YOST LABS
630 Second Street
Portsmouth Ohio 45662, USA

Phone: 740.876.4936
info@yostlabs.com
yostlabs.com

AN2016.05

Getting data from the Yost Labs Skeleton API

Nick Leyder
Lead Software Engineer, Yost Labs

Introduction

The PrioVR™ Suit is a full-body suit that tracks motion using 19 PrioVR Sensors and a central PrioVR Hub. The PrioVR Sensors are inertial measurement units that give accurate drift-free absolute orientations of the user's body. The PrioVR Hub collects the data from the PrioVR Sensors, and sends it wirelessly to the PrioVR Basestation. The PrioVR Basestation communicates the received data to the host computer via USB (shown in Figure 1).

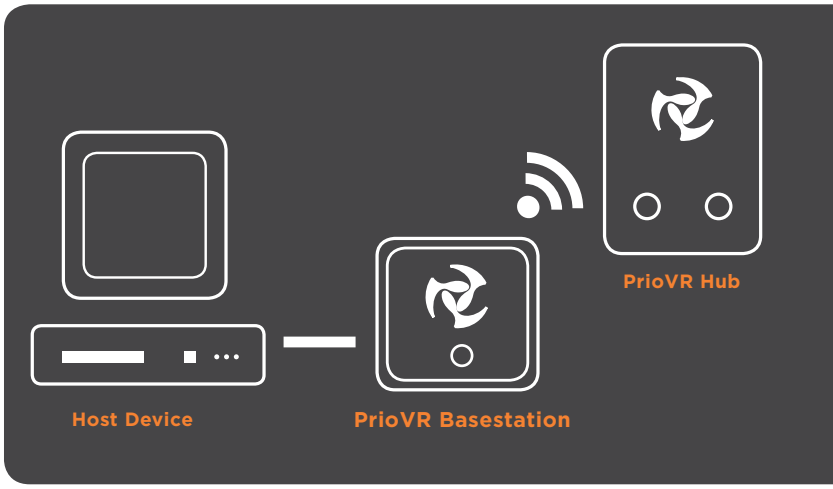

Figure 1

The purpose of this document is to provide an overview of retrieving skeleton data from a PrioVR Skeleton using the PrioVR Basestation. The example code below illustrates the use of the Skeleton API to read the global orientations and the PrioVR Joysticks from the PrioVR Skeleton and to print them to the screen.

Description

In order to retrieve skeleton data from the API, the Skeleton API must:

1. Create the skeleton
2. Create the Prio Processor
3. Add the Prio Processor to the skeleton
4. Calibrate the Prio Processor
5. Start Streaming
6. Read the data
7. Read the joystick data
8. Destroy the skeleton and prio processor and deinitialize the Skeleton API to allow for later use.

Step 1: Create the Skeleton

The first step in using a PrioVR through the Yost Skeletal API is to create a skeleton to work with. While complex bone hierarchies can be set up manually by the user, the API automatically creates standard humanoid bone hierarchies with skeleton creation.

```
// Whether or not the user is a male
bool is_male = false;
// The height of the user
float height = 1.8;
// Create a skeleton based off of the user's sex, and height
yost_skeleton_id skel_id = yostskel_createStandardSkeletonWithHeight(is_male, height);
// Tell the skeleton which bones the sensors should update
yostskel_setStandardDeviceXmlMapPrioProLayout(skel_id);
```

This code takes simple parameters about the intended user of the PrioVR Dev Kit, those being the gender and height of the user, and gives the ID of a skeleton object which the API creates. This ID will be passed to any further function that references the skeleton object. In this example, a 1.8m tall female is the suit user. The API uses these parameters to make an appropriately sized bone hierarchy for the skeleton.

Step 2: Create the Prio Processor

With the skeleton and bone hierarchy created, in order to get the PrioVR suit to drive the skeleton's bones, a Prio Processor must be created and attached to the skeleton.

```
// An index into the array of communication ports found by the Skeleton API
uint8_t com_offset = 0;
// Create the Prio Processor at the given com port with a found PrioVR
// Basestation, and assign a unique identifier to prio_id used by the Skeleton API to
// identify the Prio Processor
yost_processor_id prio_id = yostskel_createPrioProcessorWithComOffset(PRIO_TYPE::PRIO_BS, 0);
```

This code will find a PrioVR suit connected to the host PC at the given offset in the communication ports (com_offset). If none is present, prio_id will be set to an error value. This ID will be passed to further functions that reference the Prio Processor.

Step 3: Adding the Prio Processor to the skeleton

With the Prio Processor created all that remains is to add it to the skeleton. This example will add it at the end of the skeleton's processor list:

```
#define YOST_SKELETON_PROCESSOR_LIST_END 0xffffffff
// Attach the Prio Processor to the skeleton
yostskel_addProcessorToSkeleton(skel_id, YOST_SKELETON_PROCESSOR_LIST_END, prio_id);
```

If the skeleton was set up with a standard bone hierarchy as it was in this example, PrioVR sensors will be automatically assigned to the appropriate bones.

Step 4: Calibrate the Prio Processor

Before the Skeleton API can begin gathering proper data from the PrioVR, it needs to be calibrated. This process involves instructing the user to assume a particular pose, after which the sensor data will be adjusted based on the assumption that the user is in the given pose. The Skeleton API will set the skeleton to a "T" pose, and PrioVR calibration will be performed by having the user assume the same "T" pose. The user's arms are held out to the sides with the palms facing down. The user's feet are together and facing forward.

This process also detects and cancels the user's facing direction, causing bone orientations to be in a reference frame defined by the direction the user was facing during calibration. This is useful if the user needs to face a screen to play a game using the PrioVR, as the orientation data will be in a reference frame defined by the screen so long as the user faces it as they calibrate.

The calling application is responsible for informing the user to assume the given pose. The calling application can either allow the user the appropriate amount of time to enter the pose, or have the API wait for an amount of time before calibrating. This code has the API give the user two seconds to get into the correct pose before calibrating.

```
// Set the skeleton to a "T"-pose
yostskel_setSkeletonToStandardTPose(skel_id);
// An amount of time to sleep in seconds before calibrating
float wait_time = 2.0;
// Calibrate the Prio Processor
yostskel_calibratePrioProcessor(prio_id, wait_time);
```

After the wait_time, the the skeleton will be in a corresponding "T" pose, and all orientation data will have been corrected for the user's facing.

Step 5: Start Streaming

With the Prio Processor calibrated, it is now time to specify what data to gather and to instruct the Prio Processor tobegin gathering that data from the PrioVR suit.

```
// Setup the Prio Processor to stream quaternion data
yostskel_setupStandardStreamPrioProcessor(prio_id);
// Start the Prio Processor
yostskel_startPrioProcessor(prio_id);
```

Step 6: Reading the data

At this point, orientation data will be automatically obtained from the PrioVR at a regular interval. The next step is to tell the skeleton to update its bone orientations using the PrioVR data, and to bring that bone orientation data into an application. Note that this example will only read orientation data from

the chest sensor--in an actual application, this would read data from all the bones of the skeleton.

```
// For as long as you need
while(true) {
    // Update the skeleton
    yostskel_update(skel_id);
    // The quaternion for to hold the bone data
    quaternion quat;
    // Get the quaternion of the Chest bone
    yostskel_getBoneOrientation(skel_id, “Spine”, &quat);
}
```

The “quaternion” structure used here is a container for four floating point values, used here to represent an orientation. A quaternion can be converted to any other major orientation format, including rotation matrix, axis-angle, or Euler angles. The default orientation for a bone is aligned along the z-axis with the front of the bone pointing along the y-axis, and this orientation represents how far the bone has rotated from this point.

Note in the example code above, the orientation quaternion is read for the “Chest” segment of the skeleton. The API allows custom skeletal configurations and naming, but generally, the default skeleton and names can be used. By default, the following bones are available by name in a standard PrioVR skeleton:

1. Spine
2. Hips
3. Head
4. LeftShoulder
5. LeftUpperArm
6. LeftLowerArm
7. LeftHand
8. RightShoulder
9. RightUpperArm
10. RightLowerArm
11. RightHand
12. LeftUpperLeg
13. LeftLowerLeg
14. LeftFoot
15. RightUpperLeg
16. RightLowerLeg
17. RightFoot

Step 7: Read the joystick data

At this point, joystick data will be automatically obtained from the PrioVR at a regular interval. All that remains is to read the joystick from the skeleton.

```
// For as long as you want the joystick data
while(true) {
    // The variable to hold the left joystick data
    uint8_t leftJoystick[4];
    // The variable to hold the right joystick data
    uint8_t rightJoystick[4];
    // Get the joystick data from the Prio Processor
    yostskel_getJoyStickStatePrioProcessor(prio_id, &leftJoystick, &rightJoystick);
}
```

The joystick data is stored in uint8_t arrays that hold four values for the X, Y axes, uint8_t for the trigger, and the joystick button state mask. The order of the buttons in the mask (in order right to left) is: X Button, A Button, B Button, Y Button, Top Button, Joystick Button. The default value for each buttons and for the trigger is zero. The default values for the axes are 128.

Step 8: Finish Up

Before the program ends, streaming must be stopped, the Prio Processor and skeleton must be destroyed, and the skeletal API must be shut down.

```
// Stop the Prio Processor
yostskel_stopPrioProcessor(prio_id);
// Destroy the Prio Processor
yostskel_destoryProcessor(prio_id);
// Destroy the Skeleton
yostskel_destroySkeleton(skel_id);
// Destroy the Skeleton API
yostskel_resetSkeletalApi();
```

While this example includes everything required to obtain data from a PrioVR suit in an application using the YOST Skeletal API, the API features several additional tools. These tools allow name aliases to be set, and enable additional rotation offsets for bones. These tools allow for easy integration with existing skeletal systems in applications such as game engines.

Complete Example

```
#include <stdio.h>
#include <time.h>
#include “yost_skeleton_api.h”

#define YOST_SKELETON_PROCESSOR_LIST_END 0xffffffff

void main() {

    // Whether or not the user is a male
    bool is_male = false;
    // The height of the user
    float height = 1.8;
    // Create a skeleton based off of the user’s sex, and height
    yost_skeleton_id skel_id = yostskel_createStandardSkeletonWithHeight(is_male, height);

    // An index into the array of communication ports found by the Skeleton API
    uint8_t com_offset = 0;
    // Create the Prio Processor at the given com port, and assign a unique identifier
    // to prio_id used by the Skeleton API to identify the Prio Processor
    yost_processor_id prio_id = yostskel_createPrioProcessor(com_offset);

    // Attach the Prio Processor to the skeleton
    yostskel_addProcessorToSkeleton(skel_id, YOST_SKELETON_PROCESSOR_LIST_END, prio_id);

    // Tell the user to get in the T pose
    printf(“When ready, press Enter and return the T pose within 2 seconds...\n”);

    // Wait for enter
    getchar();

    // Set the skeleton to a “T”-pose
    yostskel_setSkeletonToStandardTPose(skel_id);
    // An amount of time to sleep in seconds before calibrating
    float wait_time = 2.0;
    // Calibrate the Prio Processor
    yostskel_calibratePrioProcessor(prio_id, wait_time);

    // Setup the Prio Processor to stream quaternion data
    yostskel_setupStandardStreamPrioProcessor(prio_id);
    // Start the Prio Processor
    yostskel_startPrioProcessor(prio_id);

    while(true) {

        // Update the skeleton
        yostskel_update(skel_id);
        // The quaternion for to hold the bone data
        quaternion quat;
        // Get the quaternion of the Chest bone
        yostskel_getBoneOrientation(skel_id, “Chest”, &quat);

        // The variable to hold the left joystick data
        uint8_t leftJoystick[4];
        // The variable to hold the right joystick data
        uint8_t rightJoystick[4];
```

Complete Example continued...

```
        // Get the joystick data from the Prio Processor
        yostskel_getJoyStickStatePrioProcessor(prio_id, &leftJoystick, &rightJoystick);

    }

    // Stop the Prio Processor
    yostskel_stopPrioProcessor(prio_id);
    // Destroy the Prio Processor
    yostskel_destoryProcessor(prio_id);
    // Destroy the Skeleton
    yostskel_destroySkeleton(skel_id);
    // Destroy the Skeleton API
    yostskel_resetSkeletalApi();

}
```


YOST LABS
630 Second Street
Portsmouth Ohio 45662, USA

Phone: 740.876.4936
info@yostlabs.com
yostlabs.com

Made in USA. Patents:
8498827, 8682610,
9255799, 9354058.
Additional patents pending.

About Yost Labs, Inc. We are a fast growing private company based in historic Portsmouth, Ohio. With over a decade of experience in low-latency inertial sensor innovation, we enable motion tracking in many of today's and tomorrow's most exciting products. We make virtual reality interactive. We stabilize drones and navigate autonomous cars. We measure human motion for athletic performance and rehabilitation. We are dedicated to supporting you and your team—providing expert advice and integration consulting for the world's fastest inertial motion sensor technology.

Yost Labs' innovation has been recognized with numerous patents with additional patents pending. Our customers and value-added resellers include the US Navy, US Air Force, NASA, US Army Corps of Engineers and over 1,000 leading technology firms and academic institutions around the world.