flags are decoded as text.
%The markers are turned on by this point, so data can now be collected.
for frames = 1:100
  %Collect 100 frames in this loop.
  %Each row is a frame, columns are X-Y-Z triplets for each marker.
  %Invisible markers will be triplets of NaN-s
  [, framecounter(frames), positions(frames,:),] = DataGetNext3D_as_array;
end
OptotrakDeActivateMarkers; %Turn off the markers.
%Data is in positions(:,:), framecounter can be used to check sampling continuity.
optotrak_kill; %gracefully de-activate the system
```

A.2 Acquire data using the Optotrak's data buffer

Use of the toolbox in a typical visuomotor experiment. The following code illustrates which functions should be called in what order: Prior to an experiment, we recommended performing a new registration/alignment. For the recording of raw data, this requires a working data acquisition configuration file, and a working rigid body. We have included `this_works_with_the_ndi_cube.ini`, and `ndi_cube.rig` for the test cube sold by Northern Digital. This creates a new camera file, which should be used with `optotrak_set_up_system()`. Afterwards, the visual part of the experiment can be set up normally. Before starting a trial, initialize the data buffer file by calling `DataBufferInitializeFile ('temp.dat')`. When the trial starts, start recording: `DataBufferStart;`. At the end of the trial, make sure everything is written into the file: `optotrak_stop_buffering_and_write_out;`. After the end of the trial, stop the visual stimulus, and start the data conversion with `optotrak_convert_raw_file_to_position3d_array();` This snippet shows how to use the data buffer, and convert a raw data file into 3D positions. While buffering, it is possible to have access to real-time data simultaneously.

```
optotrak_startup; %this detects what hardware is available.
%The camera file contains the coordinate system alignment.
%The 'config file' contains data acquisition settings.
optotrak_set_up_system('path_to_camera_file', 'path_to_config_file');

DataBufferInitializeFile(0, 'raw_data.dat'); %Optotrak raw data goes here
DataGetNext3D_as_array; %Call this first if framecounter is to be used!
DataBufferStart; %This begins the recording

%The program can do something else in the meantime.
%Otherwise, this will wait until the buffering is finished.
%Real-time data is accessible while buffering.
  [, framecounter, real_time_positions,] = DataGetLatest3D_as_array;
  %framecounter can be used to track time and therefore marker speed.

optotrak_stop_buffering_and_write_out;

[, positions] = optotrak_convert_raw_file_to_position3d_array('raw_data.dat');
%in 'positions', every row is a frame. Just like real-time data,
%marker coordinates are in X-Y-Z triplets as before.

%We can visualise a single frame or an entire trial using:
optotrak_plot_marker_positions(positions); %shows positions in a 3D volume
%It is possible to extract marker/rigid body data using this function:
%Say, markers 1, 2, and 4 are useful, so we can extract them.
useful_markers = optotrak_get_selected_triplet([1:2, 4], positions)
optotrak_kill; %Shut down the system gracefully.
```

References

Blinch, J., 2011. Controlling Optotrak from Matlab. (retrieved November 2017). https://motorbehaviour.wordpress.com/2011/09/02/controlling-optotrak-from-matlab/.

Camponogara, I., Volcic, R., 2017. On-line adjustments of grasping movements under visual, haptic and visuo-haptic guidance. J. Vis. 17, 460.

Chapman, C., 2012. Optotrak Motion Tracking System. (retrieved November 2017). http://real.psych.ubc.ca/index.php/Optotrak_Motion_Tracking_System.

Fantoni, C., Caudek, C., Domini, F., 2010. Systematic distortions of perceived planar surface motion in active vision. J. Vis. 10, 12.

Franz, V., 2004. The Optotrak Toolbox. (retrieved November 2017). http://www.ecogsci.cs.uni-tuebingen.de/OptotrakToolbox.

Franz, V., Hesse, C., Kollath, S., 2009. Visual illusions, delayed grasping, and memory: no shift from dorsal to ventral control. Neuropsychologia 47, 1518–1531.

Hesse, C., Franz, V.H., 2009. Corrective processes in grasping after perturbations of object size. J. Motor Behav. 41, 253–273.

Kleiner, M., Brainard, D., Pelli, D., Ingling, A., Murray, R., Broussard, C., et al., 2007. What's new in psychtoolbox-3. Perception 36, 1.

van der Kooij, K., Brenner, E., van Beers, R.J., Schot, W.D., Smeets, J.B., 2013. Alignment to natural and imposed mismatches between the senses. J. Neurophysiol. 109, 1890–1899.

Volcic, R., Domini, F., 2016. On-line visual control of grasping movements. Exp. Brain Res. 234, 2165–2177.

Volcic, R., Fantoni, C., Caudek, C., Assad, J.A., Domini, F., 2013. Visuomotor adaptation changes stereoscopic depth perception and tactile discrimination. J. Neurosci. 33, 17081–17088.