(Click here for the web formatted version)

NaturalPoint Tracking Toolkits

Users Manual version 2.0

Complete Table Of Contents

1. Introduction

1.1 Forward

For providing VRPN, NaturalPoint would like to thank the NIH National Research Resource in Molecular Graphics and Microscopy at the University of North Carolina at Chapel Hill, supported by the NIH National Center for Research Resources and the NIH National Institute of Biomedical Imaging and Bioengineering.

1.2 About NaturalPoint

NaturalPoint Inc. is pleased to provide you with superior optical tracking products, we hope that you enjoy using your NaturalPoint Tracking Toolkits.

NaturalPoint
33872 SE Eastgate Circle
Corvallis, OR 97333
Telephone: 541-753-6645
Fax: 541-753-6689
www.naturalpoint.com

2. How to use the Manual

Quick Start

It is strongly recommended to read this manual before using the Tracking Toolkits optical tracking product. To begin using the product as soon as possible without reading the full manual then you can start quickly by following these instructions :

- ◊ Read the Installation section and follow the instructions described there, otherwise your Tracking Tools application and OptiTrack Cameras may not work.
- ◊ Once the software is installed, connect the OptiTrack cameras to the USB ports of your computer.
- ◊ Activate the Tracking Tools License
- ◊ Launch the Tracking Tools software using the shortcut on your desktop.

Further Reading

The Tracking Tools software is a powerful optical tracking solution with a number of options designed to help you get the best performance. It is recommended that you read Section 4 ("Using Tracking Tools Software"), Section 5 ("Using the Tracking Tools API") to better understand how the product works.

3. Getting Started

3.1 Minimum System Requirements

◊ 3+ OptiTrack cameras
◊ Windows XP SP2 or Windows Vista
◊ .Net 3.0 Runtime
◊ VC2005 Runtime
◊ 1.5 GHz Processor
◊ 256 MB of RAM
◊ 20 MB of free hard disk space
◊ USB 2.0 Hi-Speed port

3.2 What's Included

OptiTrack SOFTWARE ACTIVATION CODE

Enter these codes in the OptiTrack software to activate it, or activate the software online at www.optitrack.com/activate

(These codes are specific to your OptiTrack hardware. If you lose the card, have your OptiTrack camera serial number ready, and call or email NaturalPoint.)

LICENSE SERIAL NUMBER

LICENSE HASH CODE

Questions? Visit www.OptiTrack.com
Call us at 1.888.965.0435 or email support@naturalpoint.com

Tracking Tools license card

Quickstart guide

Online Software Download

3.3 Hardware Compatibility

The Tracking Tools software can only be used with OptiTrack FLEX:C120, OptiTrack FLEX:V100, and OptiTrack SLIM:V100 hardware. Older camera models are not compatible and will not work with this software.

3.4 Software and Hardware Installation

For best results it is suggested to install all software before you connect the OptiTrack cameras.

3.4.1 Software Installation

NOTE: Windows XP and Vista users must be logged in as an administrator. If you only have one user on your computer, you most likely already have administrator privileges.

1. Insert the included NaturalPoint software CD into your CD drive. Wait for the install program to start. If the install program does not start within a few minutes, open "My Computer" then double click on the CD drive icon, and double click again on the Setup file.

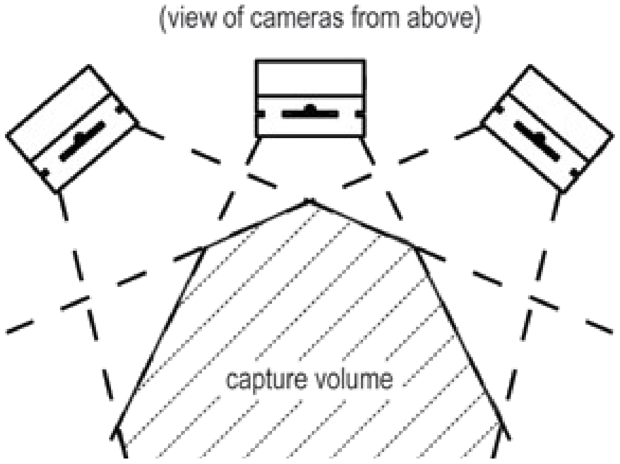
2. There are two software packages to install, the OptiTrack SDK and the Tracking Toolkits. Start with the OptiTrack SDK installer, once it has completed it will automatically start the Tracking Toolkits installer for you.
   Note: *The Tracking Toolkits will detect if you are missing required dependencies such as .Net 3.0 and install them for you.*

3. Follow the software installation instructions on the screen.
   Note: *Windows may display a warning message about the drivers not being signed. Click YES to accept the drivers, the drivers WILL NOT harm your system.*

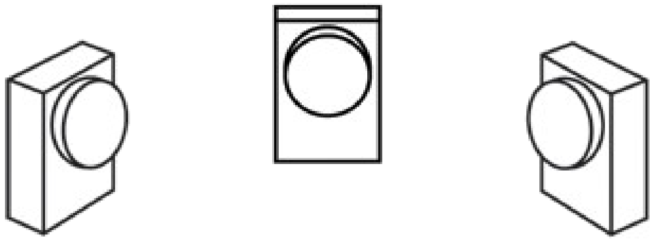
4. Tracking Tools software icon will appear on your desktop.

3.5 Camera Placement

In order to track markers, multiple OptiTrack cameras must be arranged to have overlapping fields of view. This will create an area called a *capture volume* in which tracking can occur. When possible, secure the cameras firmly in place. Whenever the cameras are moved it is necessary to recalibrate them.



For best calibration and tracking results, avoid placing the cameras all in the same plane. Instead position the cameras at different angles.



3.6 Camera Synchronization

Multiple OptiTrack cameras can be linked together to synchronize their exposure timing. Camera synchronization allows for greater precision and more robust tracking. The cameras are connected in a chain using Sync Cables*. Once the cables are connected and the cameras turned on, they will begin synchronizing automatically.
** Sync Cables sold separately*

1. Note : **All cameras must be of the same model type, otherwise synchronization will not work and should not be attempted.**

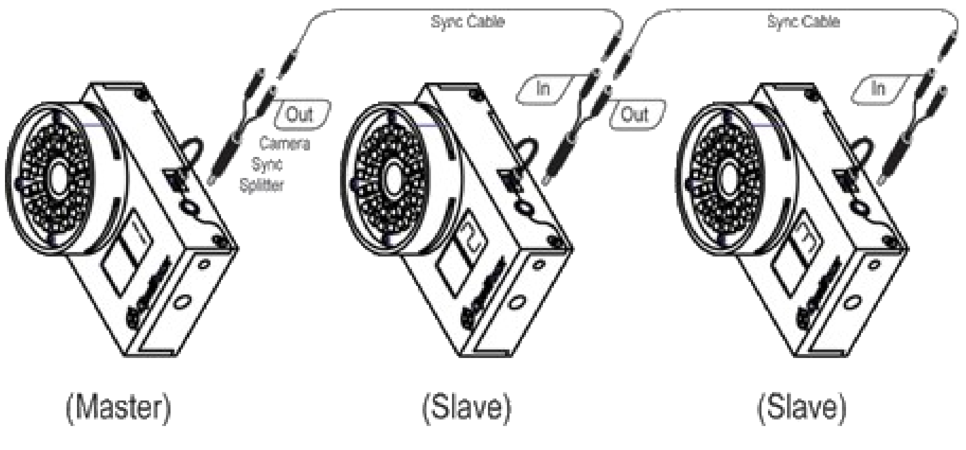
2. Connect all cameras to the computer via USB.

3. Plug the Camera Sync Splitter into each camera.

4. Chain the cameras together by connecting each camera's "out" connector to the next camera's "in" connector.

5. Do not form a loop, the last camera in the chain should **not** be connected to the first camera.

6. Start cameras in the software, the cameras will automatically synchronize.



3.7 License Activation

Before the Tracking Tools software can be used, you must first activate the included license (see License Card in section 3.2) for one of your OptiTrack cameras. Choose one of your cameras and use the serial number found on the back during the activation process. Activating the license enables you to use the software toolkit as long as the camera for which it was activated is connected to your computer.

There are two different methods for activating your license :

◊ Direct : Double click the OptiTrack License Activation tool shortcut on your desktop. Fill out the required details and then press the activate button. If the activation is successful a license will be generated and installed on your computer.

◊ Email : If your computer is behind a firewall which prevents the direct tool from working, you can activate your license on the OptiTrack website and have it emailed to you. Please visit www.optitrack.com/activate for more details.

License Activation tool :

Additional information can be found in our licensing FAQ online at http://www.naturalpoint.com/optitrack/support/activate/faq.html.

4. Using the Tracking Tools software

The Tracking Tools application is a robust, real-time 3D optical tracking solution. It provides calibration, 3D reconstruction and rigid body tracking across multiple OptiTrack cameras. Markers can be attached to multiple objects in known patterns (rigid bodies) allowing them to be tracked in full 6DOF (position and orientation). Tracking data can be accessed in real time through network streaming support or the easy to use software API. The information provided below will help you get the best results from your OptiTrack cameras and software.

4.1 Getting Started

Before using the Tracking Tools software make sure to have all of the software and hardware properly installed and licensed as described in Chapter 3. Once the installation is complete you can begin using the software by following these instructions :

1. Connect 3 or more cameras to the USB port of the computer and arrange them to set up a capture volume as described in Section 3.5 ("Camera Placement").

2. Start the Tracking Tools software.

3. Perform a camera calibration and test the capture volume. Save the resulting calibration for later use.

4. Rigid bodies for tracking can be defined using the 3D point cloud data, or by loading existing rigid body definitions from a file.

4.2 Working with Rigid Bodies

The Tracking Tools application creates a cloud of 3D points representing reflective markers which are visible to cameras within the capture volume. Rigid Bodies are clusters of reflective markers in a unique configuration which allow them to be identified and tracked in the cloud of 3D points. It is possible to track multiple rigid bodies at a time in full 6DOF (*position and orientation*).

The following section introduces best practices for defining rigid bodies and explains how to track them using the software.

4.2.1 Making Rigid Bodies

To get the best tracking performance there are several factors listed below to consider when making the physical rigid bodies. The configuration(s) will also be influenced by the tracking volume and camera placement, the size and shape of the physical object which the markers are mounted to, and the other objects which may be present within the tracking volume.

Marker Types

Spherical reflective markers will usually yield the most stable and accurate 3D tracking data. Hemispheric and flat markers are less desirable because their shape as it appears to the camera may change when it rotates, this can introduce errors when calculating their center of mass and position.

When using small rigid bodies, try to use small markers to reduce the chances of one marker occluding (*blocking from view of a camera*) other markers.

When tracking at larger distances from the camera, using larger markers can help improve the resolution of the tracking. Marker sizes typically range from 10 to 25 millimeters and larger.

NaturalPoint offers a range of high quality pre-made reflective markers for sale, for more details see Section 7 ("Cameras and Accessories").

Marker Quantity

Three markers is the minimum required to define a rigid body. A larger number of markers can be used to increase the precision and reduce likelihood of the rigid body flipping. Using more than three markers also provides redundancy, then tracking can occur even when some of the assigned markers are not visible (*the minimum visible for tracking is three*).

Rigid Body Sizing

The physical arrangement of the markers should not be too small and should not have markers too close together. Markers closer than 6 millimeters can result in the following issues :

◊ Markers are more likely to overlap or occlude one another when seen from a camera, resulting in incorrect 3D position data or markers failing to be tracked.

◊ If the point cloud tracking residual value is too high, or the calibration is poor, then rays for adjacent markers could be grouped together incorrectly resulting in misidentified marker locations and/or falsely reported markers.

Marker Arrangement
For 3-marker rigid bodies, using asymmetrical marker arrangements improves the tracking reliability. Symmetrical triangles make it harder for the software to identify the correct orientation of a rigid body and increases the likelihood of arriving at an incorrect solution . This can often manifest as the rigid body flipping on its axis between frames.

When multiple rigid bodies will be tracked at the same time, avoid making rigid bodies which are too similar to each other. Making them individually unique with different marker arrangements and sizes will reduce the likelihood of misidentification and swapping.

4.3 Real-time Tracking

4.3.1 Exporting and Streaming Tracking Data
The 3D point cloud and rigid body tracking data can be exported using files or streamed in real-time for use in other applications. The details below explain these features.

Exporting to CSV format
Recorded tracking data with rigid body locations can be exported in the CSV format for use in other applications. The exported file will contain comments at the top which describe the data formatting.

Streaming tracking data
Real-time and recorded data can be streamed over the network for use in other applications. Several different methods of streaming are supported including industry standards VRPN and Trackd. All of the streaming methods are built on a network transport which allows the data to be made available on the local computer as well as remote network computers.

Before trying to stream, make sure that either recorded data is currently loaded or the cameras are tracking in real-time. It is also necessary to have rigid bodies defined.

Since data is streamed over TCP/IP on the network, it may be necessary to review your firewall settings. If the firewall is blocking traffic it may prevent the software from properly sending the data or the client applications from reading it.

◊ **NaturalPoint Engine (NatNet)** : The NaturalPoint streaming engine is a custom streaming implementing which can be used to access real-time and recorded tracking data over the network. NaturalPoint provides sample source code for writing a client which receives the streaming data. The standard sample client can be used to verify that streaming is functioning properly.

Streams : rigid bodies and markers

Network Details : port 1510, multigast group 1001, UDP multicast

◊ **VRPN Engine** : VRPN is an open source set of classes and a protocol which can be used to access real-time and recorded tracking data over the network. It features low latency and has built-in auto-renegotiation if a connection is temporarily dropped. NaturalPoint provides sample source code for a listener that receives tracking data. Additional sample code is available from the VRPN website .

In VRPN objects are identified and accessed by a name. In the software's implementation of VRPN the name used is the one assigned to a rigid body.

If needed, the VRPN port can be changed from the default by editing the port number setting.

Streams : rigid bodies

Network Details : selectable port, default port 3883, TCP + UDP

◊ **Trackd Engine** : Trackd is a standard created by VRCO/Mechdyne which can be used to access real-time and recorded tracking data over the network. NaturalPoint provides sample source code for a listener that receives tracking data. Additional information is available from the VRCO website.

The Trackd server must be configured before data can be streamed, use the following steps to get it set up.

1. A Trackd module (dll) and sample configuration file are provided by NaturalPoint.
2. Configure Trackd by calling it the "naturalpointtracker".
3. Define the number of rigid bodies desired to track.
4. The streaming host defaults to localhost, but can be changed to a different network address.
5. Once the configuration is complete, run the Trackd server.

Streams : rigid bodies

Network Details : port 4994, TCP + UDP

5. Using the Tracking Tools API

In order to use Tracking Tools tracking in your own applications, you will need to implement support for it using the Tracking Tools API. The Tracking Tools API is written as a set of C/C++ function calls and a loadable DLL. The following section provides an overview of the system architecture as well as specific detail about the API calls.

5.1 Getting Started

Only a small amount of code needs to be written in order to begin tracking rigid bodies. The following outlines the initialization procedure :

1. Make sure the cameras have been calibrated with the result saved to a file.
2. Initialize the rigid body tracking using TT_Initialize().
3. Load an existing calibration result or Tracking Tools project using TT_LoadCalibration() or TT_LoadProject().
4. Load existing rigid body definitions using TT_LoadTrackables().

At this point, the cameras should be initialized and collecting frame information. The main loop of the application should poll for frames using TT_Update() or TT_UpdateSingleFrame().

When data processing is complete, call the TT_Shutdown().

5.2 Tracking Tools API Calls

5.2.1 Tracking Tools Startup and Shutdown

5.2.1.1 TT_Initialize( )

The TT_Initialize( ) function attempts to initialize the Tracking Tools tracking API. It should be called before attempting to use other components of the API. It returns information about whether or not it succeeded.

NPRESULT TT_Initialize()

Parameters

- none

Parameters

Value	**Meaning**
NPRESULT_SUCCESS	Method succeeded
InvalidLicense	A valid Tracking Tools license was not found

5.2.1.2 TT_Shutdown( )

The TT_Shutdown( ) function attempts to shutdown the Tracking Tools tracking API. It should be called before the application using the Tracking Tools tracking API closes. It returns information about whether or not it succeeded.

Note : If needed, applications may call TT_Initialize() again after having previously called TT_Shutdown(). While applications will typically call TT_Initialize() on startup and TT_Shutdown() before they close, some applications may choose a different implementation. Also see TT_FinalCleanup() for releasing resources used by the API.

NPRESULT TT_Shutdown()

Parameters

- none

Parameters

Value	**Meaning**
NPRESULT_SUCCESS	Method succeeded

5.2.1.3 TT_FinalCleanup( )

The TT_FinalCleanup() function should be called when applications are done using the Tracking Tools API services. This is typically when an application is closing, but can be performed sooner to recover resources when the Tracking Tools API is not being used.

This function shuts down the camera device driver and ensures all the driver threads are terminated properly.

Note : If TT_Shutdown() hasn't been called first, TT_Shutdown() will be called automatically by TT_FinalCleanup().

Note : TT_FinalCleanup() is optional, but recommened, in order to ensure orderly shutdown of cameras and other resources.

NPRESULT TT_FinalCleanup()

Parameters

- none

Parameters

Value	**Meaning**
NPRESULT_SUCCESS	Method succeeded

5.2.2 Tracking Tools Interface

5.2.2.1 TT_LoadCalibration()

The TT_LoadCalibration() function attempts to load a calibration profile stored in a file. It is passed the path to the profile and returns information about whether or not it succeeded.

NPRESULT TT_LoadCalibration(const char *filename)

Parameters

- filename
  [in] the path to the location of the camera calibration

Parameters

Value	**Meaning**
NPRESULT_SUCCESS	Method succeeded
LoadFailed	The file or path specified is not valid

5.2.2.2 TT_Update()

The TT_Update() function processes all outstanding camera data, triangulates, solves rigid bodies, and streams tracking data. This function should be called often, perhaps in the main loop of the software using the Tracking Tools API. Ideally this would be called periodically every 1ms to 10ms. If the rate at which TT_Update() is called drops below 50ms there is danger of losing tracking data. It returns information about whether or not it succeeded.

NPRESULT TT_Update()

Parameters

- none

Parameters

Value	**Meaning**
NPRESULT_SUCCESS	Method succeeded

InvalidLicense	A valid Rigid Body license was not found
NoFrameAvailable	No tracking data was found

5.2.2.3 TT_UpdateSingleFrame()

The TT_UpdateSingleFrame() function processes a single frame of outstanding camera data, triangulates, solves rigid bodies, and streams tracking data. This function should be called often, perhaps in the main loop of the software using the Tracking Tools API. Ideally this would be called periodically more often than every 10ms to ensure no loss of tracking data. It returns information about whether or not it succeeded.

NPRESULT TT_UpdateSingleFrame()

Parameters

- none

Parameters

Value	Meaning
NPRESULT_SUCCESS	Method succeeded
InvalidLicense	A valid Rigid Body license was not found
NoFrameAvailable	No tracking data was found

5.2.2.4 TT_LoadTrackables()

The TT_LoadTrackables() function loads a set of rigid body definitions from a file for tracking. It returns information about whether or not it succeeded.

NPRESULT TT_LoadTrackables(const char *filename)

Parameters

- filename
  [in] the path to the location of the rigid body definition file

Parameters

Value	Meaning
NPRESULT_SUCCESS	Method succeeded
Failed	The trackables failed to load

5.2.2.5 TT_AddTrackables()

The TT_AddTrackables() function loads a set of rigid body definitions from a file for tracking, however it does not clear the existing list of Trackables beforehand like TT_LoadTrackables() does. It returns information about whether or not it succeeded.

NPRESULT TT_AddTrackables (const char *filename)

Parameters

- filename
  [in] the path to the location of the rigid body definition file

Parameters

Value	**Meaning**
NPRESULT_SUCCESS	Method succeeded
Failed	The trackables failed to load

5.2.2.6 TT_CreateTrackable()

The TT_CreateTrackable() function creates a trackable based on the marker count and marker list provided. The MarkerList is expected to contain a list of marker coordinates in the order: x1,y1,z1,x2,y2,z2,etc...xN,yN,zN. It returns information about whether or not it succeeded.

NPRESULT TT_CreateTrackable(const char* Name, int ID, int MarkerCount, float *MarkerList)

Parameters

- Name
  [in] the Name for the newly created trackable
- ID
  [in] the ID for the newly created trackable
- MarkerCount
  [in] the number of markers used to create the trackable, this should match the number provided in MarkerList
- MarkerList
  [in] a pointer to an array of floats of marker coordinates for creating the trackable in the order: x1,y1,z1,x2,y2,z2,etc...xN,yN,zN

Parameters

Value	Meaning
NPRESULT_SUCCESS	Method succeeded
Failed	Creation of the trackable failed

5.2.2.7 TT_RemoveTrackable()

The TT_RemoveTrackable() function removes an individual rigid body from the list of tracked rigid bodies. It returns information about whether or not it succeeded.

NPRESULT TT_RemoveTrackable(int Index)

Parameters

- index
  [in] the index number of the desired rigid body. the index is zero based

Parameters

Value	Meaning
NPRESULT_SUCCESS	Method succeeded
Failed	The trackable failed to delete

5.2.2.8 TT_LoadProject()

The TT_LoadProject() function loads a Tracking Tools project.
NPRESULT TT_LoadTrackables(const char *filename)

Parameters

- filename
  [in] the path to the location of the rigid body definition file

Parameters

Value	Meaning
NPRESULT_SUCCESS	Method succeeded
Failed	The trackables failed to load

5.2.2.9 TT_SaveProject()

The TT_SaveProject() function saves a Tracking Tools project.
NPRESULT TT_SaveTrackables(const char *filename)

Parameters

- filename
  [in] the path to the location of the rigid body definition file

Parameters

Value	**Meaning**
NPRESULT_SUCCESS	Method succeeded
Failed	The trackables failed to load

5.2.3 Tracking Tools Streaming

5.2.3.1 TT_StreamTrackd()

The TT_StreamTrackd() function starts and stops streaming tracking data over the Trackd interface. It returns information about whether or not it succeeded.

NPRESULT TT_StreamTrackd(bool enabled)

Parameters

- enabled
  [in] starts and stops the streaming, true starts and false stops

Parameters

Value	**Meaning**
NPRESULT_SUCCESS	Method succeeded

5.2.3.2 TT_StreamVRPN()

The TT_StreamVRPN() function starts and stops streaming tracking data over the VRPN interface. It returns information about whether or not it succeeded.

NPRESULT TT_StreamVRPN(bool enabled, int port)

Parameters

- enabled
  [in] starts and stops the streaming, true starts and false stops

- port
  [in] selects the network port to broadcast over

Parameters

Value	**Meaning**
NPRESULT_SUCCESS	Method succeeded

5.2.3.3 TT_StreamNP()

The TT_StreamNP() function starts and stops streaming tracking data over the NaturalPoint NATNET streaming interface. It returns information about whether or not it succeeded.

NPRESULT TT_StreamNP(bool enabled)

Parameters

- enabled
  [in] starts and stops the streaming, true starts and false stops

Parameters

Value	**Meaning**
NPRESULT_SUCCESS	Method succeeded

5.2.4 Frame

5.2.4.1 TT_FrameMarkerCount()

The TT_FrameMarkerCount() function returns the number of 3D point cloud markers found in the current frame.

int TT_FrameMarkerCount()

Parameters

- (returns)
  [out] the number of 3D point cloud markers found in the current frame

5.2.4.2 TT_FrameMarkerX()

The TT_FrameMarkerX() function returns the X axis position in meters of the selected 3D point cloud marker.

float TT_FrameMarkerX(int index)

Parameters

- (returns)
  [out] the X position in meters of the selected 3D point cloud marker
- index
  [in] the index number of the desired marker. the index is zero based

Parameters

Value	Meaning
0	No frame data was found

5.2.4.3 TT_FrameMarkerY()

The TT_FrameMarkerY() function returns the Y axis position in meters of the selected 3D point cloud marker.

float TT_FrameMarkerY(int index)

Parameters

- (returns)
  [out] the Y position in meters of the selected 3D point cloud marker
- index
  [in] returns the index number of the desired marker. the index is zero based

Parameters

Value	Meaning
0	No frame data was found

5.2.4.4 TT_FrameMarkerZ()

The TT_FrameMarkerZ() function returns the Z axis position in meters of the selected 3D point cloud marker.

float TT_FrameMarkerZ(int index)

Parameters

- (returns)
  [out] the Z position in meters of the selected 3D point cloud marker
- index
  [in] returns the index number of the desired marker. the index is zero based

Parameters

Value	Meaning
0	No frame data was found

5.2.5 Rigid Body Control

5.2.5.1 TT_IsTrackableTracked()

The TT_IsTrackableTracked() function returns information about whether the selected rigid body is found in the current frame.

bool TT_IsTrackableTracked(int index)

Parameters

- (returns)
  [out] true if the selected rigid body is found in the current frame
- index
  [in] the index number of the desired rigid body. the index is zero based

5.2.5.2 TT_TrackableLocation()

The TT_TrackableLocation() function returns the position and orientation of the selected rigid body. TT_IsTrackableTracked() should be checked first to determine whether the rigid body was tracked in the current frame, otherwise the data may be stale.

void TT_TrackableLocation(int RigidIndex,
float *x, float *y, float *z,
float *qx, float *qy, float *qz, float *qw,
float *yaw, float *pitch, float *roll)

Parameters

- RigidIndex
  [in] the index number of the desired rigid body. the index is zero based
- x
  [out] receives the X axis position in meters of the selected rigid body
- y
  [out] receives the Y axis position in meters of the selected rigid body

- z
  [out] receives the Z axis position in meters of the selected rigid body
- qx
  [out] receives the quaternion x value of the selected rigid body
- qy
  [out] receives the quaternion y value of the selected rigid body
- qz
  [out] receives the quaternion z value of the selected rigid body
- qw
  [out] receives the quaternion w value of the selected rigid body
- heading
  [out] receives the yaw orientation value in degrees of the selected rigid body
- attitude
  [out] receives the pitch orientation value in degrees of the selected rigid body
- bank
  [out] receives the roll orientation value in degrees of the selected rigid body

5.2.5.3 TT_ClearTrackableList()

The TT_ClearTrackableList() function removes all of the currently loaded rigid body definitions.

void TT_ClearTrackableList()

Parameters

- none

5.2.5.4 TT_GetTrackableCount()

The TT_GetTrackableCount() function returns the number of currently loaded rigid body definitions.

int TT_GetTrackableCount()

Parameters

- (returns)
  [out] the number of currently loaded rigid body definitions.

5.2.5.4 TT_GetTrackableID()

The TT_GetTrackableID() function returns the ID for the selected rigid body.

int TT_GetTrackableID(int index)

Parameters

- (returns)
  [out] the ID of the selected rigid body
- index
  [in] the index of the desired rigid body. the index is zero based

5.2.5.5 TT_SetTrackableID()

The TT_SetTrackableID() function changes the ID for the selected rigid body.

void TT_SetTrackableID(int index, int ID)

Parameters

- index
  [in] the index of the desired rigid body. the index is zero based
- ID
  [in] the new ID value for the selected rigid body

5.2.5.6 TT_GetTrackableName()

The TT_GetTrackableName() function returns the ID for the selected rigid body.

const char* TT_GetTrackableName(int index)

Parameters

- (returns)
  [out] the name of the selected rigid body
- index
  [in] the index of the desired rigid body. the index is zero based

5.2.5.7 TT_SetTrackableEnabled()

The TT_SetTrackableEnabled() function controls whether the selected rigid body is enabled for tracking.

void TT_SetTrackableEnabled(int index, bool enabled)

Parameters

- index
  [in] the index of the desired rigid body. the index is zero based

- enabled
  [in] a value of true enables the rigid body, a value of false disables it

5.2.5.8 TT_TrackableEnabled()

The TT_TrackableEnabled() function returns information about whether the selected rigid body is enabled for tracking.

bool TT_TrackableEnabled(int index)

Parameters

- (returns)
  [out] a value of true means the rigid body is enabled, a value of false means it is disabled
- index
  [in] the index of the desired rigid body. the index is zero based

5.2.5.9 TT_TrackableMarkerCount()

The TT_TrackableMarkerCount() function returns the number of markers used in the selected rigid body definition.

int TT_TrackableMarkerCount(int index)

Parameters

- (returns)
  [out] the number of markers used in the selected rigid body
- index
  [in] the index of the desired rigid body. the index is zero based

5.2.5.10 TT_TrackableMarker()

The TT_TrackableMarker() function returns the position of the selected marker in the selected rigid body definition. Marker positions are in meters and relative to the pivot point or center of mass of the rigid body.

void TT_TrackableMarker(int RigidIndex, int MarkerIndex, float *x, float *y, float *z)

Parameters

- RigidIndex
  [in] the index of the desired rigid body. the index is zero based
- MarkerIndex
  [in] the index of the desired marker. the index is zero based

- x
  [out] the X axis position of the marker in meters
- y
  [out] the Y axis position of the marker in meters
- z
  [out] the Z axis position of the marker in meters

5.2.5.11 TT_TrackableTranslatePivot()

The TT_TrackableTranslatePivot() function translates the pivot point for the selected rigid body. The translation amounts are in meters and specify the amount to translate relative to the current pivot point or center of mass of the rigid body.

NPRESULT TT_TrackableTranslatePivot(int index, float x, float y, float z)

Parameters

- int
  [in] the index of the desired rigid body. the index is zero based
- x
  [out] the amount to translate the pivot point on the X axis in meters
- y
  [out] the amount to translate the pivot point on the Y axis in meters
- z
  [out] the amount to translate the pivot point on the Z axis in meters

5.2.6 Point Cloud Interface

5.2.6.1 TT_CameraCount()

The TT_CameraCount() function returns the number of connected cameras.

int TT_CameraCount()

Parameters

- (returns)
  [out] the number of connected cameras

5.2.6.2 TT_CameraXLocation()

The TT_CameraXLocation() function returns the X position of the selected camera in relation to the coordinate system origin.

float TT_CameraXLocation(int index)

Parameters

- (returns)
  [out] the X position of the selected camera in meters
- index
  [in] the index of the desired camera. the index is zero based

5.2.6.3 TT_CameraYLocation()

The TT_CameraYLocation() function returns the Y position of the selected camera in relation to the coordinate system origin.

float TT_CameraYLocation(int index)

Parameters

- (returns)
  [out] the Y position of the selected camera in meters
- index
  [in] the index of the desired camera. the index is zero based

5.2.6.4 TT_CameraZLocation()

The TT_CameraZLocation() function returns the Z position of the selected camera in relation to the coordinate system origin.

float TT_CameraZLocation(int index)

Parameters

- (returns)
  [out] the Z position of the selected camera in meters
- index
  [in] the index of the desired camera. the index is zero based

5.2.6.5 TT_CameraOrientationMatrix()

The TT_CameraOrientationMatrix() function returns the selected element of the orientation matrix of the selected camera.

float TT_CameraOrientationMatrix(int camera, int index)

Parameters

- (returns)
  [out] the selected element of the orientation matrix
- camera
  [in] the index of the desired camera. the index is zero based
- index
  [in] index of the item in the array to retrieve. there are 9 elements in the array, valid inputs range from 0 to 8.

5.2.6.6 TT_CameraName()

The TT_CameraName() function returns the selected element of the orientation matrix of the selected camera.

float TT_CameraName(int index)

Parameters

- (returns)
  [out] the camera name
- index
  [in] the index of the desired camera. the index is zero based

5.2.6.7 TT_CameraMarkerCount()

The TT_CameraMarkerCount() function returns the number of 2D markers seen by the camera for the current frame.

int TT_CameraMarkerCount(int CameraIndex)

Parameters

- (returns)
  [out] the number of 2D markers
- result
  [in] the index of the desired camera. the index is zero based

5.2.6.8 TT_CameraMarker()

The TT_CameraMarker() function returns coordinates of a 2D marker for the current frame.

bool TT_CameraMarker(int CameraIndex, int MarkerIndex, float &x, float &y)

Parameters

· (returns)
[out] boolean true indicates successful population of x and y input parameters
· CameraIndex
[in] the index of the desired camera. the index is zero based
· MarkerIndex
[in] the index of the desired marker. the index is zero based
· x
[in] a float passed by reference to be populated with the x coordinate of the marker
· y
[in] a float passed by reference to be populated with the x coordinate of the marker

5.2.6.9 TT_CameraMarkerPredistorted()

The TT_CameraMarkerPredistorted() function returns coordinates of a 2D marker for the current frame. The location is predistorted by the intrinsic lens parameters of the camera to return the coordinates as if there was no lens distortion.

bool TT_CameraMarkerPredistorted(int CameraIndex, int MarkerIndex, float &x, float &y)

Parameters
· (returns)
[out] boolean true indicates successful population of x and y input parameters
· CameraIndex
[in] the index of the desired camera. the index is zero based
· MarkerIndex
[in] the index of the desired marker. the index is zero based
· x
[in] a float passed by reference to be populated with the x coordinate of the marker
· y
[in] a float passed by reference to be populated with the x coordinate of the marker

5.2.6.10 TT_SetCameraSettings()

The TT_SetCameraSettings() function allows you to set some of the camera settings, such as video mode, exposure, threshold, and illumination.

bool TT_SetCameraSettings(int CameraIndex, int VideoType, int Exposure, int Threshold, int Intensity)

Parameters
· (returns)
[out] boolean true indicates successful update of the camera settings

· CameraIndex
[in] the index of the desired camera. the index is zero based

· VideoType
[in] the desired in-camera video processing mode.
Valid values are:
0 = Segment Mode
1 = Grayscale Mode
2 = Object Mode
4 = Precision Mode
6 = MJPEG Mode (V100R2 only)

· Exposure
[in] the desired exposure setting for the camera.
Valid values are: 1-480

· Threshold
[in] the desired video threshold level for the camera. Pixels with intensities darker than this value will be filtered out when using processed video modes.
Valid values are: 0-255

· Intensity
[in] the desired IR LED setting for the camera. This should be set to 15 for almost all situations. The recommended method for reducing IR LED brightness is to lower the camera exposure setting, this has the effect of shortening the IR strobe duration.
Valid values are: 0-15

5.2.6.11 TT_SetCameraAEC()

The TT_SetCameraAEC() function allows automatic exposure control (AEC) in the camera to be enabled or disabled. When automatic exposure control is enabled the camera will attempt to optimize the exposure value to produce an image suitable for operator viewing (instead of marker tracking).

AEC should **not** be turned on when a camera is being used for marker tracking.

bool TT_SetCameraAEC(int CameraIndex, bool EnableAutomaticExposureControl)

Parameters

· (returns)
[out] boolean true indicates successful update of the camera AEC setting

· CameraIndex
[in] the index of the desired camera. the index is zero based

· EnableAutomaticExposureControl
[in] a value of true enables AEC, a value of false disables it. Default is **disabled**.

5.2.6.12 TT_SetCameraAGC()

The TT_SetCameraAGC() function allows automatic gain control (AGC) in the camera to be enabled or disabled. When automatic gain control is enabled the camera will attempt to optimize the gain value to produce an image suitable for operator viewing (instead of marker tracking). AGC should **not** be turned on when a camera is being used for marker tracking.

bool TT_SetCameraAGC(int CameraIndex, bool EnableAutomaticGainControl)

Parameters

· (returns)
[out] boolean true indicates successful update of the camera AGC setting

· CameraIndex
[in] the index of the desired camera. the index is zero based

· EnableAutomaticGainControl
[in] a value of true enables AGC, a value of false disables it. Default is **disabled**.

5.2.6.13 TT_SetCameraFilterSwitch()

The TT_SetCameraFilterSwitch() function toggles the camera between capturing infrared or visible light. Operating the camera in visible light mode is useful for operator viewing or scene reference video recording. The camera should **not** be in visible light mode when used for marker tracking.
Only cameras with the Filter Switcher feature are supported by this function.

bool TT_SetCameraFilterSwitch(int CameraIndex, bool EnableIRFilter)

Parameters

· (returns)
[out] boolean true indicates successful update of the camera FilterSwitch setting

· CameraIndex
[in] the index of the desired camera. the index is zero based

· EnableIRFilter
[in] a value of true selects infrared light mode, a value of false selects visible light mode. Default is **infrared light mode**.

5.2.6.14 TT_SetCameraHighPower()

The TT_SetCameraHighPower() function toggles the camera between Normal and High power IR LED mode. High power mode provides additional power to the built-in IR LEDs which increases the maximum tracking range or allows faster shutter speeds to be used at the same distance as Normal power mode.

V100R2 cameras must be directly connected to OptiHubs in order to use High power mode.

bool TT_SetCameraHighPower(int CameraIndex, bool EnableHighPowerMode)

Parameters

· (returns)
[out] boolean true indicates successful update of the camera High power setting

· CameraIndex
[in] the index of the desired camera. the index is zero based

· EnableHighPowerMode
[in] a value of true selects High power mode, a value of false selects Normal power mode. Default is **Normal power mode**.

5.2.6.15 TT_SetCameraMJPEGHighQuality()

The TT_SetCameraMJPEGHighQuality() function toggles the camera between Normal and High quality MJPEG video mode. When MJPEG High quality mode is enabled the camera uses less compression for the MJPEG video images. Less compression results in fewer visual artifacts, however it also consumes more USB bandwidth.

Only cameras with MJPEG video mode are supported by this function.

bool TT_SetCameraMJPEGHighQuality(int CameraIndex, int MJPEGQuality)

Parameters

· (returns)
[out] boolean true indicates successful update of the camera MJPEG High quality setting

· CameraIndex
[in] the index of the desired camera. the index is zero based

· MJPEGQuality
[in] the desired MJPEG quality mode Valid values are
50 = Normal quality MJPEG mode (default)
100 = High quality MJPEG mode

5.2.6.16 TT_SetCameraGrayscaleDecimation()

The TT_SetCameraGrayscaleDecimation() function specifies which spatial decimation setting (resolution down-sampling) for the camera to use when operating in raw grayscale video mode. Smaller down-sampled resolutions consume less USB bandwidth and may allow successful transmission of raw grayscale video frames in scenarios where full-frame grayscale is not possible.

bool TT_SetCameraGrayscaleDecimation(int CameraIndex, int Decimation)

Parameters

· (returns)
[out] boolean true indicates a successful update of the camera Grayscale Decimation setting

· CameraIndex
[in] the index of the desired camera. the index is zero based

· Decimation
[in] the desired Grayscale Spatial Decimation setting Valid values are
0 = Native resolution, no down sampling (default)
2 = 1/2 scale resolution (example: 640x480 downsampled to 320x240)
4 = 1/4 scale resolution (example: 640x480 downsampled to 160x120)

5.2.6.17 TT_CameraGrayscaleDecimation()

The TT_CameraGrayscaleDecimation() function returns the current spatial decimation setting (resolution down-sampling) used when the camera is operating in raw grayscale video mode.

TTAPI int TT_CameraGrayscaleDecimation(int CameraIndex)

Parameters

· (returns)
[out] current grayscale spatial decimation setting for the camera

Return values :
0 = Native resolution, no down sampling (default)

2 = 1/2 scale resolution (example: 640x480 downsampled to 320x240)
4 = 1/4 scale resolution (example: 640x480 downsampled to 160x120)

· CameraIndex
[in] the index of the desired camera. the index is zero based

5.2.6.18 TT_CameraFrameBuffer()

The TT_CameraFrameBuffer() function returns a rasterized image of the camera's view for the current frame. This function fills the provided buffer with an image of what is in the camera view. The resulting Image depends on what video mode the camera is in. If the camera is in grayscale mode, for example, a grayscale image is returned from this call.

bool TT_CameraFrameBuffer(int CameraIndex, int BufferPixelWidth, int BufferPixelHeight,
int BufferByteSpan, int BufferPixelBitDepth, unsigned char *Buffer)

Parameters

· (returns)
[out] boolean true indicates successful rasterization
· CameraIndex
[in] the index of the desired camera. the index is zero based
· BufferPixelWidth
[in] the width in pixels of the buffer being passed for rasterization
· BufferPixelHeight
[in] the height in pixels of the buffer being passed for rasterization
· BufferByteSpan
[in] the span of a single raster row in bytes of the buffer being passed. A value of zero is automatically populated with BufferPixelWidth*BufferPixelDepth
· BufferPixelBitDepth
[in] the pixel bit depth of the buffer being passed. Valid bit depths are 8, 16, 24, 32
· Buffer
[in] the pointer to the buffer to receive the rasterized camera imager

5.2.7 Return Code Processing

5.2.7.1 TT_GetResultString()

The TT_GetResultString() function returns a plain text message for status codes returned from other functions in the Rigid Body API.

const char *TT_GetResultString(NPRESULT result)

Parameters

· (returns)
 [out] the plain text message associated with the selected return code
· result
 [in] the status code returned from another function in the Tracking Tools API

6. Tips and Tricks

6.1 Tracking Environment

The Tracking Tools application is designed to work in a wide variety of conditions, and the current generation of OptiTrack cameras are more robust and reliable than ever before. There are, however, a number of things that you can do to optimize their performance :

Distance between the cameras and the user :
The optimum range for tracking markers with OptiTrack cameras is between 0.5 to 6 meters.

Lighting :
We recommend turning off or dimming lights in the room and removing any highly reflective materials that are directly in the view of the OptiTrack cameras.

6.2 Tracking Markers

To achieve the best 3D tracking performance NaturalPoint recommends the use of spherical reflective 3D markers which are available from our online store. The markers will be detected most easily when they are clean and there has not been any abrasion to the surface of the marker. If the reflective surface of the markers will come into frequent contact with skin or other surfaces then it is recommended to replace the markers on a regular basis.

7. Cameras and Accessories

NaturalPoint offers a complete line of high quality cameras and accessories to complement your Tracking Tools software. Please visit our online store to find out more at : OptiTrack online store catalogue

8 Software Updates

In order to better serve your needs we continually update the Tracking Toolkits software, you may download these updates free of charge from www.naturalpoint.com/optitrack.

To download and install new software, please follow these steps

1. Go to www.naturalpoint.com/optitrack and click on the Support link at the top of the page, then go down to the download section.

2. If the version of software listed for your product is more recent than the version of the software installed on your computer, then download and save the updated installer file to your desktop or temp folder. If a newer version is not available, then there is no need to update your software.

3. Disconnect any OptiTrack cameras from your computer and ensure that no Tracking Toolkits software is running.

4. Use the "Add and Remove Programs" feature of the Windows Control Panel to un-install your existing Tracking Toolkits software.

5. Double click on the downloaded installer .exe file to install the new software. After the program icon reappears on your desktop, you may reconnect OptiTrack cameras to your computer and run the updated software.

9 Troubleshooting

If you are experiencing difficulty using this product, please call 1-541-753-6645 or visit the NaturalPoint website at www.naturalpoint.com, or our online forums at forum.naturalpoint.com for advanced customer support.