Tutorial of OptiTrack Arena Motion Capture System
2nd : Implementation of System

Motivation of this tutorial is to make readers understand how to use Motion Capture System using Arena software and how to stream data using NATNET SDK will be presented in a next tutorial.

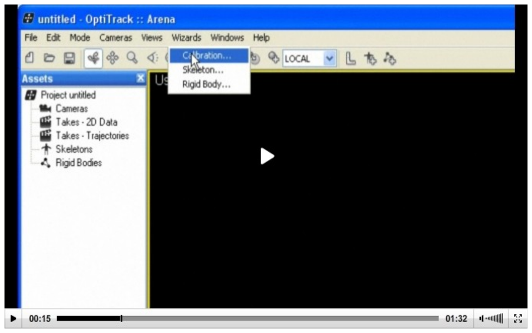
1. Calibration

This section show how to calibrate a Motion Capture System. This process is necessary only when you changed position of Cameras or you want to change coordinate of capture volume. Therefore, if your purpose of reading this tutorial is to want to capture some targets using Arena motion capture system, you do not need to read this section.

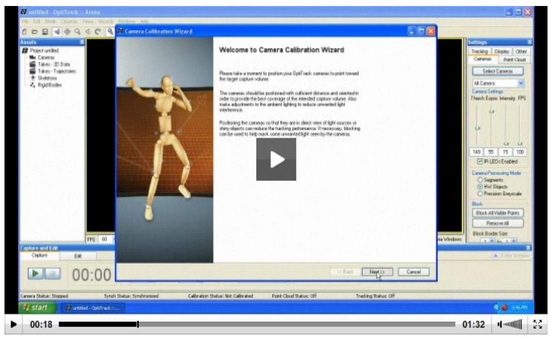
Step 1) Blocking Bad Segments

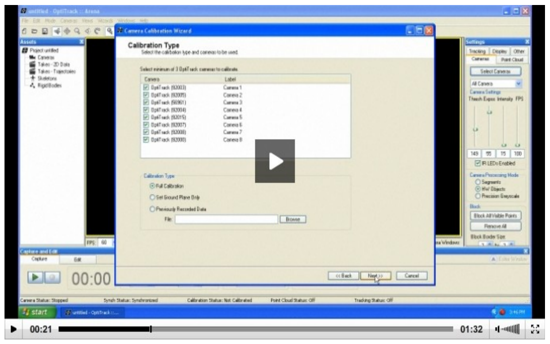
If there are unnecessary bright colored or blinking objects in capture area, these objects can generate many noise in Arena motion capture system. Easiest way to eliminate them might be removing them by yourself from capture area. However, in case of reflective dots on floor, we can not remove from capture area. This section is for those cases.

1st, go toolbar which is located in top of window, then, select Wizards tab. Click Calibration.

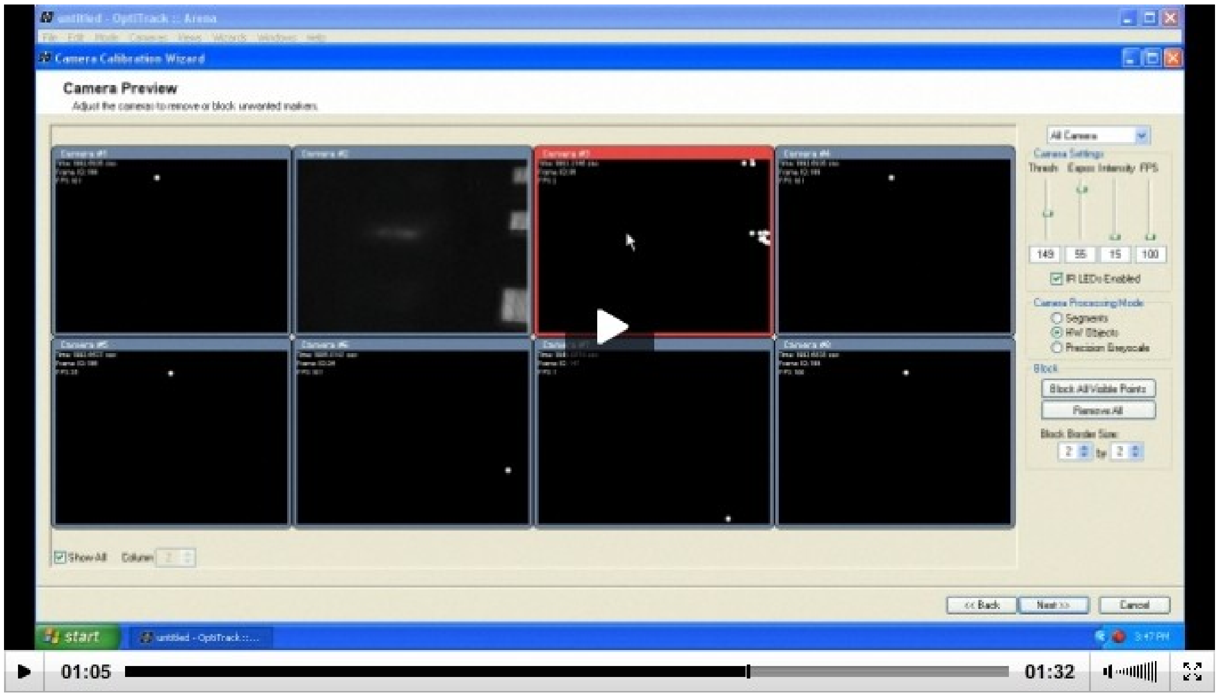


Click Next and choose Full calibration, 3 Marker Ward and 500mm(if you use long one)


Camera Calibration Wizard
Welcome to Camera Calibration Wizard
00:00
00:18
01:32



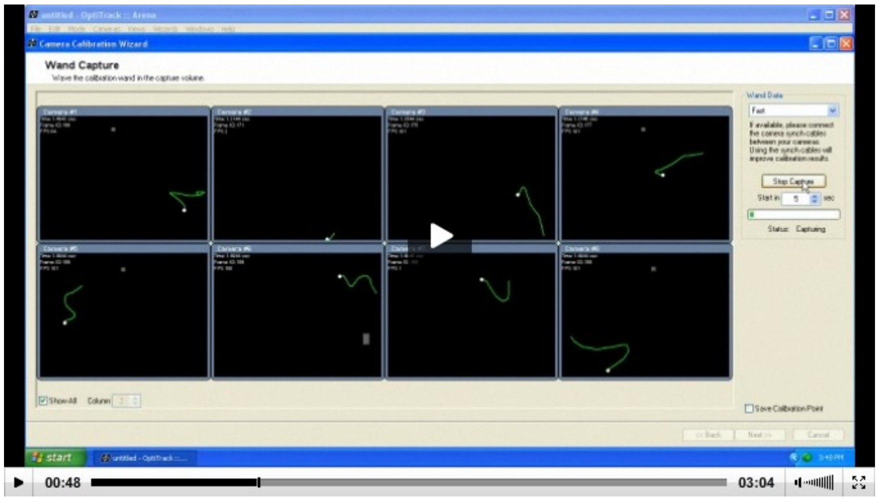
Now you can see window below and block visible markers by clicking Block All Visible Points button on right of window. At this time, there should be not markers which you want to capture in capture area. If you block those markers, you can not capture them and system will not work as you want.



You can also block individual points by manually clicking and dragging those areas. If you right click each small window, you can see many options.

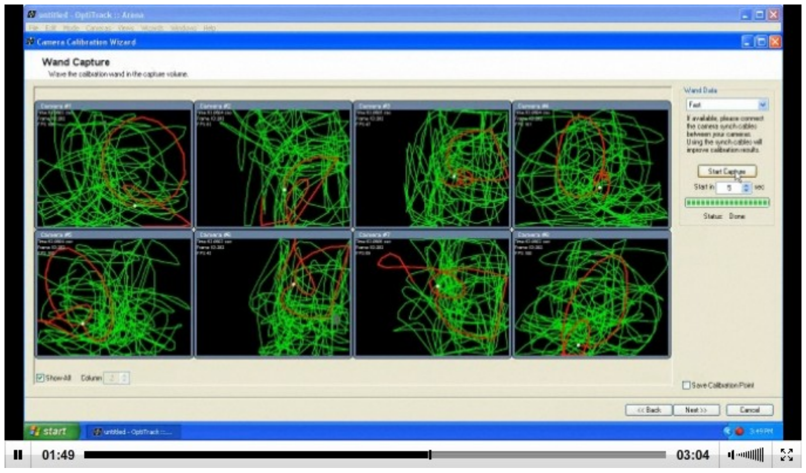
Step 2) Wanding and Solving

Click Next and Start Wanding. Stand middle in capture area and sway your ward. You can see how ward are captured by each camera like below.

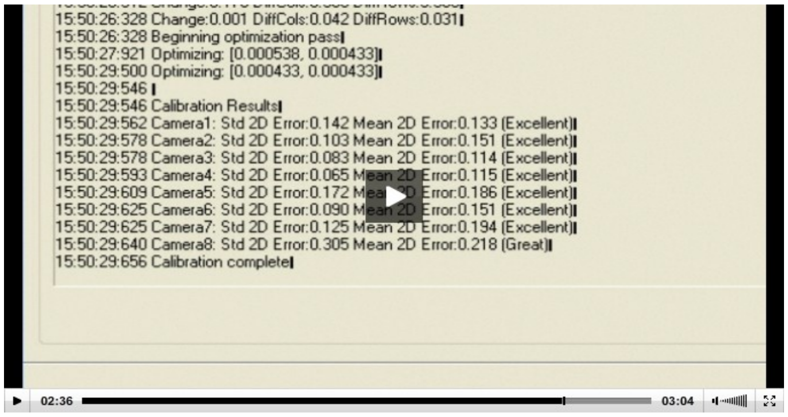

untitled - OptiTrack :: Arena
Camera Calibration Wizard
Wand Capture
Wave the calibration wand in the capture volume.
Stop Capture
Status: Capturing
Show All
Save Calibration Point
start
00:48
03:04

Move around capture area and sway ward all through the capture area with normal speed.

Try to make green line cover as many area as possible in all 18 camera windows.

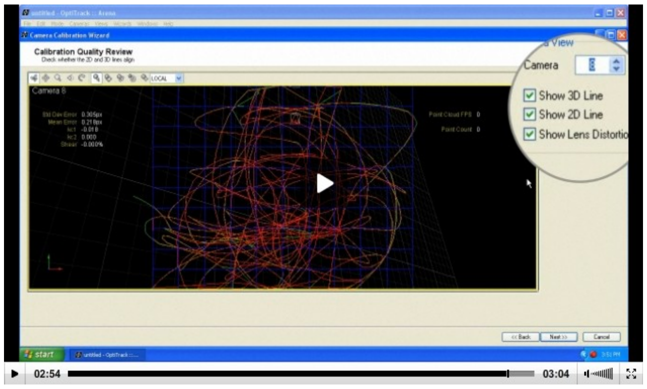


After some time, wanding process will be finished automatically. Then click next and let solving calculation works. After calculation, check all cameras have at least Excellent status. If not repeat calibration process again. Do not forget save your calibration points data in your computer.



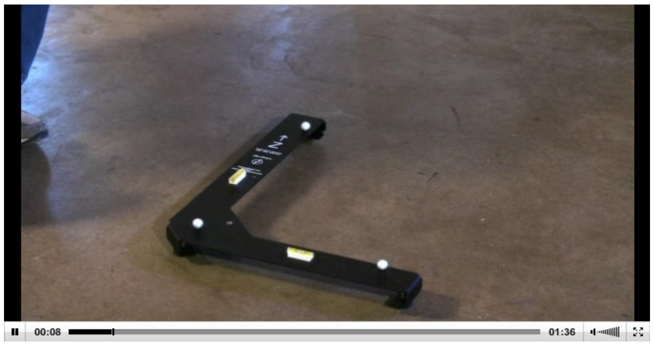
```
15:50:26:328 Change:0.001 DiffCols:0.042 DiffRows:0.031
15:50:26:328 Beginning optimization pass
15:50:27:921 Optimizing: [0.000538, 0.000433]
15:50:29:500 Optimizing: [0.000433, 0.000433]
15:50:29:546
15:50:29:546 Calibration Results
15:50:29:562 Camera1: Std 2D Error:0.142 Mean 2D Error:0.133 (Excellent)
15:50:29:578 Camera2: Std 2D Error:0.103 Mean 2D Error:0.151 (Excellent)
15:50:29:578 Camera3: Std 2D Error:0.083 Mean 2D Error:0.114 (Excellent)
15:50:29:593 Camera4: Std 2D Error:0.065 Mean 2D Error:0.115 (Excellent)
15:50:29:609 Camera5: Std 2D Error:0.172 Mean 2D Error:0.186 (Excellent)
15:50:29:625 Camera6: Std 2D Error:0.090 Mean 2D Error:0.151 (Excellent)
15:50:29:625 Camera7: Std 2D Error:0.125 Mean 2D Error:0.194 (Excellent)
15:50:29:640 Camera8: Std 2D Error:0.305 Mean 2D Error:0.218 (Great)
15:50:29:656 Calibration complete
```

After finishing successful calibration, you can see graphical calibration result in window.



Step 3) Setting the Ground Plane

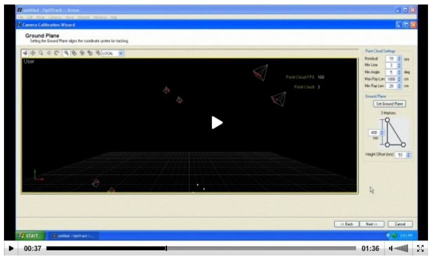
This process make us choose coordinate in capture area. Picture below shows a device which is used for setting ground coordinate.



Place this plate on origin point in capture area. If you are continuing from wanding and solving process in previous section, you just follow process which window provide to reach this Ground Plate process. *If you are not doing this process continuously from last section, you should start arena first. Then you should initially include previous calibration result (by wanding and solving). Easy way to include data is click Open Data in File toolbar. If you select your previous calibration points data, you can import previous wanding and solving information in current window.*

Next, select calibration in Wizard toolbar. Then, select Set Ground Plate Only in calibration type. After then, you should block visible markers (At this time, there should not be ground plate in capture area). If you click next, you can reach Ground Plate process. Now, let ground plate be positioned in origin point in capture area.

If you followed above steps successfully, you can see 3 markers of Ground Plate in Ground Plate window.

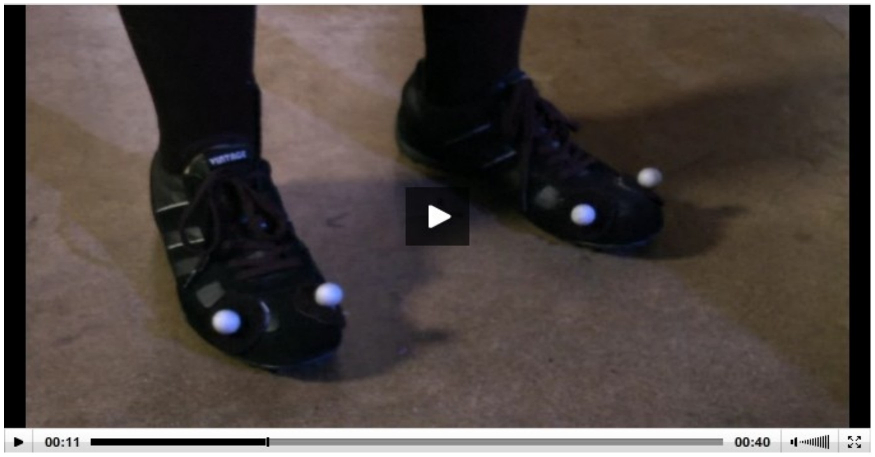


Click and drag 3 points, then click Set ground plate button on right toolbar. Of course, you should check all typed information are correct for your configuration. Follow window indicates and saves calibration result in your computer. Whenever you capture some targets in Arena Motion Capture System, you need to define coordinate of capture volume. You can define axis just importing this calibration result data. Easy way to use previous data is clicking Open Data in file toolbar.

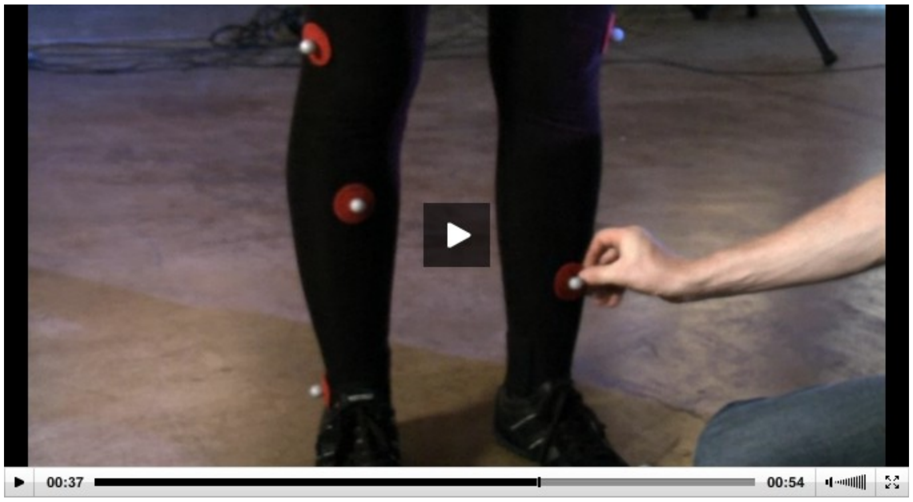
2. Attaching Markers in Human Body

This Process is necessary only when you capture a human body. If your goal is to capture a robot(non humanoids) or other kinds of objects, please refer Section 5. Rigid Body Wizards.

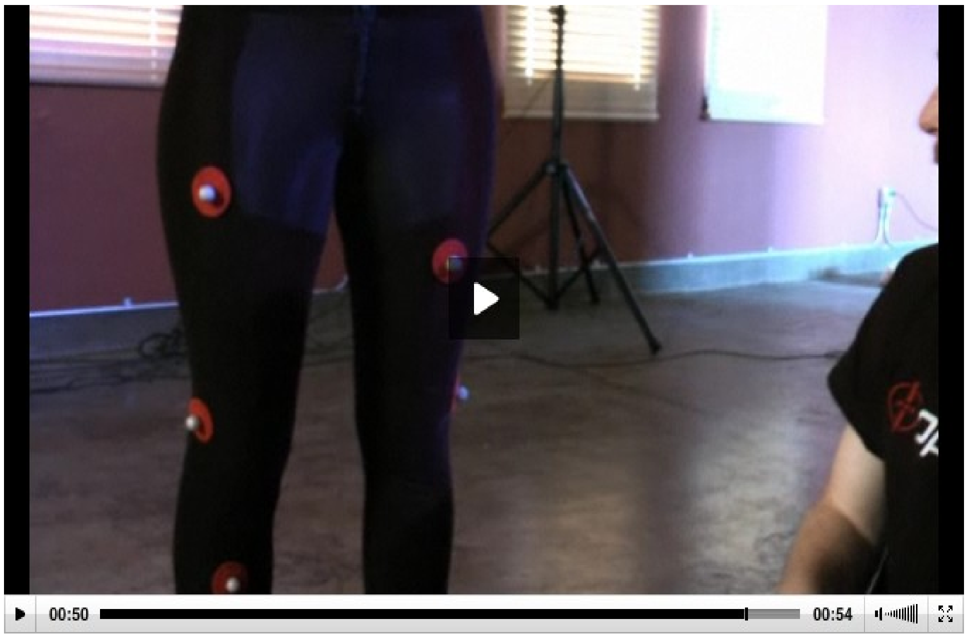
1) Foot : 2 markers per each foot



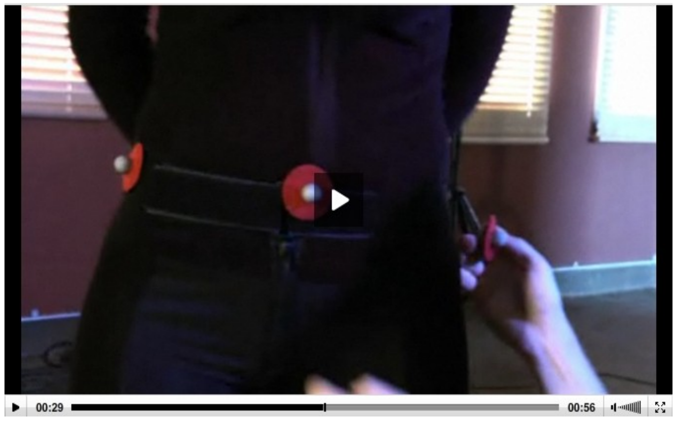
2) Lower Leg: 3 markers per each leg. Middle marker should be asymmetrically positioned between 2 legs.

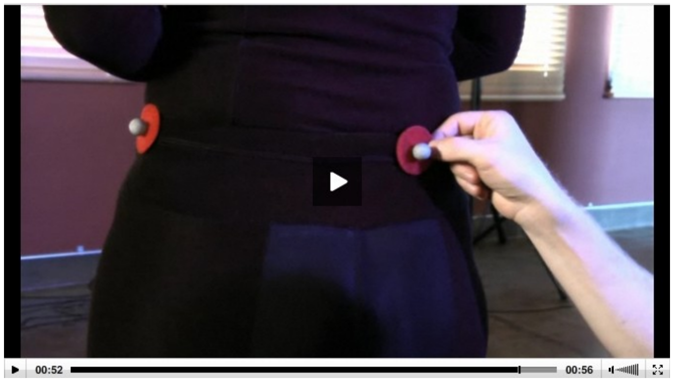


3) Upper Leg : One per each upper leg. Asymmetrically positioned between 2 legs.

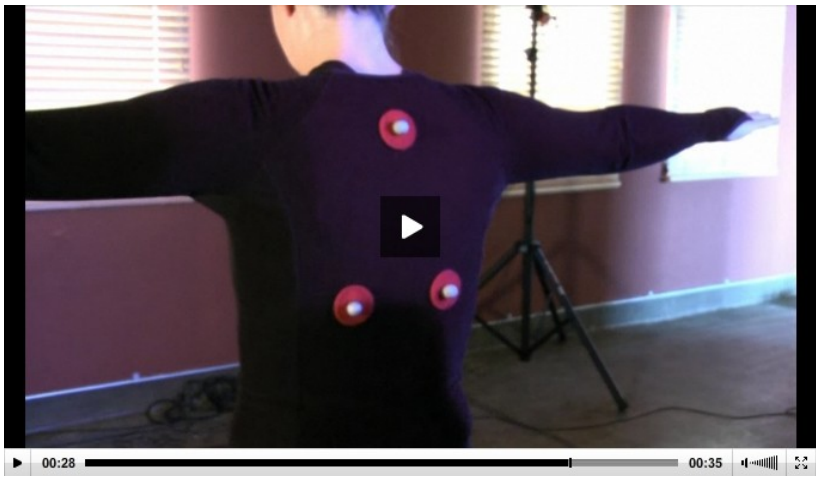


4) Hip : 4 markers. 1 for Front, 2 for Side and 1 for Back. In case of a marker in back position can be attached to either left or right position from center. If you want to track 2 people in Capture system, each person should have different position. Left for 1 person and right for another one.

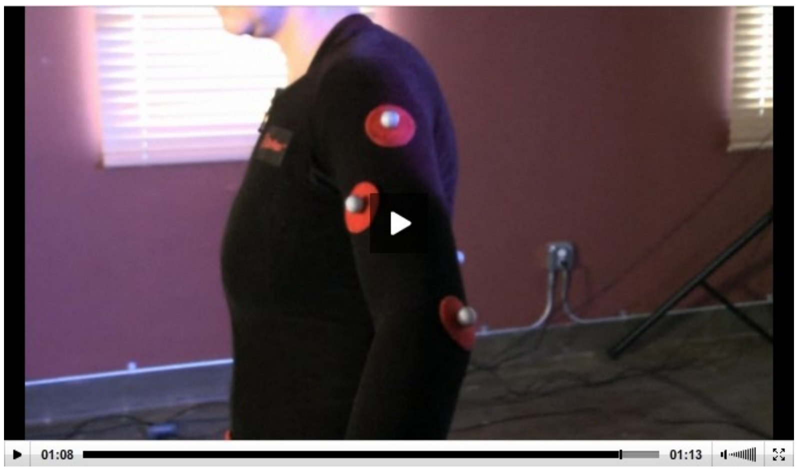




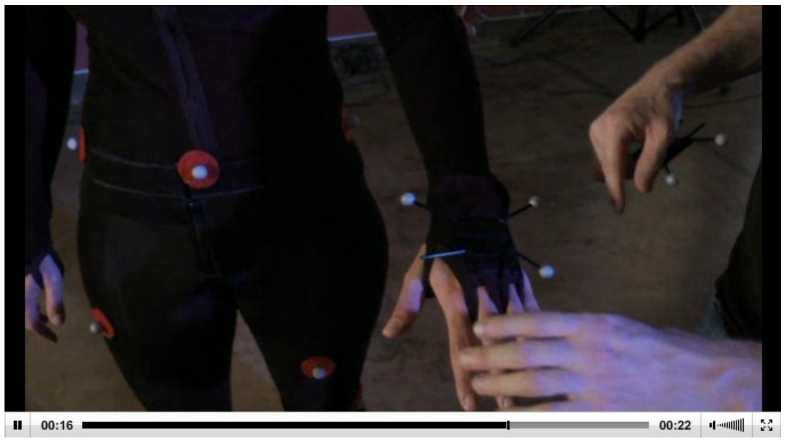
5) Back : 3 markers for back.



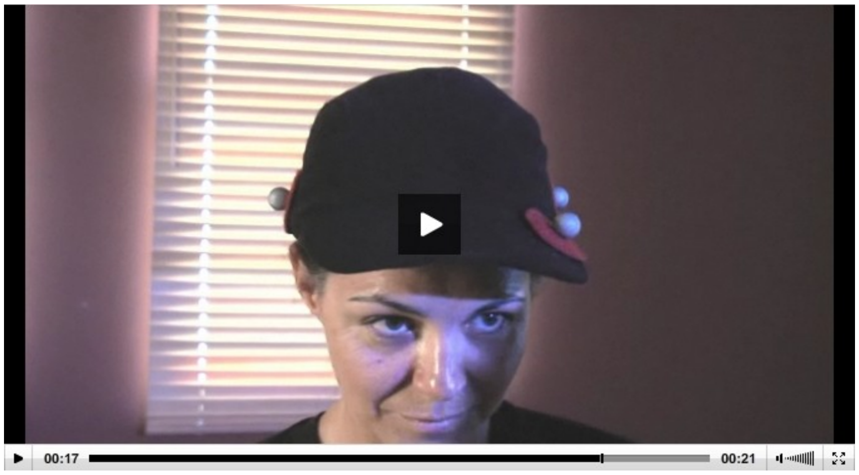
6) Arm : 3 per each arm. Do not make 2 side of triangle have same length. If middle point is closer to top point in left arm, try to make middle point closer to bottom point in right arm.



7) Hand: 1 rigid body which has 3 markers per each hand



8) Head : 3 Markers. 2For Side and 1 for Front. Front marker should left side of cap.

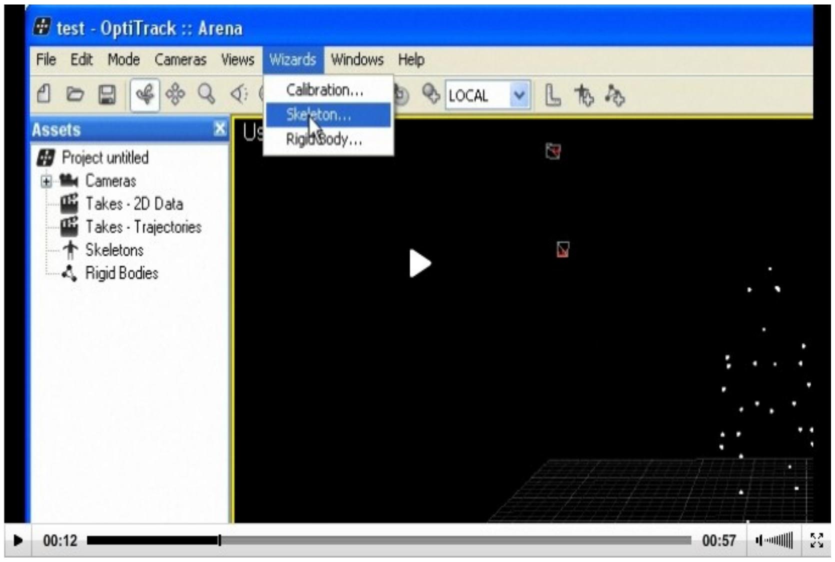


For more details, please watch tutorial in Arena web-page (http://www.naturalpoint.com/optitrack/products/arena/tutorials.html).

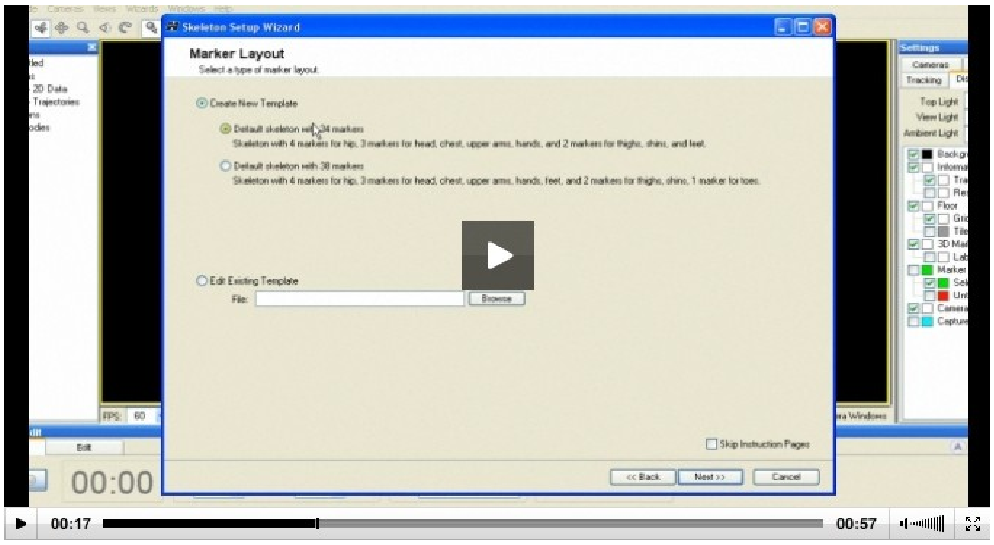
3. The Skeleton Wizards

This Process is necessary only when you capture a human body. If your goal is to capture a robot(non humanoids) or other kinds of objects, please refer Section 5. Rigid Body Wizards.

Step1) Select Skeleton wizard in Calibration toolbar in top of window.



Step 2) Select Default Skeleton of 34 markers (If you want to use Skeleton of 38 markers, you should attach 2 more markers to each foot. 1 for front, another for back).



Marker Layout

Select a type of marker layout.

(✓) Create New Template

(✓) Default skeleton with 34 markers

Skeleton with 4 markers for hip, 3 markers for head, chest, upper arms, hands, and 2 markers for thighs, shins, and feet.

( ) Default skeleton with 38 markers

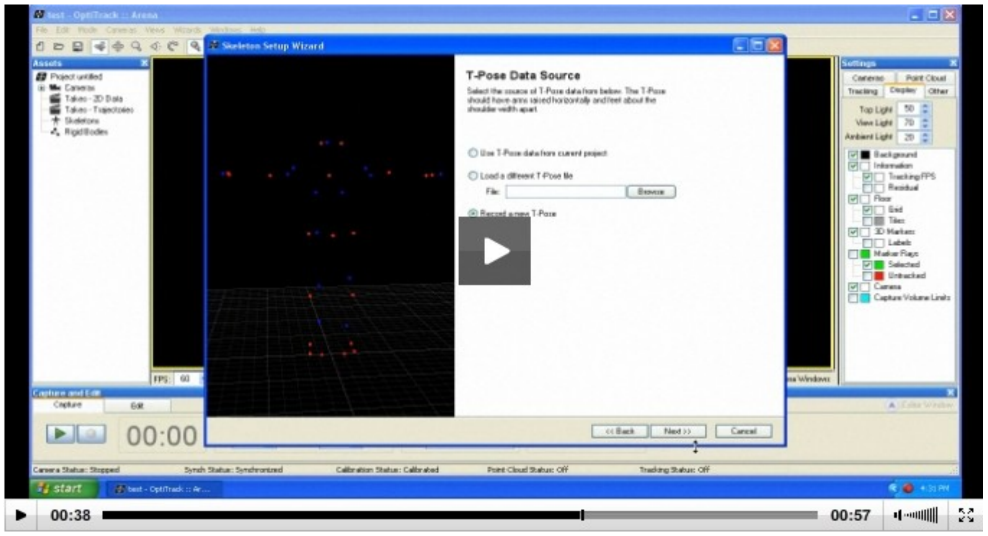
Skeleton with 4 markers for hip, 3 markers for head, chest, upper arms, hands, feet, and 2 markers for thighs, shins, 1 marker for toes.

( ) Edit Existing Template

File:

Step 3) Select skip instruction if you do not want to see. Then, click Next.

Step 4) Select a Proceed with New T-pose.



T-Pose Data Source

Select the source of T-Pose data from below. The T-Pose should have arms raised horizontally and feet about the shoulder width apart.

( ) Use T-Pose data from current project

( ) Load a different T-Pose file

File:

(✓) Record a new T-Pose

Camera Status: Stopped

Synch Status: Synchronized

Calibration Status: Calibrated

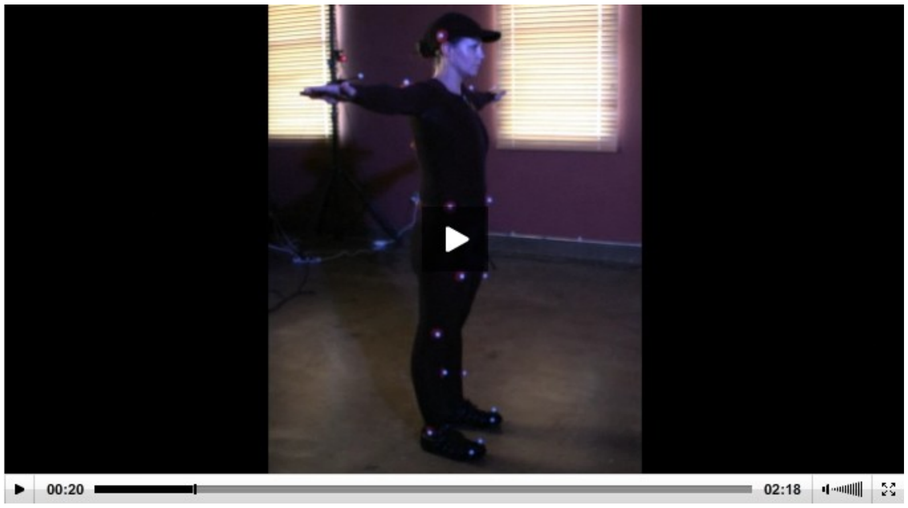
Point Cloud Status: Off

Tracking Status: Off

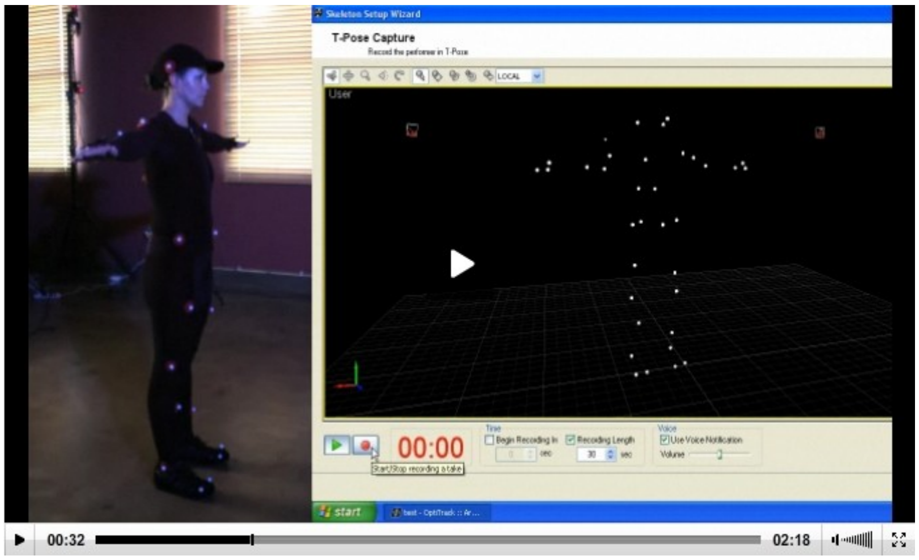
4. T-Pose and Skeleton Calibration

This Process is necessary only when you capture a human body. If your goal is to capture a robot(non humanoids) or other kinds of objects, please refer Section 5. Rigid Body Wizards.

Step 1) Make one person stand in capture are and make T-Pose

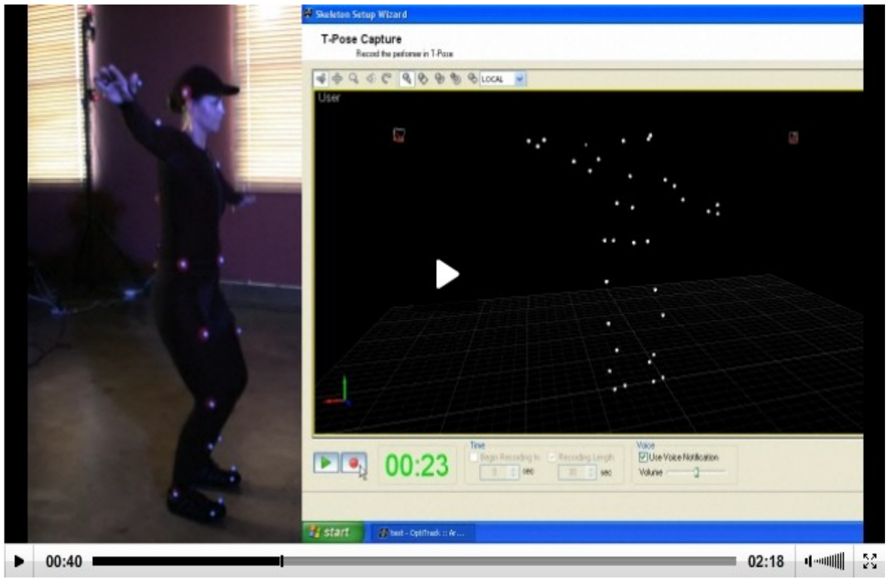


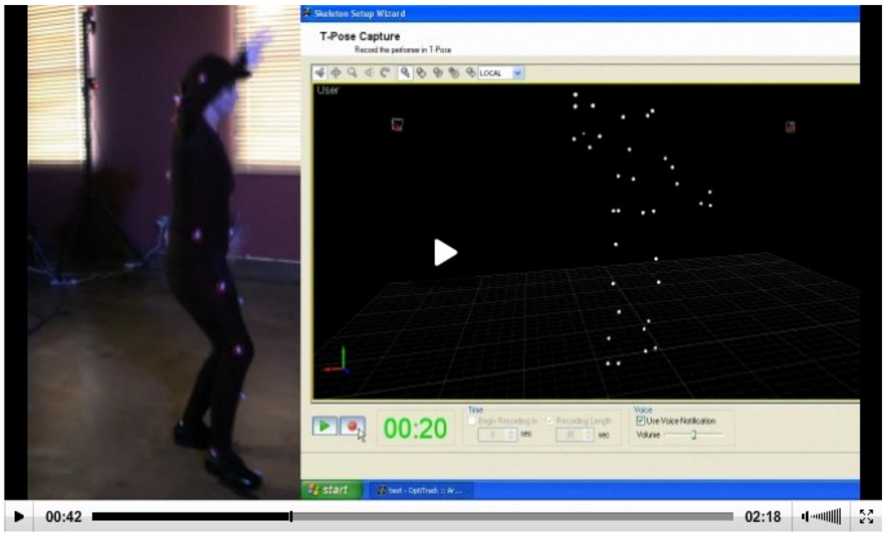
Step 2) Push Play Button and Record Button

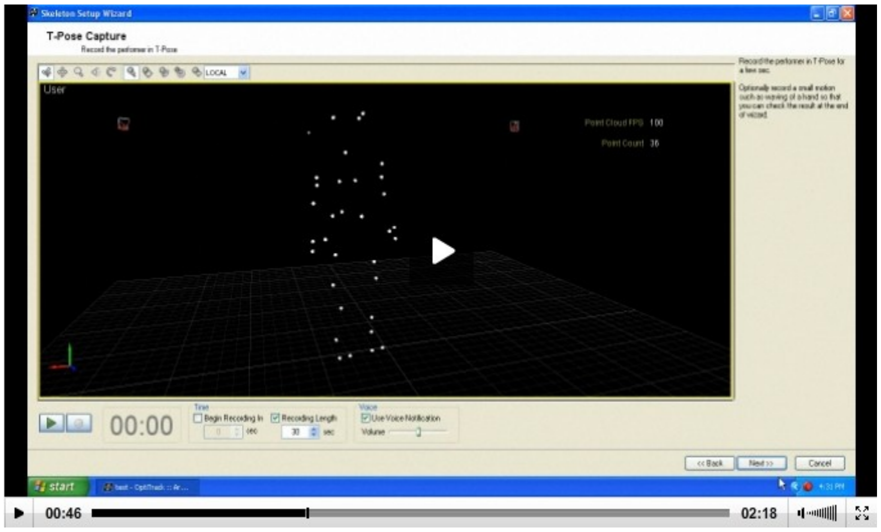


Initially, this will make MOCAP system record a person in capture area for 20 seconds.

Step 3) Make a person inside capture area make a basic movement. After 20 seconds, recording will be finished. Click Next.






Record the performer in T-Pose for a few sec.

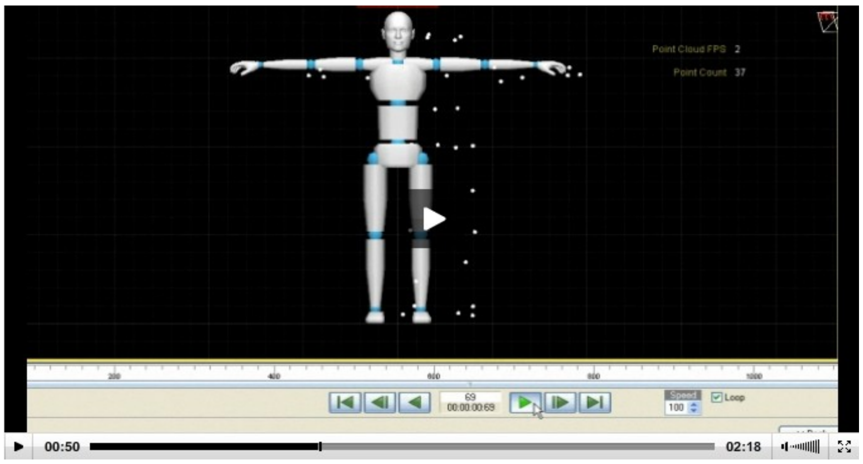
Optionally record a small motion such as waving of a hand so that you can check the result at the end of wizard.

Point Cloud FPS 100

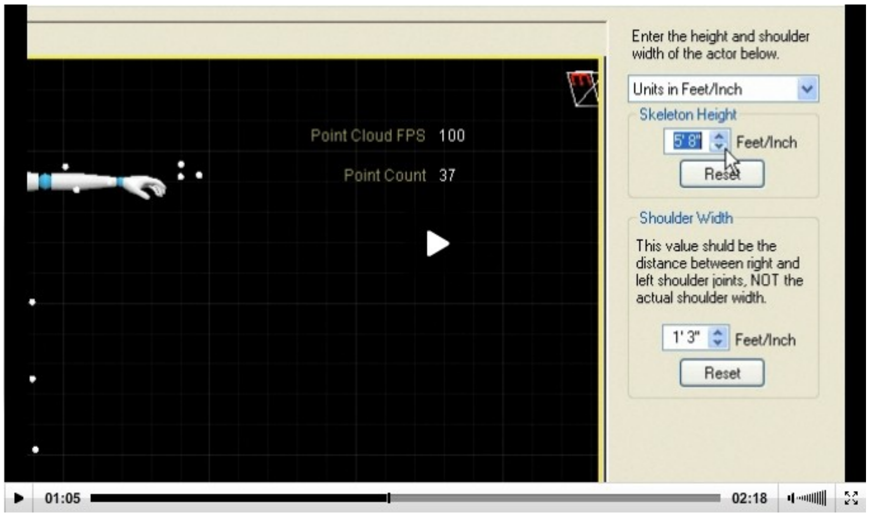
Point Count 36

Begin Recording In [ ] · Recording Length [✓] 30 sec · Use Voice Notification [✓]

Step 4) Push Play Button and Stop when T-pose look good



Step 5) Type height and shoulder width


Enter the height and shoulder width of the actor below.

Units in Feet/Inch

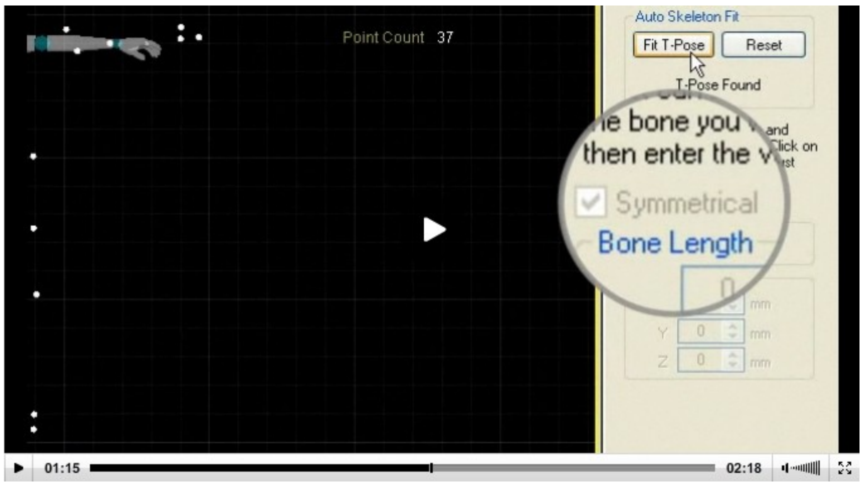
Skeleton Height: 5' 8" Feet/Inch

Shoulder Width

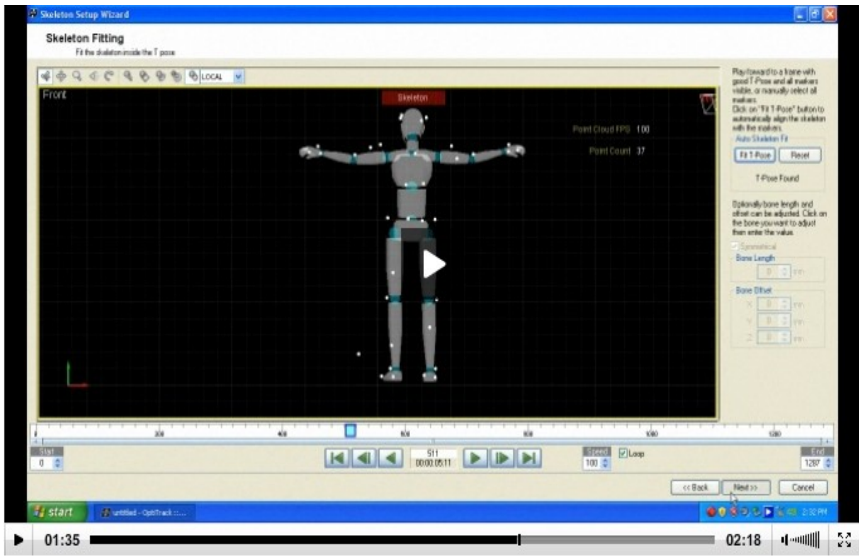
This value shuld be the distance between right and left shoulder joints, NOT the actual shoulder width.

1' 3" Feet/Inch

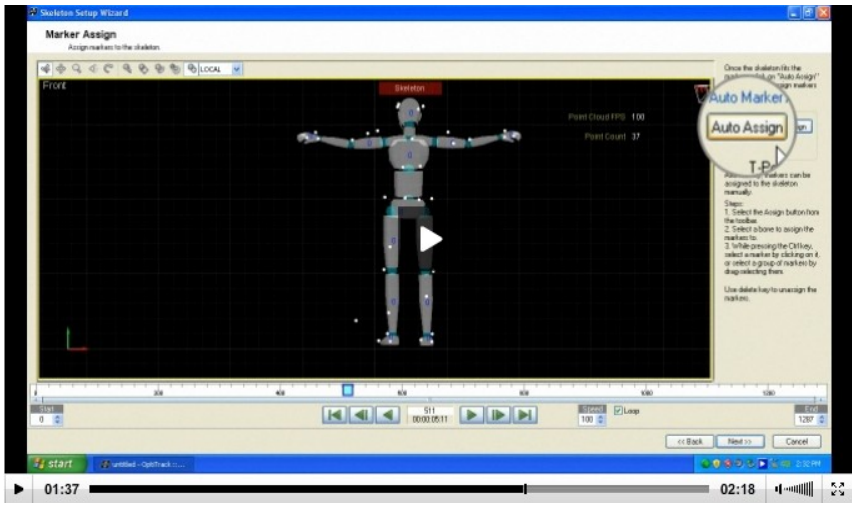
Step 6) Select Fit-Tpose

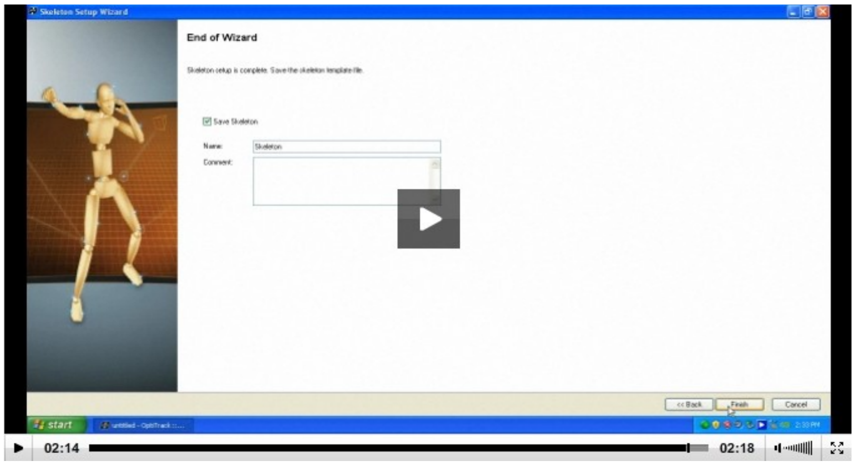


Step 7) Select Next



Step 8) Click Auto-Assign and save Skeleton file

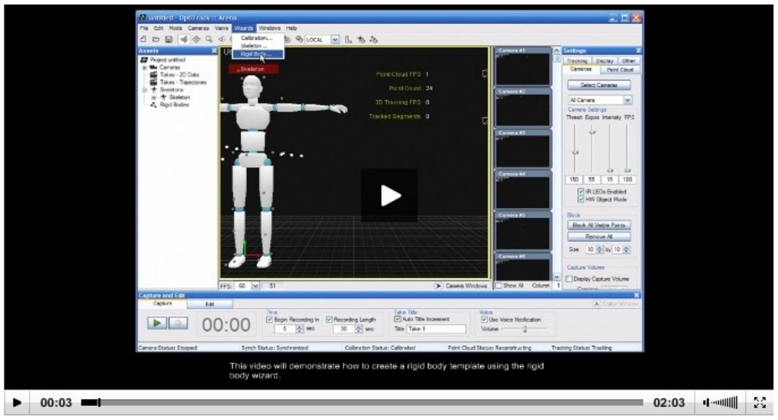




5. Rigid Body Wizards

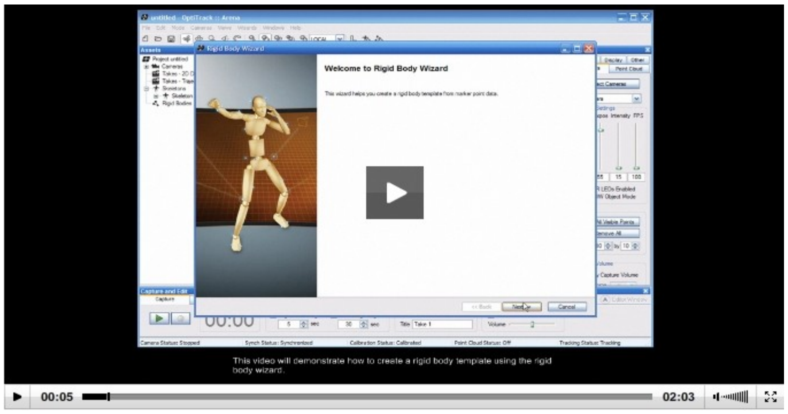
This Process is necessary only when you capture a robot(non humanoids) or other kinds of objects. If your goal is to capture a human body, please refer Section 3 and 4.

Step 1) Select Rigid Body toolbar in Calibration toolbar



This video will demonstrate how to create a rigid body template using the rigid body wizard.

Step 2) Click Next

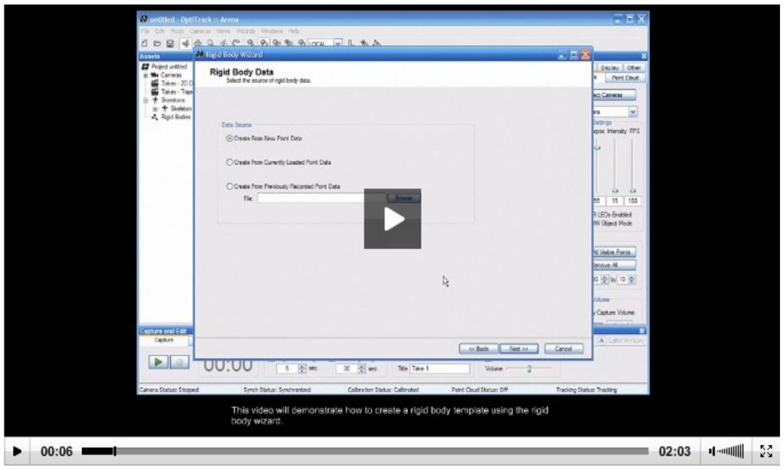


Welcome to Rigid Body Wizard

This wizard helps you create a rigid body template from marker point data.

This video will demonstrate how to create a rigid body template using the rigid body wizard.

Step 3) Create New Rigid Body and Click Next



Rigid Body Data

Select the source of rigid body data.

Data Source

Create From New Point Data

Create From Currently Loaded Point Data

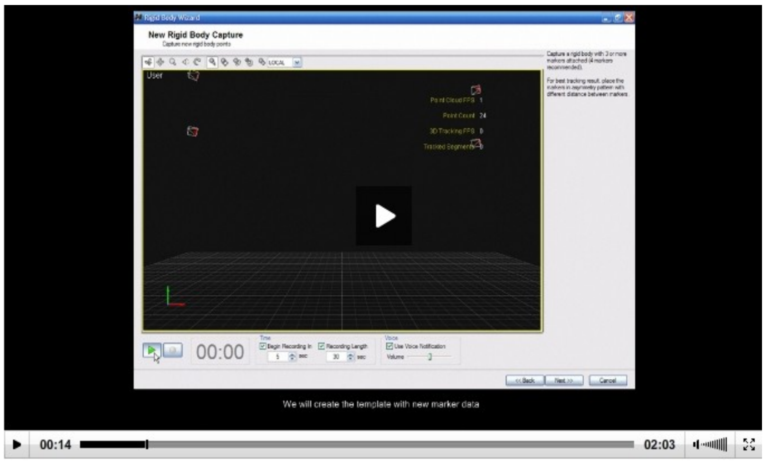
Create From Previously Recorded Point Data

File:

This video will demonstrate how to create a rigid body template using the rigid body wizard.

Step 4) Attach 3 markers on target objects(such as mobile robot) to be tracked. 3 markers will be captured as one rigid body. In Mocap system, we can get position and rotation data of each rigid body. If your object consists of several rigid body, you should attach 3 markers per each rigid body. For example, if your target is mobile robot which consists of 3 rigid body, you need totally 9 markers. Each set of 3 markers should be close enough and form triangle. Do not forget that triangle of each rigid body should have different shape. If not, system will be confused to distinguish rigid bodies. You also can use 4 markers to define one rigid body if you have enough number of spare markers

Step 5) Push Play button to activate cameras

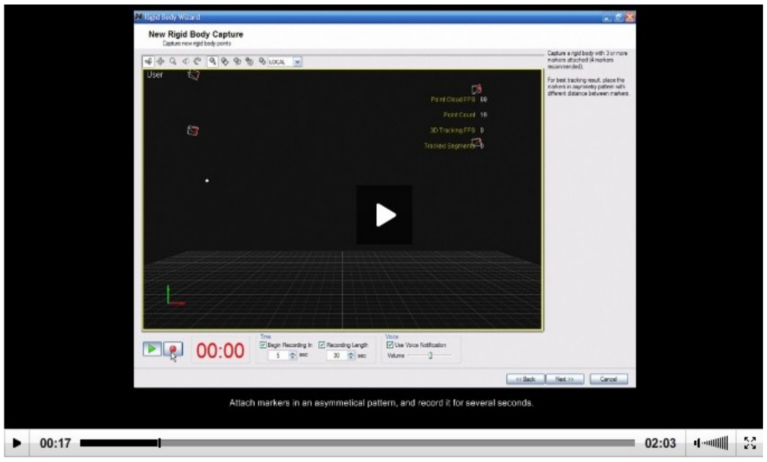


Capture a rigid body with 3 or more markers attached (4 markers recommended).

For best tracking result, place the markers in asymmetry pattern with different distance between markers.

We will create the template with new marker data

Step 6) Push Record Button

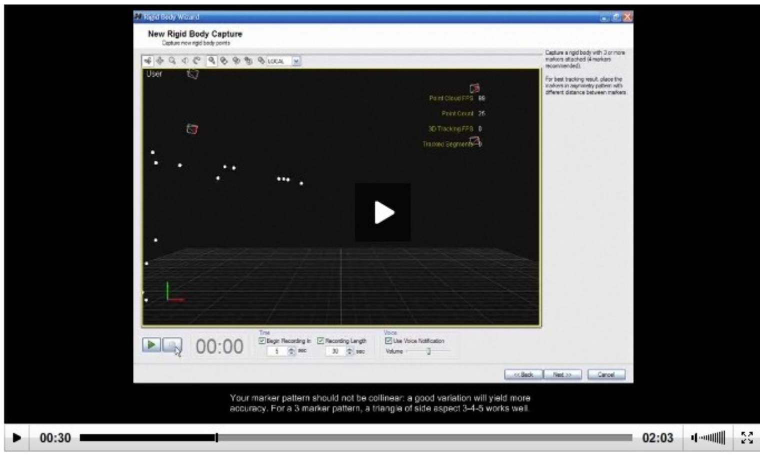


Capture a rigid body with 3 or more markers attached (4 markers recommended).

For best tracking result, place the markers in asymmetry pattern with different distance between markers.

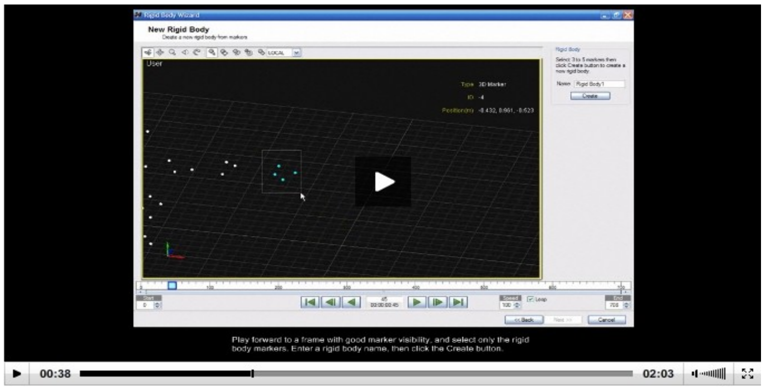
Attach markers in an asymmetrical pattern, and record it for several seconds.

Step 7) After finishing recording of movement (or experiment), Push record button again to finish recording.



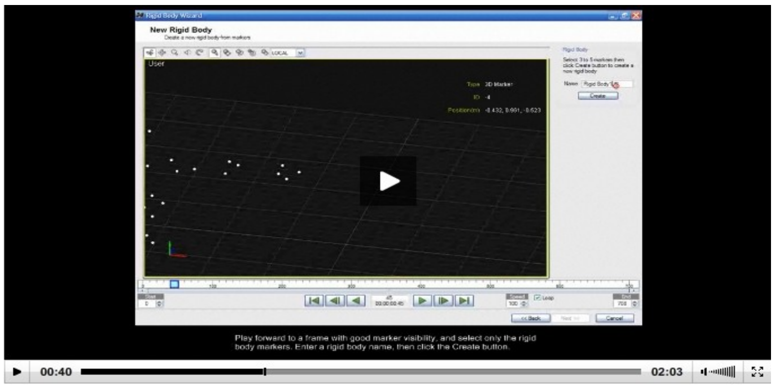
Your marker pattern should not be collinear: a good variation will yield more accuracy. For a 3 marker pattern, a triangle of side aspect 3-4-5 works well.

Step 8) Play back to start time of recorded file. Then, click and drag markers to define a rigid body. If your target consists of 3 rigid bodies, you can define one rigid body first in this step. You should repeat this section to define remaining rigid bodies. In sum, you should click and drag markers which are of one rigid body only in this step. You should not click and drag other markers which define other rigid bodies. You can do deal with them by repeating this section.



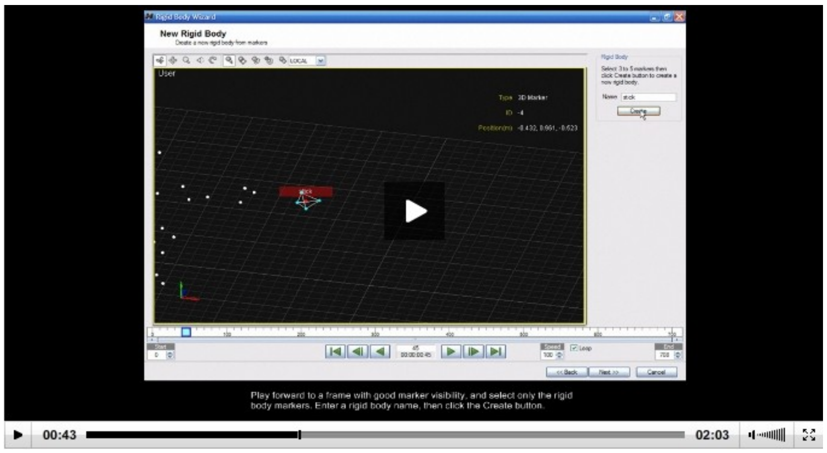
Play forward to a frame with good marker visibility, and select only the rigid body markers. Enter a rigid body name, then click the Create button.

Step 9) Type name of this rigid body



Play forward to a frame with good marker visibility, and select only the rigid body markers. Enter a rigid body name, then click the Create button.

Step 10) Click Create button

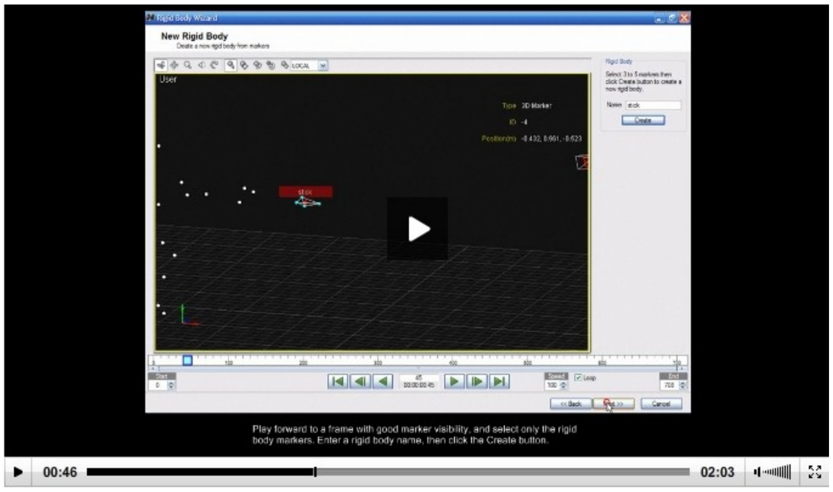


Select 3 to 5 markers then click Create button to create a new rigid body.

Type 3D Marker
ID -4
Position(m) -0.432, 0.961, -0.623

Play forward to a frame with good marker visibility, and select only the rigid body markers. Enter a rigid body name, then click the Create button.

Step 11) Click Next

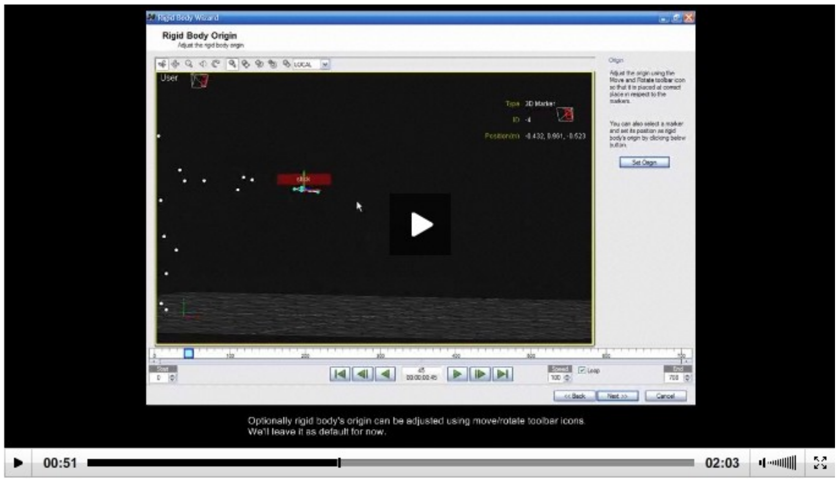


Select 3 to 5 markers then click Create button to create a new rigid body.

Type 3D Marker
ID -4
Position(m) -0.432, 0.961, -0.623

Play forward to a frame with good marker visibility, and select only the rigid body markers. Enter a rigid body name, then click the Create button.

Step 12) In this step, you can select origin point. Click one marker which you want to set as origin and click Set Origin button



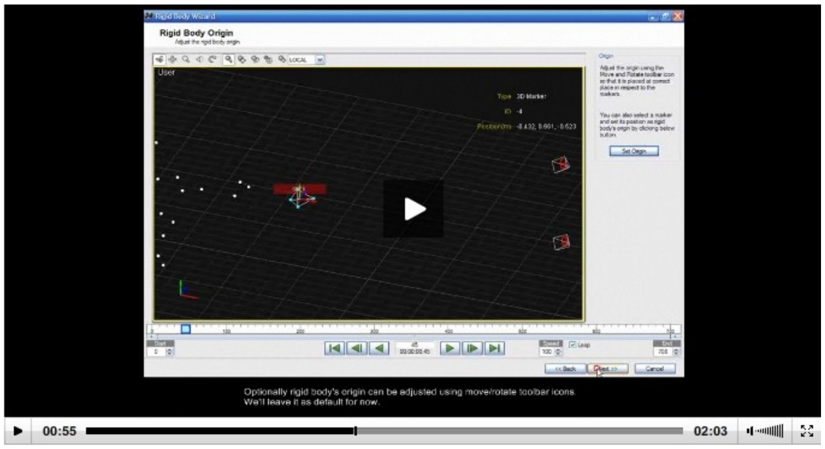
Adjust the origin using the Move and Rotate toolbar icon so that it is placed at correct place in respect to the markers.

You can also select a marker and set its position as rigid body's origin by clicking below button.

Type 3D Marker
ID -4
Position(m) -0.432, 0.961, -0.623

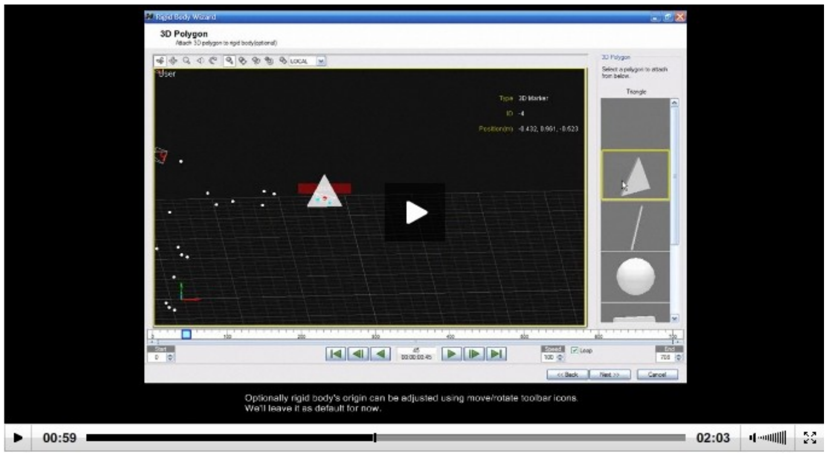
Optionally rigid body's origin can be adjusted using move/rotate toolbar icons. We'll leave it as default for now.

Step 13) Click Next



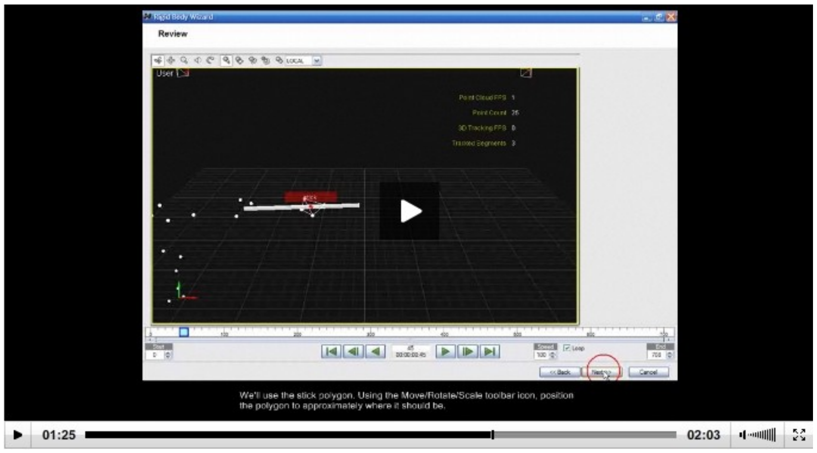
Optionally rigid body's origin can be adjusted using move/rotate toolbar icons.
We'll leave it as default for now.

Step 14) You can decide shape of rigid body



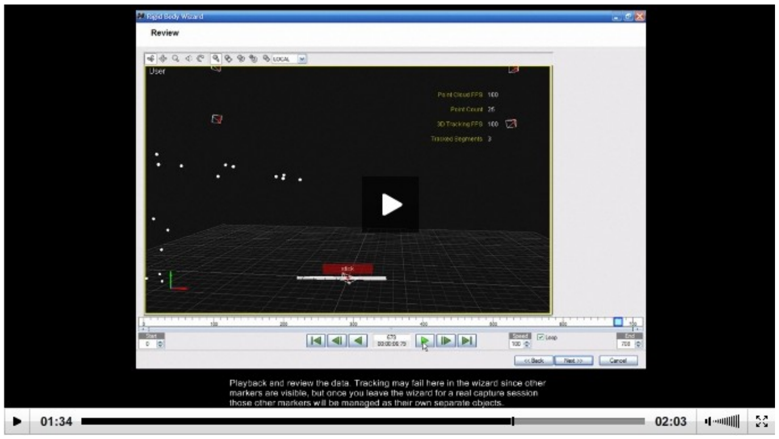
Optionally rigid body's origin can be adjusted using move/rotate toolbar icons.
We'll leave it as default for now.

Step 15) Push Next button



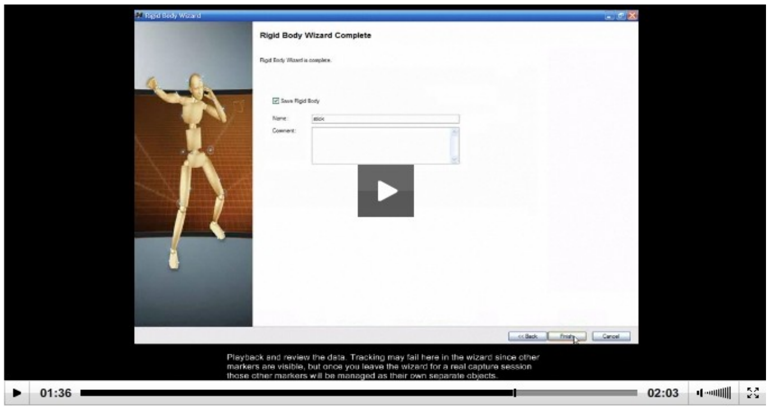
We'll use the stick polygon. Using the Move/Rotate/Scale toolbar icon, position the polygon to approximately where it should be.

Step 16) If you playback to start time, you can see markers are handled as one rigid body in recorded file.



Playback and review the data. Tracking may fail here in the wizard since other markers are visible, but once you leave the wizard for a real capture session those other markers will be managed as their own separate objects.

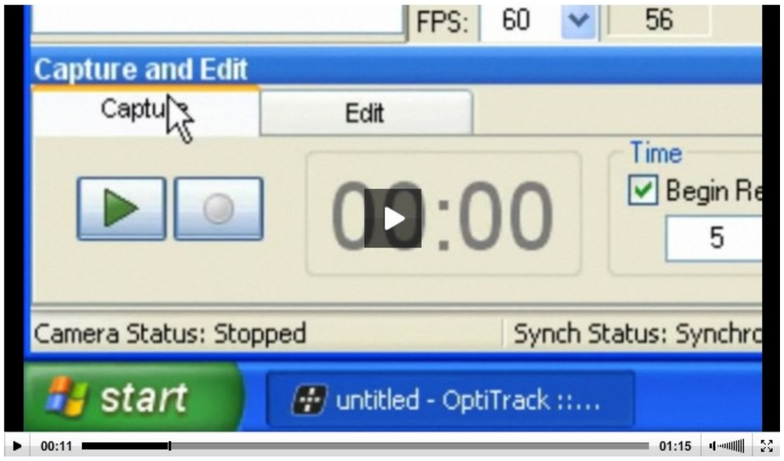
Step 17) Save rigid body file



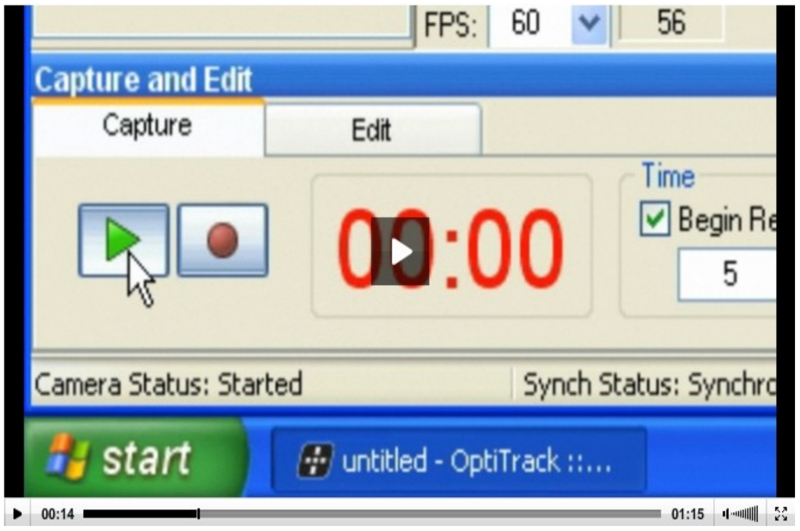
Playback and review the data. Tracking may fail here in the wizard since other markers are visible, but once you leave the wizard for a real capture session those other markers will be managed as their own separate objects.

6. Capture and Trajectorizing

Step 1) Go to Capture session in bottom of window



Step 2) Push Play button to activate cameras



FPS: 60 56

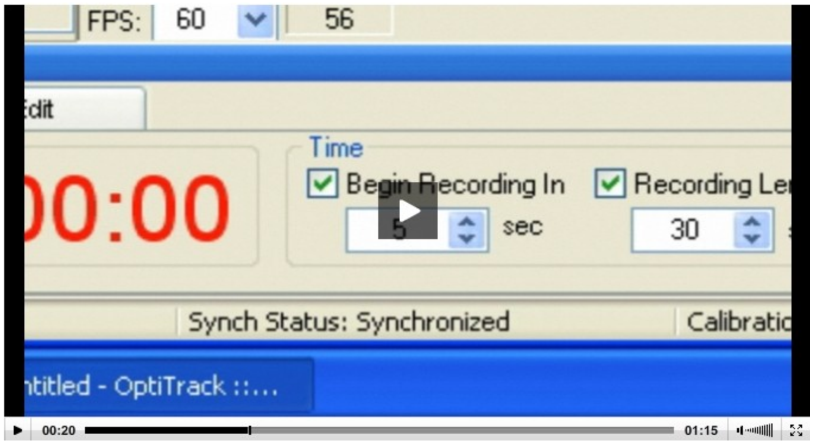
00:00

Begin Re 5

Camera Status: Started

Synch Status: Synchro

Step 3) Type Begin time and recording time



FPS: 60 56

00:00

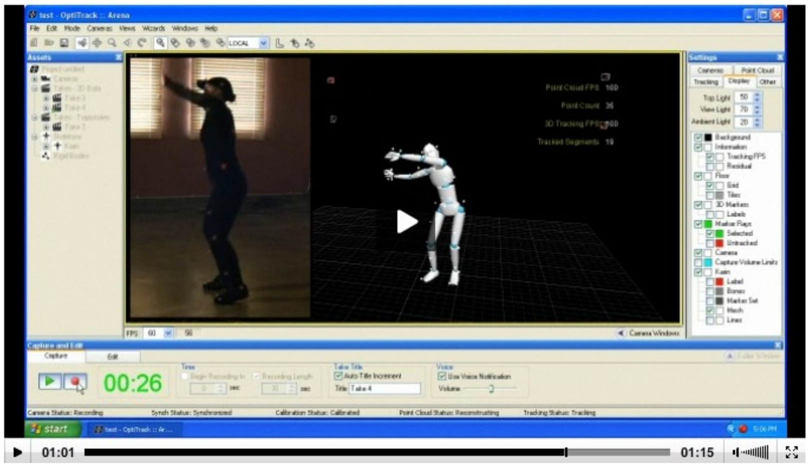
Begin Recording In 5 sec

Recording Le 30

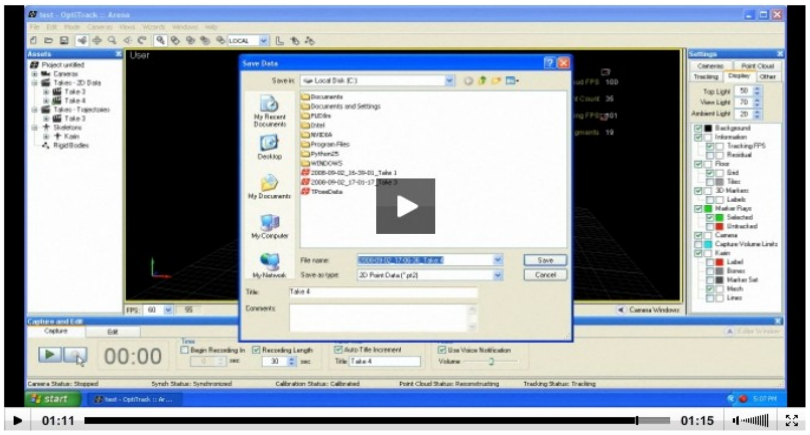
Synch Status: Synchronized

Calibratio

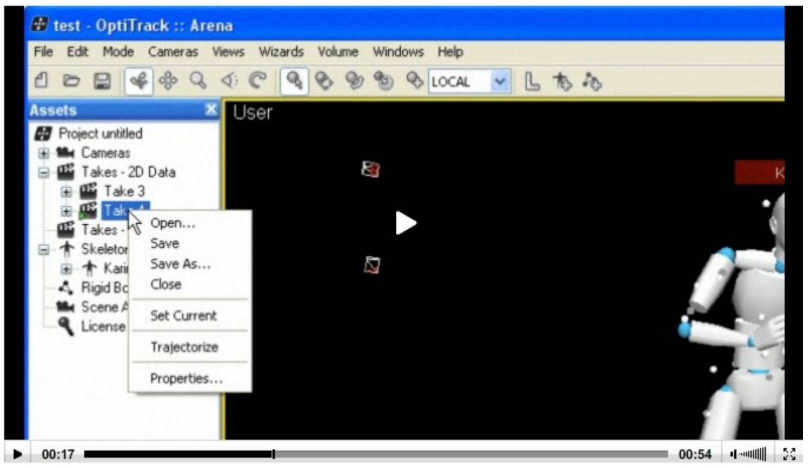
Step 4) Then, push record button. Let people or objects make a movement to be recorded in capture area.

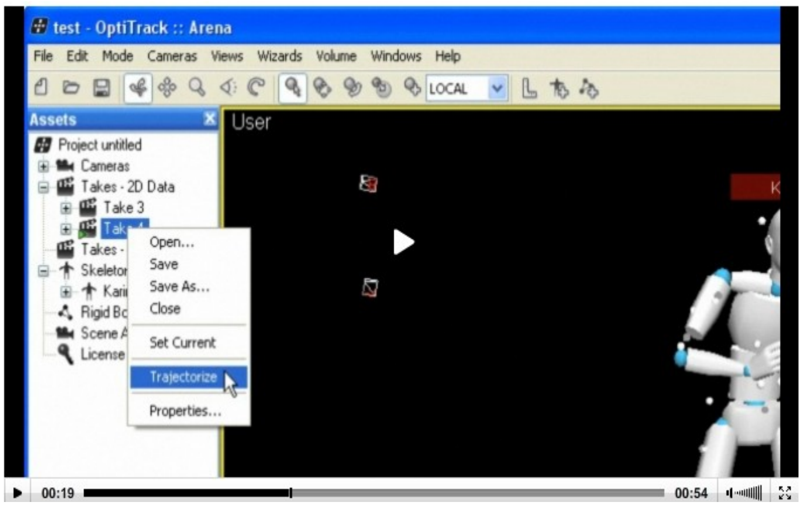


Step 5) Click Record button again to finish recording. Or recording will be terminated automatically after recording time. Save Take file (2D Point data).

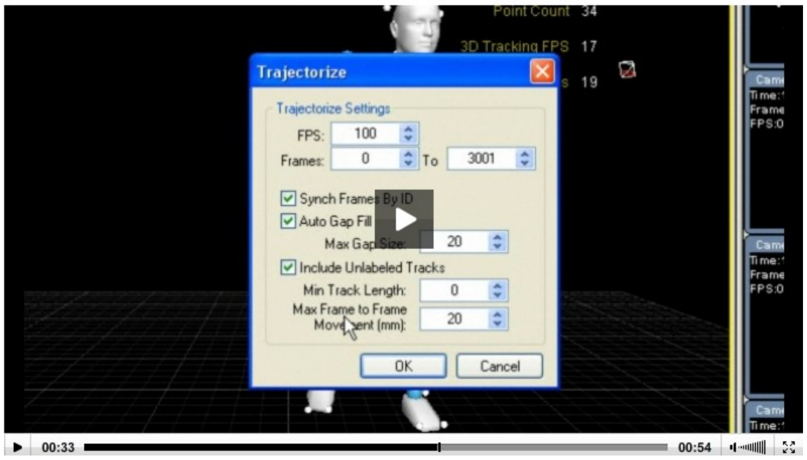


Step 6) To export this data, you need to trajectorize this Take file (2D Data). Goto left toolbar and right click Take file (2D Data) which you want to convert. Then select Trajectorize.





Step 7) Input necessary data and click OK


Trajectorize

Trajectorize Settings

FPS: 100

Frames: 0 To 3001

☑ Synch Frames By ID

☑ Auto Gap Fill

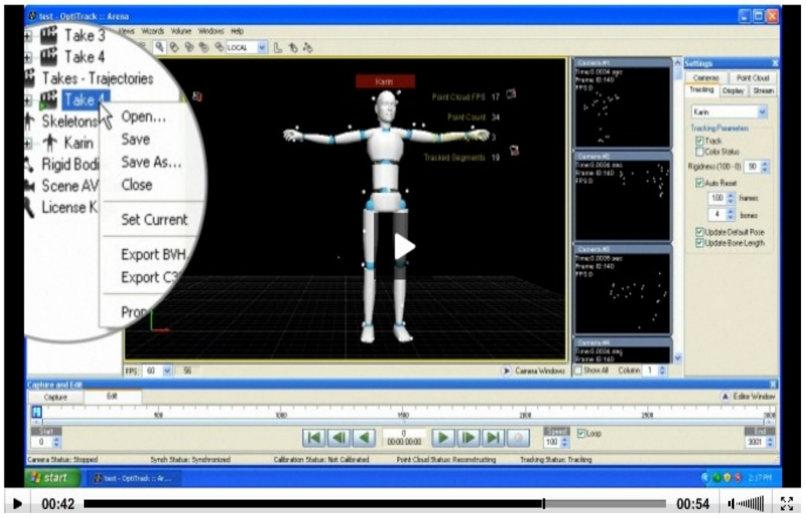
Max Gap Size: 20

☑ Include Unlabeled Tracks

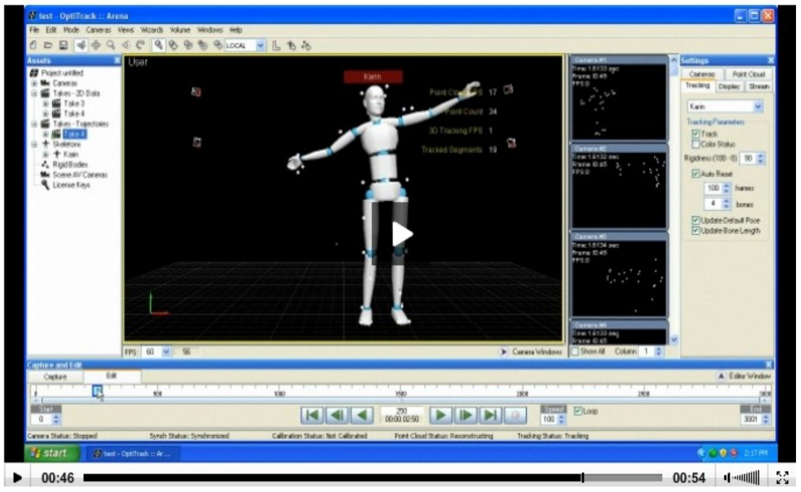
Min Track Length: 0

Max Frame to Frame Movement (mm): 20

Step 8) Now you have new Take (trajectory) file of this Take (2D Data) file. Right click the new Take (Trajectory) file and click Set Current.



Step 9) Now you are ready to export this MOCAP data file



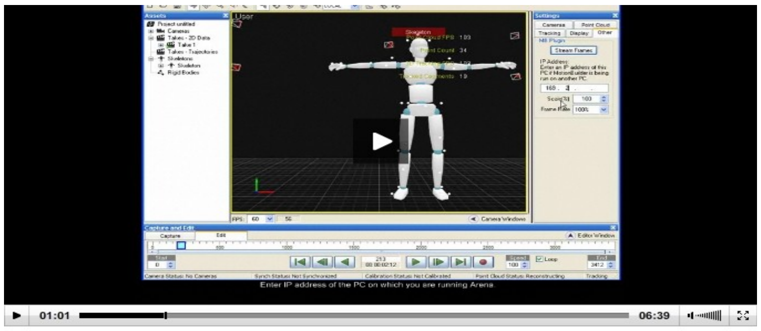
Important tips: When you use old Take (either 2D Data or Trajectory file), and want to see how skeleton move, you should import corresponding skeleton file. In left toolbar, right-click skeleton and choose proper pre-defined skeleton data. Other way is open skeleton file 1st, then go to file toolbar. And select open-data and choose old Take (either 2D Data or Trajectory file) which you want to see.

7. Editing Trajectory

You can fix errors in your captured trajectory. Since this will not be used frequently by users, we will skip this process in this tutorial. If you want to know more details, you can visit Arena web-site (http://www.naturalpoint.com/optitrack/products/arena/tutorials.html).

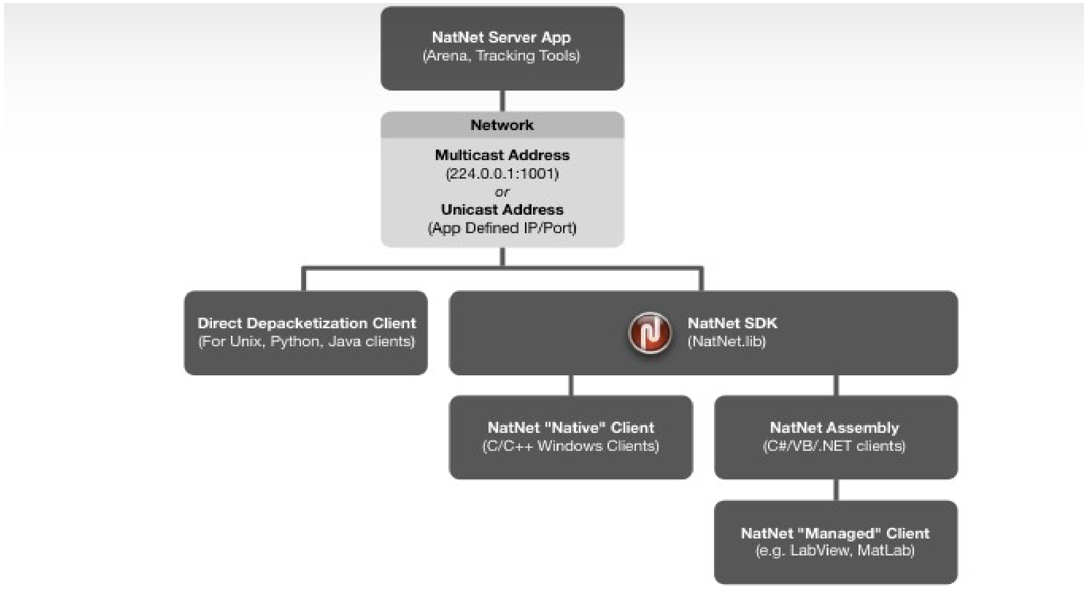
8. Data Streaming and Get Motion data.

If you have either Takes-2D Data (recorded data of skeleton or rigid body movement) or Takes-Trajectories (imported trajectory file from Takes-2D Data), you can stream motion capture data through network. If you click Stream toolbar in right of window, you can see IP Address section which you can write. After inputting correct address, you can choose data which you want to stream. Usually we choose 3D Marker Positions and Rigid Body Data. Then push Start Streaming Frames button.



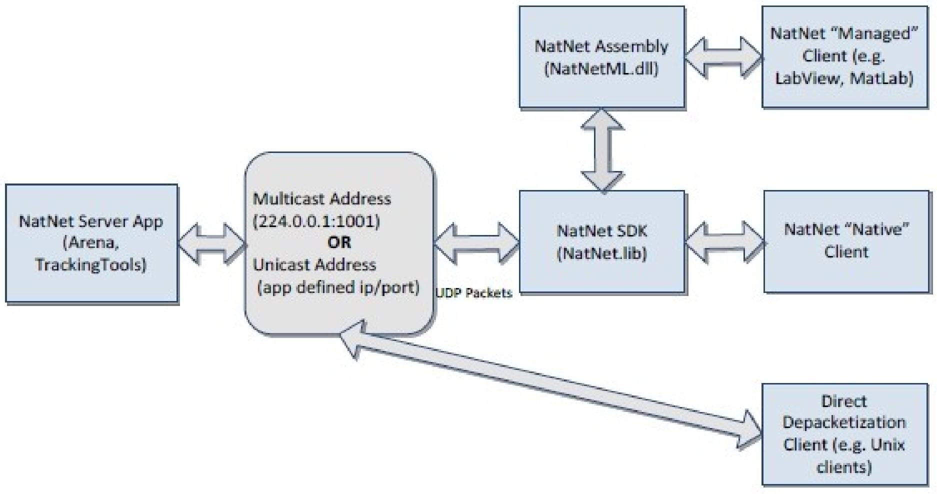
Next, you should know how to receive these streaming data through network.
For this, you should download NATNET SDK package in Arena web-site (http://www.naturalpoint.com/optitrack/downloads/arena.html).

NatNet's client/server architecture allows client applications to run on the same system as the tracking software (ARENA/Tracking Tools), on separate system(s), or both. The SDK integrates seamlessly with standard APIs (C/C++/.NET), tools (Microsoft Visual Studio) and protocols (UDP/Unicast/Multicast). Using the NatNet SDK, developers can quickly integrate OptiTrack motion tracking data into new and existing applications, including custom plugins to third-party applications and engines for real-time streaming.



The version which we currently use is 2.2.

Figure 1 – NatNet Component Overview


NatNet Assembly (NatNetML.dll)
NatNet "Managed" Client (e.g. LabView, MatLab)
NatNet Server App (Arena, TrackingTools)
Multicast Address (224.0.0.1:1001) OR Unicast Address (app defined ip/port)
UDP Packets
NatNet SDK (NatNet.lib)
NatNet "Native" Client
Direct Depacketization Client (e.g. Unix clients)

The NatNet SDK consists of :

1· NatNet Library Native C++ networking library (headers, static library (.lib) and dynamic import library (.lib/.dll))
2· NatNet Assembly Managed .NET assembly (NatNetML.dll) for use in .Net compatible clients.
3· NatNet Samples Sample projects and executables designed to be quickly integrated into your own code.

Pre-compiled versions of the NatNet samples have been provided in the \Samples\bin folder. These versions can be used to quickly test your application. Please refer to the instructions in this section for information on running specific samples.

Note! The Visual C++ runtime libraries are required to run the samples. If you encounter an error message when attempting to run the samples, especially on machines without Visual C++ installed, please install the VC runtime redistributable package located in Samples\VCRedist. If the problem peresists, pleas try rebuildingthe samples using VisualC++, or contact support.

How to run the simple CLIENT-SERVER sample :

1. Start the server:
SimpleServer.exe

2. Start the client:
SampleClient.exe [IPAddress] [OutputFilename.txt]

3. Start streaming by pressing 's' in the SimpleServer console window.
You should begin to see data streaming in the client window or to text file.

Note
1· [parameters] are optional.
2· If no IP address is specified, the client will assume the server is on the same machine (local machine).

The code samples are the quickest path towards getting NatNet data into your application. We typically recommend you :

1. Identify your application's development/interface requirements (managed, native, etc).
2. Adapt the NatNet sample code from the corresponding NatNet sample application in the samples folder into your application.
3. Use the API reference for additional information.
The Visual Studio solution file \Samples\NatNetSamples.sln will open and build all of the NatNet sample projects.

If you are creating an application from scratch, please refer to the following sections for application specific requirements.

BUILDING A NATIVE CLIENT TO RECEIVE NATNET DATA

Steps for building a NatNet client application/library to receive data from a NatNet server application such as Arena or TrackingTools:
1. Adapt the SampleClient sample (SampleClient.cpp) to your application's code.
2. Include NatNetClient.h, NatNetHelper.h, and NatNetTypes.h
3. Link to NatNetLib.lib (dynamic) OR NatNetLibStatic.lib (static)
4. [OPTIONAL] If linking dynamically, define NATNETLIB_IMPORTS and distribute NatNetLib.dll with your application
Note : Be sure to link to ws2_32.lib if linking to NatLetLib statically.

BUILDING A NATIVE SERVER TO SEND NATNET DATA

Steps for building a NatNet server application/library to send/forward NatNet formatted data to a NatNet client application:
1. Adapt SimpleServer (SampleServer.cpp) to your application's code.
2. Include NatNetServer.h, NatNetHelper.h, and NatNetTypes.h
3. Link to NatNetLib.lib (dynamic) OR NatNetLibStatic.lib (static)
4. [OPTIONAL] if linking dynamically, define NATNETLIB_IMPORTS and distribute NatNetLib.dll with your application
Note : Be sure to link to ws2_32.lib if linking to NatLetLib statically.

BUILDING A MANAGED .NET CLIENT TO RECEIVE NATNET DATA

Steps for building a managed NatNet client application.
1. Add the NatNetML.dll .NET assembly as a reference to your VB.NET/C# project.
2. The NatNetML namespace is now available to your code, in addition to intellisense library comments.
Note : When distributing your .NET application, be sure to distribute the NatNetML.dll as well.

For more details, you can refer NATNET SDK API (http://www.naturalpoint.com/optitrack/static/documents/NatNet%20API%20User%20Guide.pdf).

Programming

The source code to receive marker data and rigid body data is provided below:

To be compiled with Visual C++ compiler and with NATNET SDK

```
//=============================================================================
// Copyright ?2010 NaturalPoint, Inc. All Rights Reserved.
// Modified by Kiwon Sohn for Personal Use. skw1125@gmail.com
//=============================================================================

/*

SampleClient.cpp

This program connects to a NatNet server, receives a data stream, and writes that data stream
to an ascii file.  The purpose is to illustrate using the NatNetClient class.

Usage [optional]:

	SampleClient [ServerIP] [LocalIP] [OutputFilename]

	[ServerIP]			IP address of the server (e.g. 192.168.0.107) ( defaults to local
machine)
	[OutputFilename]	Name of points file (pts) to write out.  defaults to Client-output.pts

*/

#include <stdio.h>
#include <tchar.h>
#include <conio.h>
#include <winsock2.h>
#include <math.h>

#include "NatNetTypes.h"
#include "NatNetClient.h"

#pragma warning( disable : 4996 )

void _WriteHeader(FILE* fp, sDataDescriptions* pBodyDefs);
void _WriteFrame(FILE* fp, sFrameOfMocapData* data);
void _WriteFooter(FILE* fp);
void __cdecl DataHandler(sFrameOfMocapData* data, void* pUserData);		// receives data from
```

```
the server
void __cdecl MessageHandler(int msgType, char* msg);        // receives NatNet error mesages
void resetClient();
int CreateClient(int iConnectionType);

unsigned int MyServersDataPort = 3130;
unsigned int MyServersCommandPort = 3131;

NatNetClient* theClient;
FILE* fp;

char szMyIPAddress[128] = "";
char szServerIPAddress[128] = "";

int _tmain(int argc, _TCHAR* argv[])
{
    int iResult;
    int iConnectionType = ConnectionType_Multicast;
    //int iConnectionType = ConnectionType_Unicast;

    // parse command line args
    if(argc>1)
    {
        strcpy(szServerIPAddress, argv[1]);  // specified on command line
        printf("Connecting to server at %s...₩n", szServerIPAddress);
    }
    else
    {
        strcpy(szServerIPAddress, "");              // not specified - assume server is local machine
        printf("Connecting to server at LocalMachine₩n");
    }
    if(argc>2)
    {
        strcpy(szMyIPAddress, argv[2]);      // specified on command line
        printf("Connecting from %s...₩n", szMyIPAddress);
    }
    else
    {
        strcpy(szMyIPAddress, "");          // not specified - assume server is local machine
        printf("Connecting from LocalMachine...₩n");
    }

    // Create NatNet Client
    iResult = CreateClient(iConnectionType);
    if(iResult != ErrorCode_OK)
    {
        printf("Error initializing client.  See log for details.  Exiting");
        return 1;
    }
    else
    {
        printf("Client initialized and ready.₩n");
    }
```

```
        // send/receive test request
        printf("[SampleClient] Sending Test Request₩n");
        void* response;
        int nBytes;
        iResult = theClient->SendMessageAndWait("TestRequest", &response, &nBytes);
        if (iResult == ErrorCode_OK)
        {
                printf("[SampleClient] Received: %s", (char*)response);
        }

        // Retrieve Data Descriptions from server
        printf("₩n₩n[SampleClient] Requesting Data Descriptions...");
        sDataDescriptions* pDataDefs = NULL;
        int nBodies = theClient->GetDataDescriptions(&pDataDefs);
        if(!pDataDefs)
        {
                printf("[SampleClient] Unable to retrieve Data Descriptions.");
                //return 1;
        }
        else
        {
         printf("[SampleClient] Received %d Data Descriptions:₩n", pDataDefs->nDataDescriptions );
         for(int i=0; i < pDataDefs->nDataDescriptions; i++)
         {
             printf("Data Description # %d (type=%d)₩n", i, pDataDefs->arrDataDescriptions[i].type);
             if(pDataDefs->arrDataDescriptions[i].type == Descriptor_MarkerSet)
             {
                 // MarkerSet
                 sMarkerSetDescription* pMS = pDataDefs-
>arrDataDescriptions[i].Data.MarkerSetDescription;
                 printf("MarkerSet Name : %s₩n", pMS->szName);
                 for(int i=0; i < pMS->nMarkers; i++)
                     printf("%s₩n", pMS->szMarkerNames[i]);

             }
             else if(pDataDefs->arrDataDescriptions[i].type == Descriptor_RigidBody)
             {
                 // RigidBody
                 sRigidBodyDescription* pRB = pDataDefs-
>arrDataDescriptions[i].Data.RigidBodyDescription;
                 printf("RigidBody Name : %s₩n", pRB->szName);
                 printf("RigidBody ID : %d₩n", pRB->ID);
                 printf("RigidBody Parent ID : %d₩n", pRB->parentID);
                 printf("Parent Offset : %3.2f,%3.2f,%3.2f₩n", pRB->offsetx, pRB->offsety, pRB->offsetz);
             }
             else if(pDataDefs->arrDataDescriptions[i].type == Descriptor_Skeleton)
             {
                 // Skeleton
                 sSkeletonDescription* pSK = pDataDefs->arrDataDescriptions[i].Data.SkeletonDescription;
                 printf("Skeleton Name : %s₩n", pSK->szName);
                 printf("Skeleton ID : %d₩n", pSK->skeletonID);
                 printf("RigidBody (Bone) Count : %d₩n", pSK->nRigidBodies);
                 for(int j=0; j < pSK->nRigidBodies; j++)
                 {
                     sRigidBodyDescription* pRB = &pSK->RigidBodies[j];
                     printf("  RigidBody Name : %s₩n", pRB->szName);
                     printf("  RigidBody ID : %d₩n", pRB->ID);
                     printf("  RigidBody Parent ID : %d₩n", pRB->parentID);
                     printf("  Parent Offset : %3.2f,%3.2f,%3.2f₩n", pRB->offsetx, pRB->offsety, pRB-
 >offsetz);
                 }
             }
             else
             {
                 printf("Unknown data type.");
                 // Unknown
             }
         }
         }


         // Create data file for writing received stream into
         char szFile[MAX_PATH];
         char szFolder[MAX_PATH];
         GetCurrentDirectory(MAX_PATH, szFolder);
         if(argc > 3)
                sprintf(szFile, "%s₩₩%s", szFolder, argv[3]);
         else
                sprintf(szFile, "%s₩₩Client-output.pts",szFolder);
         fp = fopen(szFile, "w");
         if(!fp)
         {
                printf("error opening output file %s.  Exiting.", szFile);
                exit(1);
         }
         if(pDataDefs)
                _WriteHeader(fp, pDataDefs);

         // Ready to receive marker stream!
         printf("₩nClient is connected to server and listening for data...₩n");
         int c;
         bool bExit = false;
         while(c =_getch())
         {
                switch(c)
                {
                       case 'q':
                               bExit = true;
                               break;
                       case 'r':
                               resetClient();
                               break;
             case 'p':
                 sServerDescription ServerDescription;
                 memset(&ServerDescription, 0, sizeof(ServerDescription));
                 theClient->GetServerDescription(&ServerDescription);
                 if(!ServerDescription.HostPresent)
                 {
                     printf("Unable to connect to server. Host not present. Exiting.");
                     return 1;
                 }
                 break;
             case 'f':
                 {
                     sFrameOfMocapData* pData = theClient->GetLastFrameOfData();
                     printf("Most Recent Frame: %d", pData->iFrame);
                 }
                 break;
             case 'm':                          // change to multicast
                 iResult = CreateClient(ConnectionType_Multicast);
                 if(iResult == ErrorCode_OK)
                     printf("Client connection type changed to Multicast.₩n₩n");
                 else
                     printf("Error changing client connection type to Multicast.₩n₩n");
                 break;
             case 'u':                          // change to unicast
                 iResult = CreateClient(ConnectionType_Unicast);
                 if(iResult == ErrorCode_OK)
                     printf("Client connection type changed to Unicast.₩n₩n");
                 else
                     printf("Error changing client connection type to Unicast.₩n₩n");
                 break;


                       default:
                               break;
                }
                if(bExit)
                       break;
         }

         // Done - clean up.
         theClient->Uninitialize();
         _WriteFooter(fp);
         fclose(fp);

         return ErrorCode_OK;
     }

     // Establish a NatNet Client connection
     int CreateClient(int iConnectionType)
     {
         // release previous server
         if(theClient)
         {
         theClient->Uninitialize();
         delete theClient;
         }

         // create NatNet client
         theClient = new NatNetClient(iConnectionType);
         // [optional] use old multicast group
         //theClient->SetMulticastAddress("224.0.0.1");

         // print version info
         unsigned char ver[4];
         theClient->NatNetVersion(ver);
         printf("NatNet Sample Client (NatNet ver. %d.%d.%d.%d)₩n", ver[0], ver[1], ver[2], ver[3]);

         // Set callback handlers
         theClient->SetMessageCallback(MessageHandler);
         theClient->SetVerbosityLevel(Verbosity_Debug);
         theClient->SetDataCallback( DataHandler, theClient );   // this function will receive data from the
     server

         // Init Client and connect to NatNet server
         // to use NatNet default port assigments
         int retCode = theClient->Initialize(szMyIPAddress, szServerIPAddress);
         // to use a different port for commands and/or data:
         //int retCode = theClient->Initialize(szMyIPAddress, szServerIPAddress, MyServersCommandPort,
     MyServersDataPort);
         if (retCode != ErrorCode_OK)
         {
         printf("Unable to connect to server.  Error code: %d. Exiting", retCode);
         return ErrorCode_Internal;
         }
         else
         {
         // print server info
         sServerDescription ServerDescription;
         memset(&ServerDescription, 0, sizeof(ServerDescription));
         theClient->GetServerDescription(&ServerDescription);
         if(!ServerDescription.HostPresent)
         {
             printf("Unable to connect to server. Host not present. Exiting.");
             return 1;
         }
         printf("[SampleClient] Server application info:₩n");
         printf("Application: %s (ver. %d.%d.%d.%d)₩n", ServerDescription.szHostApp,
     ServerDescription.HostAppVersion[0],
             ServerDescription.HostAppVersion[1],ServerDescription.HostAppVersion[2],ServerDescription.Ho
     stAppVersion[3]);
         printf("NatNet Version: %d.%d.%d.%d₩n", ServerDescription.NatNetVersion[0],
     ServerDescription.NatNetVersion[1],
             ServerDescription.NatNetVersion[2], ServerDescription.NatNetVersion[3]);
         printf("Client IP:%s₩n", szMyIPAddress);
         printf("Server IP:%s₩n", szServerIPAddress);
         printf("Server Name:%s₩n₩n", ServerDescription.szHostComputerName);
         }

         return ErrorCode_OK;

     }
```

```
// DataHandler receives data from the server
void __cdecl DataHandler(sFrameOfMocapData* data, void* pUserData)
{
        NatNetClient* pClient = (NatNetClient*) pUserData;

        printf("Received frame %d₩n", data->iFrame);
        if(fp)
                _WriteFrame(fp,data);
        int i=0;

    // same system latency test
    float fThisTick = (float)GetTickCount();
    float fDiff = fThisTick - data->fLatency;
    double dDuration = fDiff;
    printf("Latency (same system) (msecs): %3.2lf₩n", dDuration);


        // Other Markers
        printf("Other Markers [Count=%d]₩n", data->nOtherMarkers);
        for(i=0; i < data->nOtherMarkers; i++)
        {
                printf("Other Marker %d : %3.2f₩t%3.2f₩t%3.2f₩n",
                        i,
                        data->OtherMarkers[i][0],
                        data->OtherMarkers[i][1],
                        data->OtherMarkers[i][2]);
        }

        // Rigid Bodies
        printf("Rigid Bodies [Count=%d]₩n", data->nRigidBodies);
        for(i=0; i < data->nRigidBodies; i++)
        {
                printf("Rigid Body [ID=%d  Error=%3.2f]₩n", data->RigidBodies[i].ID, data-
>RigidBodies[i].MeanError);
                printf("₩tx₩ty₩tz₩tqx₩tqy₩tqz₩tqw₩n");
                printf("₩t%3.2f₩t%3.2f₩t%3.2f₩t%3.2f₩t%3.2f₩t%3.2f₩t%3.2f₩n",
                        data->RigidBodies[i].x,
                        data->RigidBodies[i].y,
                        data->RigidBodies[i].z,
                        data->RigidBodies[i].qx,
                        data->RigidBodies[i].qy,
                        data->RigidBodies[i].qz,
                        data->RigidBodies[i].qw);

                printf("₩tRigid body markers [Count=%d]₩n", data->RigidBodies[i].nMarkers);
                for(int iMarker=0; iMarker < data->RigidBodies[i].nMarkers; iMarker++)
                {
              printf("₩t₩t");
              if(data->RigidBodies[i].MarkerIDs)
                  printf("MarkerID:%d", data->RigidBodies[i].MarkerIDs[iMarker]);
              if(data->RigidBodies[i].MarkerSizes)
                  printf("₩tMarkerSize:%3.2f", data->RigidBodies[i].MarkerSizes[iMarker]);
              if(data->RigidBodies[i].Markers)
                  printf("₩tMarkerPos:%3.2f,%3.2f,%3.2f₩n" ,
                    data->RigidBodies[i].Markers[iMarker][0],
                    data->RigidBodies[i].Markers[iMarker][1],
                    data->RigidBodies[i].Markers[iMarker][2]);
        }
       }

    // skeletons
    printf("Skeletons [Count=%d]₩n", data->nSkeletons);
    for(i=0; i < data->nSkeletons; i++)
    {
        sSkeletonData skData = data->Skeletons[i];
        printf("Skeleton [ID=%d  Bone count=%d]₩n", skData.skeletonID, skData.nRigidBodies);
        for(int j=0; j< skData.nRigidBodies; j++)
        {
            sRigidBodyData rbData = skData.RigidBodyData[j];
            printf("Bone %d₩t%3.2f₩t%3.2f₩t%3.2f₩t%3.2f₩t%3.2f₩t%3.2f₩t%3.2f₩n",
                    rbData.ID, rbData.x, rbData.y, rbData.z, rbData.qx, rbData.qy, rbData.qz,
rbData.qw );

            printf("₩tRigid body markers [Count=%d]₩n", rbData.nMarkers);
            for(int iMarker=0; iMarker < data->RigidBodies[i].nMarkers; iMarker++)
            {
                printf("₩t₩t");
                if(rbData.MarkerIDs)
                    printf("MarkerID:%d", rbData.MarkerIDs[iMarker]);
                if(rbData.MarkerSizes)
                    printf("₩tMarkerSize:%3.2f", rbData.MarkerSizes[iMarker]);
                if(rbData.Markers)
                    printf("₩tMarkerPos:%3.2f,%3.2f,%3.2f₩n" ,
                    data->RigidBodies[i].Markers[iMarker][0],
                    data->RigidBodies[i].Markers[iMarker][1],
                    data->RigidBodies[i].Markers[iMarker][2]);
            }
        }
    }



}

// MessageHandler receives NatNet error/debug messages
void __cdecl MessageHandler(int msgType, char* msg)
{
       printf("₩n%s₩n", msg);
}

/* File writing routines */
void _WriteHeader(FILE* fp, sDataDescriptions* pBodyDefs)
{
        int i=0;

    if(!pBodyDefs->arrDataDescriptions[0].type == Descriptor_MarkerSet)
        return;
        sMarkerSetDescription* pMS = pBodyDefs->arrDataDescriptions[0].Data.MarkerSetDescription;

        fprintf(fp, "<MarkerSet>₩n₩n");
        fprintf(fp, "<Name>₩n%s₩n</Name>₩n₩n", pMS->szName);

        fprintf(fp, "<Markers>₩n");
        for(i=0; i < pMS->nMarkers; i++)
        {
                fprintf(fp, "%s₩n", pMS->szMarkerNames[i]);
        }
        fprintf(fp, "</Markers>₩n₩n");

        fprintf(fp, "<Data>₩n");
        fprintf(fp, "Frame#₩t");
        for(i=0; i < pMS->nMarkers; i++)
        {
                fprintf(fp, "M%dX₩tM%dY₩tM%dZ₩t", i, i, i);
        }
        fprintf(fp,"₩n");
    fprintf(fp,"₩n");
        fprintf(fp,"Bone: 'Hip','Ab','Chest', 'Neck', 'Head', 'N', 'LShoulder', 'LUarm', 'LFarm',
'LHand', 'N', 'RShoulder', 'RUarm', 'RFarm',
'RHand','N','LThigh','LShin','LFoot','N','LThigh','LShin','LFoot','N'");
    fprintf(fp,"₩n");


}


inline void RadiansToDegrees(float *value)
{
    *value = (*value)*(180.0f/3.14159265f);
}

void GetEulers(float qx, float qy, float qz, float qw, float *angle1,float *angle2, float *angle3)
{
    float &heading = *angle1;
    float &attitude = *angle2;
    float &bank = *angle3;

        double test = qw*qx + qy*qz;
        if (test > 0.499) { // singularity at north pole
                heading = (float) 2.0f * atan2(qy,qw);
                attitude = 3.14159265f/2.0f;
                bank = 0;

         RadiansToDegrees(&heading);
         RadiansToDegrees(&attitude);
         RadiansToDegrees(&bank);
                return;
        }
        if (test < -0.499) { // singularity at south pole
                heading = (float) -2.0f * atan2(qy,qw);
                attitude = - 3.14159265f/2.0f;
                bank = 0;

         RadiansToDegrees(&heading);
         RadiansToDegrees(&attitude);
         RadiansToDegrees(&bank);
                return;
        }
    double sqx = qx*qx;
    double sqy = qy*qy;
    double sqz = qz*qz;
    heading = (float) atan2((double)2.0*qw*qz-2.0*qx*qy , (double)1 - 2.0*sqz - 2.0*sqx);
        attitude = (float)asin(2.0*test);
        bank = (float) atan2((double)2.0*qw*qy-2.0*qx*qz , (double)1.0 - 2.0*sqy - 2.0*sqx);

    RadiansToDegrees(&heading);
    RadiansToDegrees(&attitude);
    RadiansToDegrees(&bank);
}



void _WriteFrame(FILE* fp, sFrameOfMocapData* data)
{   fprintf(fp, "₩n");  fprintf(fp, "Frame ");
        fprintf(fp, "%d ", data->iFrame);
        /*for(int i =0; i < data->MocapData->nMarkers; i++)
        {
                fprintf(fp, "₩t%.5f₩t%.5f₩t%.5f", data->MocapData->Markers[i][0], data->MocapData-
>Markers[i][1], data->MocapData->Markers[i][2]);
        }*/

        /*for(int i =0; i < data->nRigidBodies; i++)
        {
                fprintf(fp, "₩t%3.2f₩t%3.2f₩t%3.2f₩t%3.2f₩t%3.2f₩t%3.2f₩t%3.2f₩n",
                        data->RigidBodies[i].x,
                        data->RigidBodies[i].y,
                        data->RigidBodies[i].z,
                        data->RigidBodies[i].qx,
                        data->RigidBodies[i].qy,
                        data->RigidBodies[i].qz,
                        data->RigidBodies[i].qw);
        }*/


            for(int i=0; i < data->nSkeletons; i++)
    {
        sSkeletonData skData = data->Skeletons[i];
        fprintf(fp, "Skeleton [ID=%d  Bone count=%d]₩n", skData.skeletonID, skData.nRigidBodies);
        for(int j=0; j< skData.nRigidBodies; j++)
        {   fprintf(fp, "₩n");
            sRigidBodyData rbData = skData.RigidBodyData[j];

                        float yaw, pitch, roll;
            GetEulers(rbData.qx, rbData.qy, rbData.qz, rbData.qw, &yaw, &pitch, &roll);
            //fprintf(fp,"%s ", bonename[j]);
            fprintf(fp, "Bone %d₩n Position %3.2f₩t%3.2f₩t%3.2f₩n Rotation %3.2f₩t%3.2f₩t%3.2f₩t%3.2f₩n
Euler %3.2f₩t%3.2f₩t%3.2f₩n",
                    rbData.ID, 1.0*rbData.x, rbData.y, rbData.z, rbData.qx, rbData.qy, rbData.qz,
rbData.qw , yaw, pitch, roll);

            fprintf(fp, "₩tRigid body markers [Count=%d]₩n", rbData.nMarkers);
            //for(int iMarker=0; iMarker < data->RigidBodies[i].nMarkers; iMarker++)
                   for(int iMarker=0; iMarker < rbData.nMarkers; iMarker++)
            {
                fprintf(fp, "₩t₩t");
                if(rbData.Markers)
                    fprintf(fp, "₩tMarkerPos:%3.2f,%3.2f,%3.2f₩n" ,
                    1.0*rbData.Markers[iMarker][0],
                    rbData.Markers[iMarker][1],
                    rbData.Markers[iMarker][2]);
            }
        }
    }

       fprintf(fp, "₩n");
}

void _WriteFooter(FILE* fp)
{
       fprintf(fp, "</Data>₩n₩n");
       fprintf(fp, "</MarkerSet>₩n");
}




void resetClient()
{
       int iSuccess;

       printf("₩n₩nre-setting Client₩n₩n.");

       iSuccess = theClient->Uninitialize();
       if(iSuccess != 0)
              printf("error un-initting Client₩n");

       iSuccess = theClient->Initialize(szMyIPAddress, szServerIPAddress);
       if(iSuccess != 0)
              printf("error re-initting Client₩n");


}
```

We will also upload this code “SampleClient.cpp” in our Website.

Final Words

Objective of Tutorial 1 and 2 was to make readers understand how to install and use Arena Motion Capture System. Tutorial 1 and 2 complete construction details, calibration and capturing for this system. Once the concepts were conveyed the reader could setup systems and get motion data of targets to be tracked. If you have questions, please feel free to contact with me. You can also get solutions from Arena website. They have online tutorials and show all of processes in this tutorial in details.