Integrating A Heart Rate (HR) Monitor into Unity VR

Goals of this lab are to discuss and demonstrate additional techniques for collecting sensor data and making it available for use within VR. We’ll work with the following concepts:

* Bluetooth-based sensor connectivity and data integration
* Simply SQL database usage
* Database integration into UnityVR
* Synchronous versus asynchronous communication
* Inter-process data communication patterns
* Using a database to asynchronously capture and provide data to VR applications
* Key processing constructs inside virtual reality development products like Unity, especially Coroutines and how they differ from frame rate-coupled processes

The lab will focus on asynchronously collecting HR data from an optical sensor and displaying it in real time within a Unity virtual reality environment.

We are going to use an inexpensive Bluetooth device (Polar Verity Sense), pair it with the Windows laptop, and read the HR data from the device and continuously display the subject’s heart rate inside Unity VR.

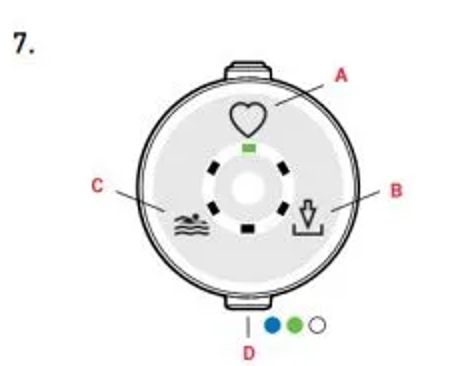
Polar Verity Sense is an optical heart rate monitor that provides freedom of movement and multiple options for viewing and recording your heart rate. With Bluetooth®, ANT+, and internal

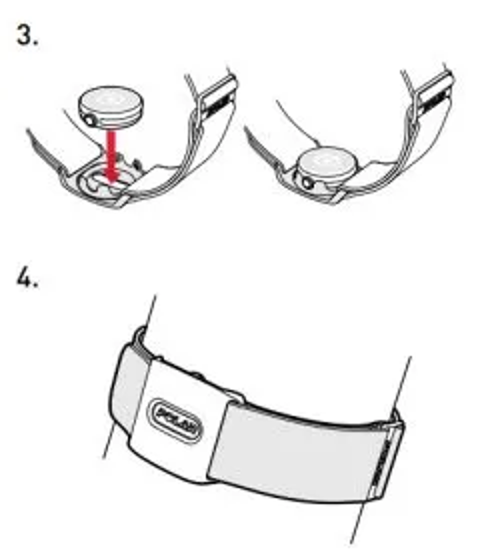
The device is a versatile high-quality optical heart rate sensor that measures heart rate from your arm or temple. It's a great alternative to heart rate chest straps or wrist based devices.

Make sure the device is charged before proceeding. Turn the sensor on by pressing the button. The LEDs indicate which mode is currently active (picture 7). To change the mode, press the button shortly. Choose (A) heart rate sensor mode to use the sensor with a compatible device or app, (B) recording mode\* to record a training session with the sensor, or (C) swimming mode\* to record a pool swimming session. After your choice, all six LEDs light up and you can start your training session. The led on the side (D) blinks blue in heart rate sensor mode, green in recording mode, and white in swimming mode. When you’re done training press and hold the button until the lights are switched off. Pictures 3 and 4 show how to insert the device into the arm band and attach to the forearm.

Wear on your arm: Place the sensor into the armband holder with the lens facing up (picture 3). Wear the armband so that the sensor is on the underside of the armband firmly against your skin (picture 4). Wear the armband around your lower or upper arm, not around your wrist (picture 5).

Read more: https://manuals.plus/polar/92083450-verity-sense-optical-heart-rate-sensor-manual#ixzz7n7g5kvfN



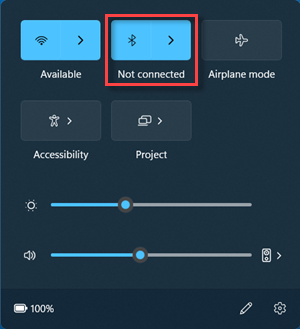


Read more: https://manuals.plus/polar/92083450-verity-sense-optical-heart-rate-sensor-manual#ixzz7n7cwnMzW

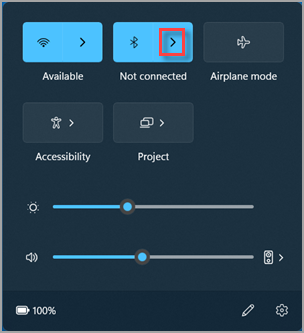
Before you start pairing, make sure that your Windows 11 PC supports Bluetooth. For more info on how to check, see [Fix Bluetooth problems in Windows](https://support.microsoft.com/windows/723e092f-03fa-858b-5c80-131ec3fba75c#WindowsVersion=Windows_11). If you need help adding a device without Bluetooth capabilities, see [Add a device to a Windows PC](https://support.microsoft.com/windows/ae095699-4d4f-40da-8702-e9662a855364).

Turn on Bluetooth

After you've checked that your Windows 11 PC supports Bluetooth, you'll need to turn it on. Here's how:

* **In Settings:** Select **Start** > **Settings** > **Bluetooth & devices**, and then turn on **Bluetooth** .
* **In quick settings:** To find the quick setting for Bluetooth, select the **Network**, **Sound**, or **Battery**icons (  ) next to the time and date on the right side of your taskbar. Select **Bluetooth** to turn it on. If it's turned on without any Bluetooth devices connected, it might appear as **Not connected**.  
    
    
    
  If you don't see **Bluetooth** in quick settings, you might need to add it. For more info, see [Change notifications and quick settings in Windows 11](https://support.microsoft.com/en-us/windows/change-notification-and-quick-settings-in-windows-ddcbbcd4-0a02-f6e4-fe14-6766d850f294).

To pair a Bluetooth device

1. Turn on your Bluetooth device and make it discoverable. The way you make it discoverable depends on the device. Check the device or visit the manufacturer's website to learn how.
2. On your PC, select the **Network**, **Sound**, or **Battery**icons (  ) next to the time and date on the right side of your taskbar.
3. Select **Manage Bluetooth devices** on the **Bluetooth** quick setting, then select your device under **Not paired**.  
     
   
4. Follow additional instructions if they appear, then select **Done**.

Your Bluetooth device and PC will usually automatically connect anytime the two devices are in range of each other with Bluetooth turned on.

Interactive Utility to Read Data from the HR Sensor

On the laptop, locate the BLEMonitor utility (located in: ???). This application is a .NET C# command line utility that can interact with the HR device. In the same director at the utility is a configuration file that contains certain setup data for the utility. We will need to record some device information from the Polar Verity Sense HR monitor for integration with Unity VR.

Once the utility has started issues the following commands as shown in the screen shot below. Record the “key” associated with the HR monitor device as illustrated. You can interact witj the Polar device continuously displaying the HR it is sensing from the optical sensors on the forearm. Once you’ve recorded the key information you can stop the monitor utility as show below.

Communicating Sensor Data into UnityVR

There are a variety of ways to bring data from external sources into UnityVR, including Inter-Process Communication (IPC), shared files, web services, and databases amongst others. You’ve already seen IPC in prior lab activities. The data exchange may consist if simply a signal that some event has occurred and the receiving process (UnityVR in our case) should respond to it, or it may include a payload of data to process by the receiving application

Inter-Process Communication (IPC)

Inter-Process Communication (IPC) is a mechanism that involves communication of one process with another process. This usually occurs only in one system.

Communication can be of two types −

* Between related processes initiating from only one process, such as parent and child processes.
* Between unrelated processes, or two or more different processes.

Some important concepts to understanding IPC:

Pipes − Communication between two related processes. The mechanism is half duplex meaning the first process communicates with the second process. To achieve a full duplex i.e., for the second process to communicate with the first process another pipe is required.

FIFO − Communication between two unrelated processes. FIFO is a full duplex, meaning the first process can communicate with the second process and vice versa at the same time.

Message Queues − Communication between two or more processes with full duplex capacity. The processes will communicate with each other by posting a message and retrieving it out of the queue. Once retrieved, the message is no longer available in the queue.

Shared Memory − Communication between two or more processes is achieved through a shared piece of memory among all processes. The shared memory needs to be protected from each other by synchronizing access to all the processes.

Semaphores − Semaphores are meant for synchronizing access to multiple processes. When one process wants to access the memory (for reading or writing), it needs to be locked (or protected) and released when the access is removed. This needs to be repeated by all the processes to secure data.

Signals − Signal is a mechanism to communication between multiple processes by way of signaling. This means a source process will send a signal (recognized by number) and the destination process will handle it accordingly.

IPC is a good choice for high-speed transfer for data into UnityVR, especially if the data is not needed by other processes or needs to be persisted.

Using a Database “Ring Buffer” for Data Transfer

We’re going to explore in the lab another common technique of using a lightweight database to receive and hold sensor data, HR in this case from our Polar device. This decouples the communication between the UnityVR internal processing management and the process that is collecting the sensor data. In oir simplified case, the specific data structure we are going to mimic is what is called a ring buffer.

A ring buffer is a special type of buffer with a distinct structure that changes its use case in practice. But what is a ring buffer exactly, and how do ring buffers work?

What are ring buffers?

A ring buffer (also known as a circular buffer or a circular queue) is a buffer data structure that behaves as if it had a circular shape, in which the last element in the buffer is connected to the first element. Like standard buffers, ring buffers typically have a fixed size.

There are several benefits of using a ring buffer over a standard buffer. For example, suppose that your buffer has 1000 elements in it, and you consume the first element at the start of the buffer. With a standard linear buffer, you would then need to shift all the other 999 elements one position to the left, so that you can keep inserting new elements at the end of the buffer. This can easily become very inefficient as the size of the buffer grows.

Ring buffers fix this issue by maintaining a circular structure. If you consume the first element from a ring buffer, you can simply point to the next element in the buffer as the new head of the buffer, and then insert a new element in the empty space. This pointer to the first element will rotate around the ring buffer as you continue to consume elements.

Ring buffers can be established using external files as well, and this data structure helps simplify concurrency red and writes of the file from multiple processes.

In our case, and for illustration purposes and gain some exposure to dusing databases and SQL with UnityVR, we’re going to use a very popular, open source SQL-based database that is implemented as a library. SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. The code for SQLite is in the public domain and is thus free for use for any purpose, commercial or private. SQLite is the most widely deployed database in the world with more applications than we can count, including several high-profile projects.

SQLite is an embedded SQL database engine. Unlike most other SQL databases, SQLite does not have a separate server process. SQLite reads and writes directly to ordinary disk files. A complete SQL database with multiple tables, indices, triggers, and views, is contained in a single disk file. The database file format is cross-platform - you can freely copy a database between 32-bit and 64-bit systems or between big-endian and little-endian architectures. These features make SQLite a popular choice as an Application File Format. SQLite database files are a recommended storage format by the US Library of Congress.

SQLite is a compact library. With all features enabled, the library size can be less than 750KiB, depending on the target platform and compiler optimization settings. (64-bit code is larger. And some compiler optimizations such as aggressive function inlining and loop unrolling can cause the object code to be much larger.) There is a tradeoff between memory usage and speed. SQLite generally runs faster the more memory you give it. Nevertheless, performance is usually quite good even in low-memory environments. Depending on how it is used, SQLite can be faster than direct filesystem I/O.

We are going to create a SQLite database containing a single table that acts as a ring buffer when combined with appropriate SQL. We will use a modified version of the BLEMonitor we used before to collect HR information from the Polar Verity Sense HR device but instead of simply displaying it to the console we will write it to a shared SQLite database and HR table. We will modify a simply UnityVR game to continuously read the HR data from the SQLite table, and display the HR to the user within their head mounted device (HMD)

To create the database, create a directory C:\sqlite, download and install the SQLite software into that directory. Create a SQLite database by executing the following commands as shown in the following screen shots.

<image>

A key part of establishing the database-part of the ring buffer system is pre-populating HR table with dummy rows for the collector to update. Run the hr.sql scripts to create the rows in your newly created HR table as shown below:

<image>

We will use a modified BLEMonitor program (already installed on the system) that interacts with the Polar device automatically and writes the HR data it receives to HR table you just created. A key aspect of implementing the ring buffer structure is the SQL that is used to update table. We’ll examine this code in the lab.

Remember the key information you recorded earlier? This is where we use it to configure the modified BLEMonitor that writes to the SQLite database to communicate over Bluetooth with the Polar HR device. In the directory where monitor software is located, find the configuration file (named: “name”, edit it, and replace the key value with the one you recorded earlier as shown below:

<image>

You can verify that the program is working properly and recording data to the database by reviewing its log file located in: <user account specific>, or looking at the contents of the HR table using the SQLite command line utility as shown below:

<image>

At this point, we now have continuous collection of HR data into the SQLite database. Let’s switch to a simple Unity project that consumes and displays the data.

Open the Unity project name HRDisplay already installed on the laptop in the Unity editor. A couple things worth noting is that the SQLite binary library components have been included in the project to allow interaction with SQLite, and SQL and SQLite client library components have been included.

Another key feature of UnityVR we are taking advantage of is the use of coroutines.

Coroutines are methods that run independently of but simultaneously with the main execution thread inside Unity. This independence allows the Coroutine to run across frames, making it easy to perform large tasks that would otherwise bring a projects execution to a halt. Another major benefit is that the Coroutine retains flow control, allowing for easy looping and/or conditional behavior. Coroutines are an important Unity features in many data integration techniques using UnityVR. Code from the project that uses Coroutines to retrieve and update HR data within the game is shown below:

<code snippet>

Run the UnityVR game and make sure that HR data is being displayed to the subject.

You’ve now learned a bit about the following topics, all of which are important in the virtual reality programming domain and Unity:

* Bluetooth connectivity and data integration
* Database integration into UnityVR
* Synchronous versus asynchronous communication
* Inter-process data communication patterns
* Using a database to capture and provide data to VR applications
* Key processing constructs inside virtual reality development products like Unity, especially Coroutines and how they differ from frame rate-coupled processes