

# Contents

[Azure Kinect DK 文档](#)

[概述](#)

[关于 Azure Kinect DK](#)

[快速入门](#)

[设置 Azure Kinect DK](#)

[将传感器流录制到文件中](#)

[生成你的第一个应用程序](#)

[设置人体跟踪 SDK](#)

[生成第一个人体跟踪应用程序](#)

[概念](#)

[深度相机](#)

[坐标系](#)

[人体跟踪关节](#)

[人体跟踪索引映射](#)

[操作指南](#)

[使用传感器 SDK](#)

[Azure Kinect 传感器 SDK](#)

[找到并打开设备](#)

[检索图像](#)

[检索 IMU 样本](#)

[访问麦克风](#)

[使用图像转换](#)

[使用校准函数](#)

[捕获设备同步](#)

[录制和播放](#)

[使用人体跟踪 SDK](#)

[获取人体跟踪结果](#)

[访问人体帧中的数据](#)

[将 Azure Kinect 库添加到 VS 项目](#)

[更新 Azure Kinect 固件](#)

[将录制器与外部同步单元配合使用](#)

[工具](#)

[Azure Kinect 查看器](#)

[Azure Kinect 录制器](#)

[Azure Kinect 固件工具](#)

[资源](#)

[下载传感器 SDK](#)

[下载人体跟踪 SDK](#)

[系统要求](#)

[硬件规格](#)

[同步多个设备](#)

[与 Kinect for Windows 的比较](#)

[重置 Azure Kinect DK](#)

[Azure Kinect 支持](#)

[Azure Kinect 故障排除](#)

[保修、扩展服务计划以及条款和条件](#)

[安全信息](#)

[参考](#)

[传感器 API](#)

[人体跟踪 API](#)

[录制文件格式](#)

# 关于 Azure Kinect DK

2020/4/2 • [Edit Online](#)

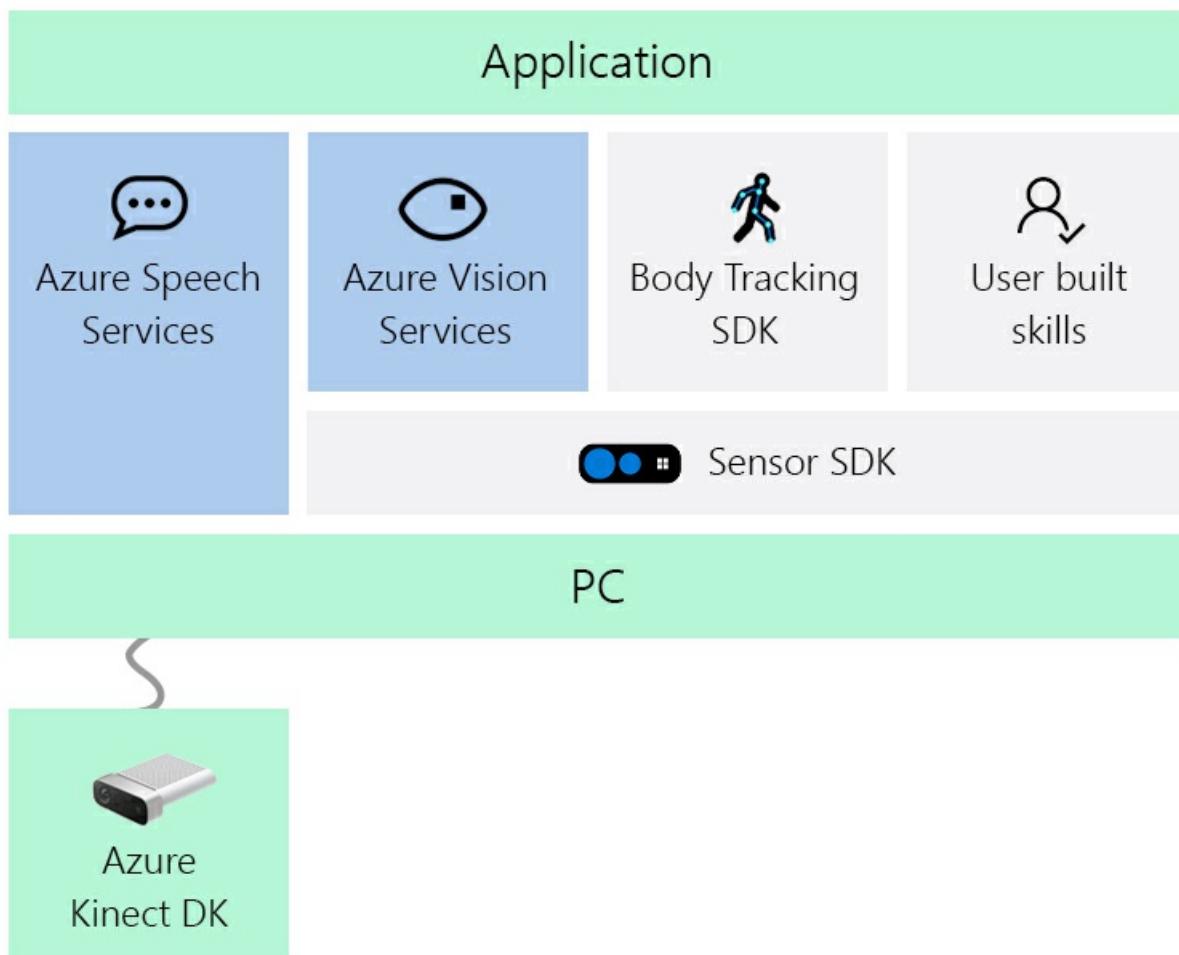


Azure Kinect DK 是一款开发人员工具包，配有先进的 AI 传感器，提供复杂的计算机视觉和语音模型。Kinect 将深度传感器、空间麦克风阵列与视频摄像头和方向传感器整合成一体式的小型设备，提供多种模式、选项和软件开发工具包 (SDK)。它可在 [Microsoft 在线商店](#) 中购买。

Azure Kinect DK 开发环境由以下多个 SDK 组成：

- 用于访问低级别传感器和设备的传感器 SDK。
- 用于跟踪 3D 人体的人体跟踪 SDK。
- 用于启用麦克风访问和基于 Azure 云的语音服务的语音认知服务 SDK。

此外，可将认知视觉服务与设备 RGB 相机配合使用。



Azure Kinect 传感器 SDK

Azure Kinect 传感器 SDK 提供低级别传感器访问用于完成 Azure Kinect DK 硬件传感器和设备配置。

若要详细了解 Azure Kinect 传感器 SDK, 请参阅[使用传感器 SDK](#)。

### Azure Kinect 传感器 SDK 功能

传感器 SDK 提供以下功能, 这些功能一经安装即可在 Azure Kinect DK 上运行:

- 深度相机访问和模式控制(被动 IR 模式, 以及宽视场和窄视场深度模式)
- RGB 相机访问和控制(例如曝光和白平衡)
- 运动传感器(陀螺仪和加速度传感器)访问
- 同步的深度 RGB 相机流, 相机之间的延迟可配置
- 外部设备同步控制, 设备之间的延迟偏移量可配置
- 用于处理图像分辨率、时间戳等的相机帧元数据访问。
- 设备校准数据访问

### Azure Kinect 传感器 SDK 工具

传感器 SDK 中提供了以下工具:

- 查看器工具, 可用于监视设备数据流和配置不同的模式。
- 使用 Matroska 容器格式的传感器录制工具和播放读取器 API。
- Azure Kinect DK 固件更新工具。

## Azure Kinect 人体跟踪 SDK

人体跟踪 SDK 包含 Windows 库和运行时, 在 Azure Kinect DK 硬件上使用时可以跟踪 3D 人体。

### Azure Kinect 人体跟踪功能

随附的 SDK 提供以下人体跟踪功能:

- 提供人体图像分段。
- 包含 FOV 中每个不完整或完整人体的在解剖学上正确的骨干。
- 提供每个人体的唯一标识。
- 可跟踪人体在不同时间的位置。

### Azure Kinect 人体跟踪工具

- 人体跟踪器提供一个查看器工具用于跟踪 3D 人体。

## 语音认知服务 SDK

语音 SDK 启用已连接到 Azure 的语音服务。

### 语音服务

- 语音转文本
- 语音翻译
- 文本转语音

#### NOTE

Azure Kinect DK 没有扬声器。

有关更多详细信息, 请访问[语音服务文档](#)。

## 视觉服务

以下 [Azure 认知视觉服务](#) 提供可在图像与视频中识别和分析内容的 Azure 服务。

- [计算机视觉](#)
- [人脸](#)
- [视频索引器](#)
- [内容审查器](#)
- [自定义视觉](#)

服务在不断发展和完善，因此，请记得定期检查新的或补充的[认知服务](#)以改进应用程序。若要抢先了解新兴服务，请查看[认知服务实验室](#)。

## Azure Kinect 硬件要求

Azure Kinect DK 将 Microsoft 的最新传感器技术集成到了与 USB 连接的单个附件中。有关详细信息，请参阅 [Azure Kinect DK 硬件规格](#)。

## 后续步骤

现在你已了解 Azure Kinect DK 的概况。接下来，请深入到它的每项功能并进行设置！

[快速入门: 设置 Azure Kinect DK](#)

# 快速入门：设置 Azure Kinect DK

2020/3/27 • [Edit Online](#)

本快速入门提供有关如何设置 Azure Kinect DK 的指导。其中介绍了如何测试传感器流可视化以及如何使用 [Azure Kinect 查看器](#)。

如果没有 Azure 订阅，请在开始之前创建一个[免费帐户](#)。

## 系统要求

查看[系统要求](#)，验证主机电脑配置是否符合所有的 Azure Kinect DK 最低要求。

## 设置硬件

### NOTE

使用本设备之前，请务必除去相机保护膜。

1. 将电源连接器插入设备背面的电源插孔。将 USB 电源适配器连接到线缆的另一端，然后将适配器插入电源插座。
2. 将 USB 数据线的一端连接到设备，将另一端连接到电脑上的 USB 3.0 端口。

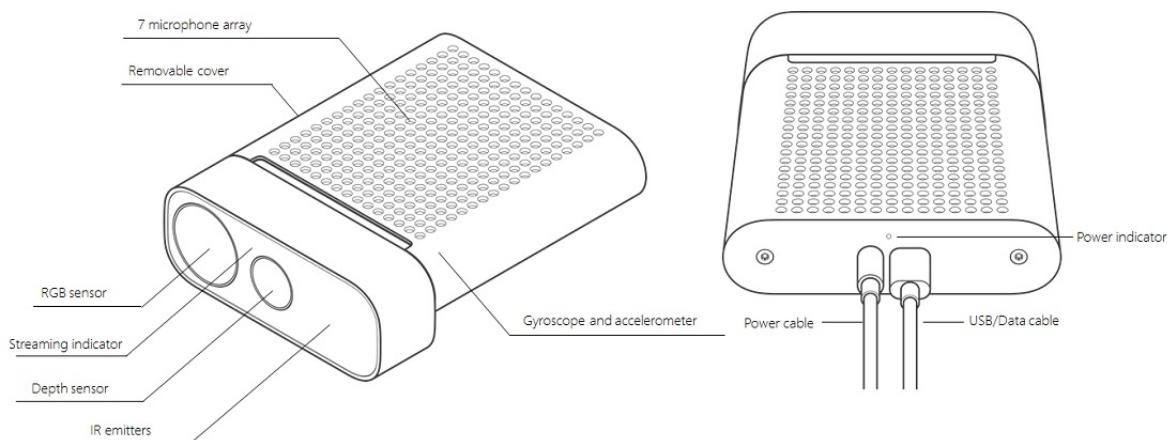
### NOTE

为了获得最佳结果，请将电缆直接连接到设备和电脑。避免在连接中使用扩展或额外的适配器。

3. 验证电源指示器 LED(USB 电缆旁边)是否为纯白。

设备通电需要几秒钟时间。当前置 LED 流指示灯熄灭时，表示设备可供使用。

有关电源指示器 LED 的详细信息，请参阅[光的含义是什么？](#)



## 下载该 SDK

1. 选择[下载 SDK](#)的链接。
2. 在电脑上安装 SDK。

## 更新固件

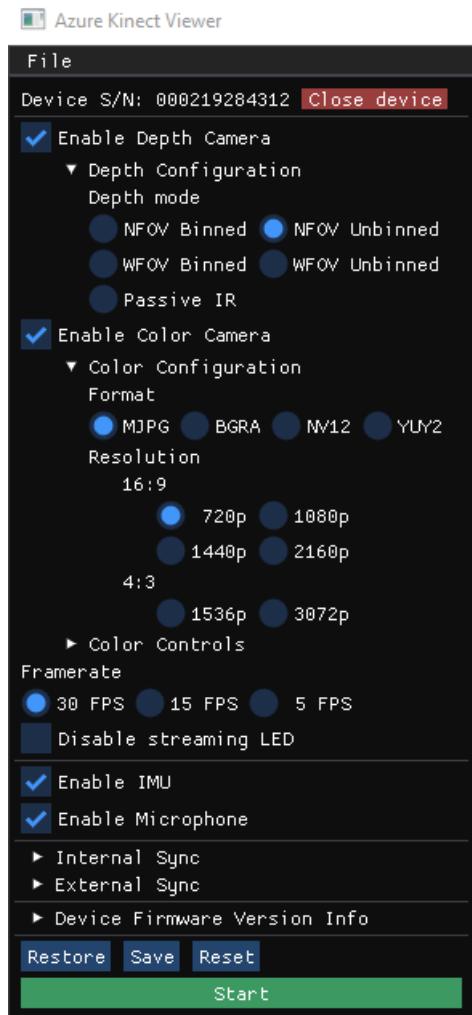
SDK 需要设备固件的最新版本才能正常工作。若要检查并更新固件版本，请按[更新 Azure Kinect DK 固件](#)中的步骤操作。

## 验证设备是否流式传输数据

1. 启动 [Azure Kinect 查看器](#)。可以使用以下方法之一来启动此工具：

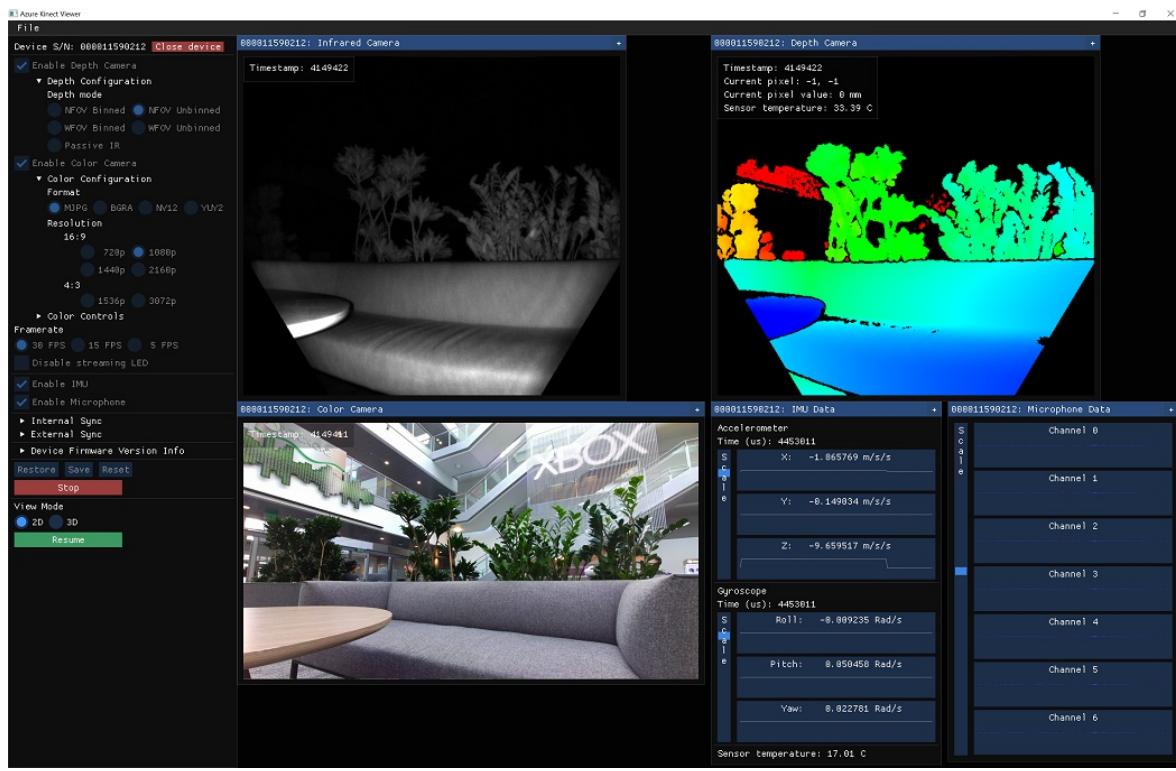
- 可以通过命令行启动查看器，也可以通过双击可执行文件来这样做。文件 `k4aviewer.exe` 位于 SDK 工具目录(例如 `C:\Program Files\Azure Kinect SDK vX.Y.Z\tools\k4aviewer.exe`，其中的 `X.Y.Z` 是安装的 SDK 版本)。
- 可以从设备的“启动”菜单启动 Azure Kinect 查看器。

2. 在 Azure Kinect 查看器中，选择“打开设备”>“启动”。



3. 验证该工具是否可视化每个传感器流：

- 深度相机
- 彩色相机
- 红外相机
- IMU
- 麦克风



现已完成 Azure Kinect DK 的设置。接下来，可以开始开发应用程序或集成服务。

如果遇到任何问题，请查看[故障排除](#)。

## 另请参阅

- [Azure Kinect DK 硬件信息](#)
- [更新设备固件](#)
- [详细了解 Azure Kinect 查看器](#)

## 后续步骤

Azure Kinect DK 准备就绪并运行后，你还可以了解如何

[将传感器流录制到文件中](#)

# 快速入门：将 Azure Kinect 传感器流录制到文件中

2020/3/27 • [Edit Online](#)

本快速入门提供有关如何使用 [Azure Kinect 录制器](#) 工具将传感器 SDK 发出的数据流录制到文件的信息。

如果没有 Azure 订阅，请在开始之前创建一个[免费帐户](#)。

## 先决条件

本快速入门假设：

- 已将 Azure Kinect DK 连接到主机电脑，并已正常通电。
- 已完成硬件[设置](#)。

## 创建录制内容

1. 打开命令提示符，提供 [Azure Kinect 录制器](#) 的路径，即 `k4arecorder.exe` 工具安装目录的位置。例如：

```
C:\Program Files\Azure Kinect SDK\tools\k4arecorder.exe
```

2. 录制 5 秒。

```
k4arecorder.exe -l 5 %TEMP%\output.mkv
```

3. 默认情况下，该录制器使用 NFOV 非装箱深度模式，输出 1080p RGB @ 30 fps 的内容（包括 IMU 数据）。有关录制选项的完整概述和提示，请参阅 [Azure Kinect 录制器](#)。

## 播放录制内容

可以使用 [Azure Kinect 查看器](#) 播放录制内容。

- 启动 `k4aviewer.exe`
- 展开“打开录制内容”选项卡并打开你的录制内容。

## 后续步骤

你现已了解如何将传感器流录制到文件中，接下来可以

[生成第一个 Azure Kinect 应用程序](#)

# 快速入门：生成第一个 Azure Kinect 应用程序

2020/3/27 • [Edit Online](#)

想要开始使用 Azure Kinect DK？本快速入门可帮助你启动并运行该设备！

如果没有 Azure 订阅，请在开始之前创建一个[免费帐户](#)。

本文将介绍以下函数：

- `k4a_device_get_installed_count()`
- `k4a_device_open()`
- `k4a_device_get_serialnum()`
- `k4a_device_start_cameras()`
- `k4a_device_stop_cameras()`
- `k4a_device_close()`

## 先决条件

1. [设置 Azure Kinect DK 设备](#)。
2. [下载并安装 Azure Kinect 传感器 SDK](#)。

## 头文件

只需要一个头文件，即 `k4a.h`。请确保所选的编译器设置为使用 SDK 的库并包含文件夹。此外，需要链接 `k4a.lib` 和 `k4a.dll` 文件。建议参阅[将 Azure Kinect 库添加到项目](#)。

```
#include <k4a/k4a.h>
```

## 查找 Azure Kinect DK 设备

可将多个 Azure Kinect DK 设备连接到计算机。首先，我们将使用 `k4a_device_get_installed_count()` 函数确定有多少个设备，或者是否连接了任何设备。此函数应可立即运行，而无需经过附加的设置。

```
uint32_t count = k4a_device_get_installed_count();
```

确定某个设备已连接到计算机后，可以使用 `k4a_device_open()` 将其打开。可以提供想要打开的设备的索引，或者只对第一个设备使用 `K4A_DEVICE_DEFAULT`。

```
// Open the first plugged in Kinect device
k4a_device_t device = NULL;
k4a_device_open(K4A_DEVICE_DEFAULT, &device);
```

与 Azure Kinect 库中的大多数内容一样，当你打开某种内容时，也应该在用完时将其关闭！关闭时，请记得调用 `k4a_device_close()`。

```
k4a_device_close(device);
```

打开设备后，可以进行测试以确保它正常工作。让我们读取设备的序列号！

```
// Get the size of the serial number
size_t serial_size = 0;
k4a_device_get_serialnum(device, NULL, &serial_size);

// Allocate memory for the serial, then acquire it
char *serial = (char*)(malloc(serial_size));
k4a_device_get_serialnum(device, serial, &serial_size);
printf("Opened device: %s\n", serial);
free(serial);
```

## 启动相机

打开设备后，需要使用 [k4a\\_device\\_configuration\\_t](#) 对象配置相机。相机配置包含大量不同的选项。请选择最适合自己的方案的设置。

```
// Configure a stream of 4096x3072 BRGA color data at 15 frames per second
k4a_device_configuration_t config = K4A_DEVICE_CONFIG_INIT_DISABLE_ALL;
config.camera_fps      = K4A_FRAMES_PER_SECOND_15;
config.color_format    = K4A_IMAGE_FORMAT_COLOR_BGRA32;
config.color_resolution = K4A_COLOR_RESOLUTION_3072P;

// Start the camera with the given configuration
k4a_device_start_cameras(device, &config);

// ...Camera capture and application specific code would go here...

// Shut down the camera when finished with application logic
k4a_device_stop_cameras(device);
```

## 错误处理。

为简洁起见，我们不会在某些内联示例中显示错误处理。但是，错误处理始终很重要！许多函数返回常规的成功/失败类型 [k4a\\_result\\_t](#)，或者包含详细信息的更具体的变量，比如 [k4a\\_wait\\_result\\_t](#)。请查看每个函数的文档或 IntelliSense，以了解该函数预期显示的错误消息！

可以使用 [K4A\\_SUCCEEDED](#) 和 [K4A\\_FAILED](#) 宏检查函数的结果。因此，除了打开 Azure Kinect DK 设备以外，我们还可以按如下所示保护函数调用：

```
// Open the first plugged in Kinect device
k4a_device_t device = NULL;
if ( K4A_FAILED( k4a_device_open(K4A_DEVICE_DEFAULT, &device) ) )
{
    printf("Failed to open k4a device!\n");
    return;
}
```

## 完整源代码

```

#pragma comment(lib, "k4a.lib")
#include <k4a/k4a.h>

#include <stdio.h>
#include <stdlib.h>

int main()
{
    uint32_t count = k4a_device_get_installed_count();
    if (count == 0)
    {
        printf("No k4a devices attached!\n");
        return 1;
    }

    // Open the first plugged in Kinect device
    k4a_device_t device = NULL;
    if (K4A_FAILED(k4a_device_open(K4A_DEVICE_DEFAULT, &device)))
    {
        printf("Failed to open k4a device!\n");
        return 1;
    }

    // Get the size of the serial number
    size_t serial_size = 0;
    k4a_device_get_serialnum(device, NULL, &serial_size);

    // Allocate memory for the serial, then acquire it
    char *serial = (char*)(malloc(serial_size));
    k4a_device_get_serialnum(device, serial, &serial_size);
    printf("Opened device: %s\n", serial);
    free(serial);

    // Configure a stream of 4096x3072 BRGA color data at 15 frames per second
    k4a_device_configuration_t config = K4A_DEVICE_CONFIG_INIT_DISABLE_ALL;
    config.camera_fps      = K4A_FRAMES_PER_SECOND_15;
    config.color_format    = K4A_IMAGE_FORMAT_COLOR_BGRA32;
    config.color_resolution = K4A_COLOR_RESOLUTION_3072P;

    // Start the camera with the given configuration
    if (K4A_FAILED(k4a_device_start_cameras(device, &config)))
    {
        printf("Failed to start cameras!\n");
        k4a_device_close(device);
        return 1;
    }

    // Camera capture and application specific code would go here

    // Shut down the camera when finished with application logic
    k4a_device_stop_cameras(device);
    k4a_device_close(device);

    return 0;
}

```

## 后续步骤

了解如何使用传感器 SDK 查找并打开 Azure Kinect DK 设备

[查找并打开设备](#)

# 快速入门：设置 Azure Kinect 人体跟踪

2020/3/27 • [Edit Online](#)

本快速入门将引导你完成在 Azure Kinect DK 上运行人体跟踪的过程。

## 系统要求

人体跟踪 SDK 要求在主机电脑中安装 NVIDIA GPU。[系统要求](#)页中描述了建议的人体跟踪主机电脑要求。

## 安装软件

### 安装最新的 NVIDIA 驱动程序

请下载并安装显卡的最新 NVIDIA 驱动程序。旧版驱动程序可能与随人体跟踪 SDK 一起重新分发的 CUDA 二进制文件不兼容。

### Visual C++ Redistributable for Visual Studio 2015

下载并安装 Microsoft Visual C++ Redistributable for Visual Studio 2015。

## 设置硬件

### 设置 Azure Kinect DK

启动 [Azure Kinect 查看器](#) 来检查是否已正确设置 Azure Kinect DK。

## 下载人体跟踪 SDK

1. 选择[下载人体跟踪 SDK](#) 的链接
2. 在电脑上安装人体跟踪 SDK。

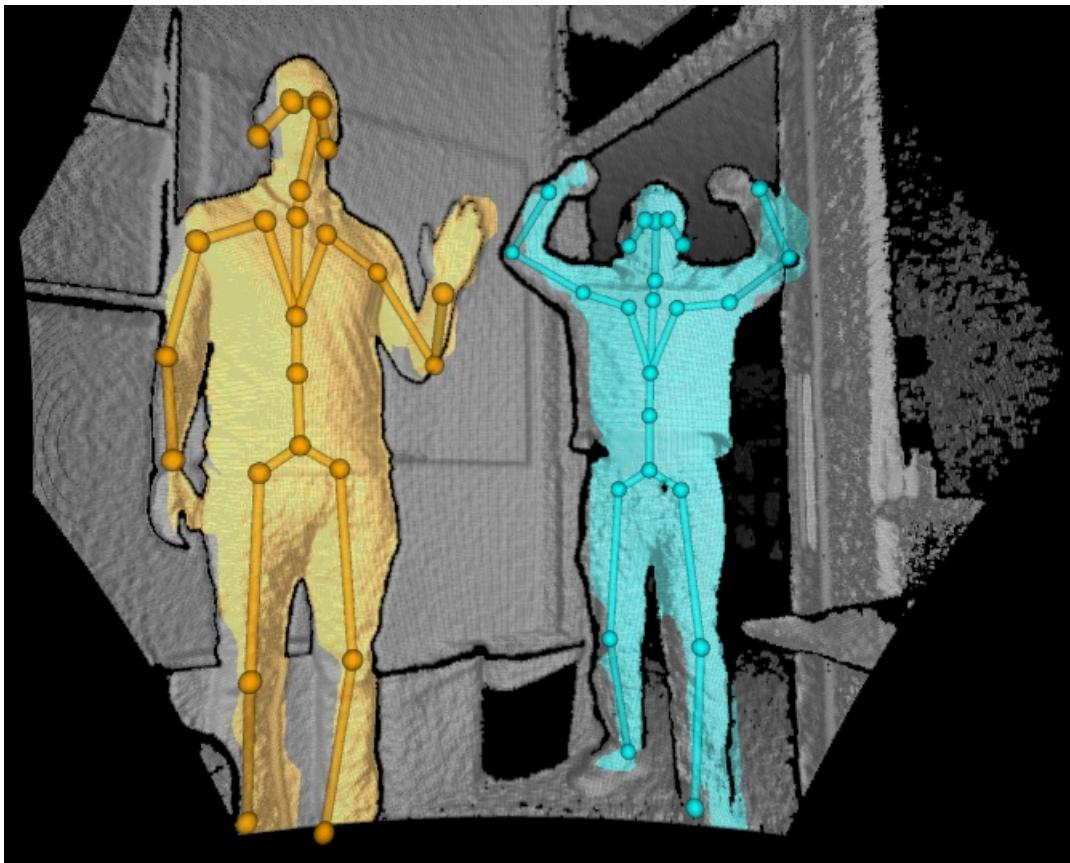
## 验证人体跟踪

启动 [Azure Kinect 人体跟踪查看器](#) 来检查是否已正确设置人体跟踪 SDK。该查看器是使用 SDK msi 安装程序安装的。可以在开始菜单或 `<SDK Installation Path>\tools\k4abt_simple_3d_viewer.exe` 中找到它。

如果没有足够强大的 GPU 但仍想测试结果，可以通过以下命令在命令行中启动 [Azure Kinect 人体跟踪查看器](#)：

`<SDK Installation Path>\tools\k4abt_simple_3d_viewer.exe CPU`

如果设置全都正确，应会显示一个窗口，其中包含 3D 点云和跟踪到的人体。



## 示例

可在[此处](#)找到有关如何使用正文跟踪 SDK 的示例。

## 后续步骤

[生成第一个人体跟踪应用程序](#)

# 快速入门：生成 Azure Kinect 人体跟踪应用程序

2020/3/27 • [Edit Online](#)

想要开始使用人体跟踪 SDK？本快速入门可帮助你启动并运行人体跟踪！可以在此 [Azure-Kinect-Sample 存储库](#) 中找到更多示例。

## 先决条件

- [设置 Azure Kinect DK](#)
- [设置人体跟踪 SDK](#)
- 已完成有关[生成第一个 Azure Kinect 应用程序](#)的快速入门。
- 熟悉以下传感器 SDK 函数：
  - `k4a_device_open()`
  - `k4a_device_start_cameras()`
  - `k4a_device_stop_cameras()`
  - `k4a_device_close()`
- 查看有关以下人体跟踪 SDK 函数的文档：
  - `k4abt_tracker_create()`
  - `k4abt_tracker_enqueue_capture()`
  - `k4abt_tracker_pop_result()`
  - `k4abt_tracker_shutdown()`
  - `k4abt_tracker_destroy()`

## 头文件

人体跟踪使用单个头文件 `k4abt.h`。请包含此头文件以及 `k4a.h`。确保所选的编译器已针对传感器 SDK 和人体跟踪 SDK `lib` 与 `include` 文件夹进行设置。还需要链接到 `k4a.lib` 和 `k4abt.lib` 文件。运行该应用程序需要 `k4a.dll`、`k4abt.dll`、`onnxruntime.dll` 和 `dnn_model.onnx` 位于应用程序执行路径中。

```
#include <k4a/k4a.h>
#include <k4abt.h>
```

## 打开设备并启动相机

第一个人体跟踪应用程序假设已将单个 Azure Kinect 设备连接到电脑。

人体跟踪基于传感器 SDK。若要使用人体跟踪，首先需要打开并配置设备。使用 `k4a_device_open()` 函数打开设备，然后使用 `k4a_device_configuration_t` 对象对其进行配置。为获得最佳结果，请将深度模式设置为 `K4A_DEPTH_MODE_NFOV_UNBINNED` 或 `K4A_DEPTH_MODE_WFOV_2X2BINNED`。如果深度模式设置为 `K4A_DEPTH_MODE_OFF` 或 `K4A_DEPTH_MODE_PASSIVE_IR`，人体跟踪器将无法运行。

在[此页](#)上可以找到有关查找和打开设备的详细信息。

可在以下页面上找到有关 Azure Kinect 深度模式的详细信息：[硬件规范](#)和 `k4a_depth_mode_t` 枚举。

```
k4a_device_t device = NULL;
k4a_device_open(0, &device);

// Start camera. Make sure depth camera is enabled.
k4a_device_configuration_t deviceConfig = K4A_DEVICE_CONFIG_INIT_DISABLE_ALL;
deviceConfig.depth_mode = K4A_DEPTH_MODE_NFOV_UNBINNED;
deviceConfig.color_resolution = K4A_COLOR_RESOLUTION_OFF;
k4a_device_start_cameras(device, &deviceConfig);
```

## 创建跟踪器

获取人体跟踪结果的第一步是创建人体跟踪器。该跟踪器需要 [k4a\\_calibration\\_t](#) 传感器校准结构。可以使用 [k4a\\_device\\_get\\_calibration\(\)](#) 函数查询传感器校准。

```
k4a_calibration_t sensor_calibration;
k4a_device_get_calibration(device, deviceConfig.depth_mode, deviceConfig.color_resolution,
&sensor_calibration);

k4abt_tracker_t tracker = NULL;
k4abt_tracker_configuration_t tracker_config = K4ABT_TRACKER_CONFIG_DEFAULT;
k4abt_tracker_create(&sensor_calibration, tracker_config, &tracker);
```

## 从 Azure Kinect 设备获取捕获

在[此页](#)上可以找到有关检索图像数据的详细信息。

```
// Capture a depth frame
k4a_device_get_capture(device, &capture, TIMEOUT_IN_MS);
```

## 将捕获排入队列并弹出结果

跟踪器在内部维护一个输入队列和一个输出队列，以便更有效地以异步方式处理 Azure Kinect DK 捕获。下一步是使用 [k4abt\\_tracker\\_enqueue\\_capture\(\)](#) 函数将新的捕获添加到输入队列。使用 [k4abt\\_tracker\\_pop\\_result\(\)](#) 函数弹出输出队列的结果。超时值与应用程序相关，控制队列等待时间。

第一人体跟踪应用程序使用实时处理模式。有关其他模式的详细说明，请参阅[获取人体跟踪结果](#)。

```
k4a_wait_result_t queue_capture_result = k4abt_tracker_enqueue_capture(tracker, sensor_capture,
K4A_WAIT_INFINITE);
k4a_capture_release(sensor_capture); // Remember to release the sensor capture once you finish using it
if (queue_capture_result == K4A_WAIT_RESULT_FAILED)
{
    printf("Error! Adding capture to tracker process queue failed!\n");
    break;
}

k4abt_frame_t body_frame = NULL;
k4a_wait_result_t pop_frame_result = k4abt_tracker_pop_result(tracker, &body_frame, K4A_WAIT_INFINITE);
if (pop_frame_result == K4A_WAIT_RESULT_SUCCEEDED)
{
    // Successfully popped the body tracking result. Start your processing
    ...

    k4abt_frame_release(body_frame); // Remember to release the body frame once you finish using it
}
```

## 访问人体跟踪结果数据

每个传感器捕获的人体跟踪结果存储在人体帧 `k4abt_frame_t` 结构中。每个人体帧包含三个重要组成部分：人体结构的集合、2D 人体索引映射和输入捕获。

第一个人体跟踪应用程序只访问检测到人体数。有关人体帧中的数据的详细说明，请参阅[访问人体帧中的数据](#)。

```
size_t num_bodies = k4abt_frame_get_num_bodies(body_frame);
printf("%zu bodies are detected!\n", num_bodies);
```

## 清除

最后一步是关闭人体跟踪器并释放人体跟踪对象。此外，还需要停止并关闭设备。

```
k4abt_tracker_shutdown(tracker);
k4abt_tracker_destroy(tracker);
k4a_device_stop_cameras(device);
k4a_device_close(device);
```

## 完整源代码

```
#include <stdio.h>
#include <stdlib.h>

#include <k4a/k4a.h>
#include <k4abt.h>

#define VERIFY(result, error)
    if(result != K4A_RESULT_SUCCEEDED)
    {
        printf("%s \n - (File: %s, Function: %s, Line: %d)\n", error, __FILE__, __FUNCTION__, __LINE__);
        exit(1);
    }

int main()
{
    k4a_device_t device = NULL;
    VERIFY(k4a_device_open(0, &device), "Open K4A Device failed!");

    // Start camera. Make sure depth camera is enabled.
    k4a_device_configuration_t deviceConfig = K4A_DEVICE_CONFIG_INIT_DISABLE_ALL;
    deviceConfig.depth_mode = K4A_DEPTH_MODE_NFOV_UNBINNED;
    deviceConfig.color_resolution = K4A_COLOR_RESOLUTION_OFF;
    VERIFY(k4a_device_start_cameras(device, &deviceConfig), "Start K4A cameras failed!");

    k4a_calibration_t sensor_calibration;
    VERIFY(k4a_device_get_calibration(device, deviceConfig.depth_mode, deviceConfig.color_resolution,
&sensor_calibration),
        "Get depth camera calibration failed!");

    k4abt_tracker_t tracker = NULL;
    k4abt_tracker_configuration_t tracker_config = K4ABT_TRACKER_CONFIG_DEFAULT;
    VERIFY(k4abt_tracker_create(&sensor_calibration, tracker_config, &tracker), "Body tracker initialization
failed!");

    int frame_count = 0;
    do
    {
        k4a_capture_t sensor_capture;
        k4a_wait_result_t get_capture_result = k4a_device_get_capture(device, &sensor_capture,
K4A_WAIT_INFINITE);
```

```

        if (get_capture_result == K4A_WAIT_RESULT_SUCCEEDED)
        {
            frame_count++;
            k4a_wait_result_t queue_capture_result = k4abt_tracker_enqueue_capture(tracker, sensor_capture,
K4A_WAIT_INFINITE);
            k4a_capture_release(sensor_capture); // Remember to release the sensor capture once you finish
using it
            if (queue_capture_result == K4A_WAIT_RESULT_TIMEOUT)
            {
                // It should never hit timeout when K4A_WAIT_INFINITE is set.
                printf("Error! Add capture to tracker process queue timeout!\n");
                break;
            }
            else if (queue_capture_result == K4A_WAIT_RESULT_FAILED)
            {
                printf("Error! Add capture to tracker process queue failed!\n");
                break;
            }

            k4abt_frame_t body_frame = NULL;
            k4a_wait_result_t pop_frame_result = k4abt_tracker_pop_result(tracker, &body_frame,
K4A_WAIT_INFINITE);
            if (pop_frame_result == K4A_WAIT_RESULT_SUCCEEDED)
            {
                // Successfully popped the body tracking result. Start your processing

                size_t num_bodies = k4abt_frame_get_num_bodies(body_frame);
                printf("%zu bodies are detected!\n", num_bodies);

                k4abt_frame_release(body_frame); // Remember to release the body frame once you finish using
it
            }
            else if (pop_frame_result == K4A_WAIT_RESULT_TIMEOUT)
            {
                // It should never hit timeout when K4A_WAIT_INFINITE is set.
                printf("Error! Pop body frame result timeout!\n");
                break;
            }
            else
            {
                printf("Pop body frame result failed!\n");
                break;
            }
        }
        else if (get_capture_result == K4A_WAIT_RESULT_TIMEOUT)
        {
            // It should never hit time out when K4A_WAIT_INFINITE is set.
            printf("Error! Get depth frame time out!\n");
            break;
        }
        else
        {
            printf("Get depth capture returned error: %d\n", get_capture_result);
            break;
        }
    }

} while (frame_count < 100);

printf("Finished body tracking processing!\n");

k4abt_tracker_shutdown(tracker);
k4abt_tracker_destroy(tracker);
k4a_device_stop_cameras(device);
k4a_device_close(device);

return 0;
}

```

## 后续步骤

[获取人体跟踪结果](#)

# Azure Kinect DK 深度相机

2020/8/11 • [Edit Online](#)

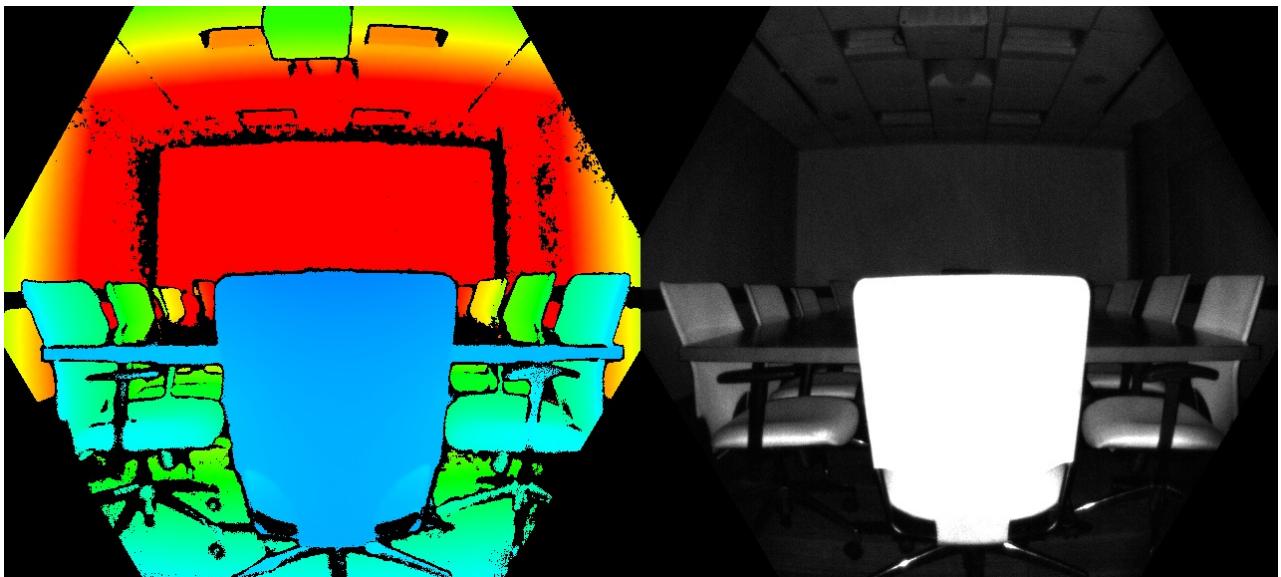
本页介绍如何在 Azure Kinect DK 中使用深度相机。深度相机是两个相机中的第二个。如前面的部分中所述，另一个相机是 RGB 相机。

## 工作原理

Azure Kinect DK 深度相机实现调幅连续波 (AMCW) 时差测距 (ToF) 原理。该相机将近红外 (NIR) 频谱中的调制光投射到场景中。然后，它会记录光线从相机传播到场景，然后从场景返回到相机所花费的间接时间测量值。

处理这些测量值可以生成深度图。深度图是图像每个像素的一组 Z 坐标值，以毫米为单位。

连同深度图一起，我们还可以获得所谓的清晰 IR 读数。清晰 IR 读数中的像素值与从场景返回的光线量成正比。图像类似于普通的 IR 图像。下图显示了示例深度图(左)的对应的清晰 IR 图像(右)。



## 主要功能

深度相机的技术特征包括：

- 配备高级像素技术的 1 兆像素 ToF 成像芯片，实现更高的调制频率和深度精度。
- 两个 NIR 激光二极管实现近距和宽视场 (FoV) 深度模式。
- 全球最小的  $3.5\mu\text{m} \times 3.5\mu\text{m}$  ToF 像素。
- 自动像素增益选择支持较大的动态范围，允许捕获清晰的近距和远距对象。
- 全局快门可帮助改善日光下的拍摄性能。
- 多相位深度计算方法能够实现可靠的准确度，即使芯片、激光和电源存在差异。
- 较低的系统误差和随机误差。



深度相机将原始的调制 IR 图像传输到电脑主机。在电脑上, GPU 加速的深度引擎软件会将原始信号转换为深度图。深度相机支持多种模式。窄视场 (FoV) 模式非常适合 X、Y 维度范围较小, 但 Z 维度范围较大的场景。如果场景中的 X、Y 范围较大, 但 Z 范围较小, 则宽 FoV 模式更合适。

深度相机支持 2x2 装箱模式, 这种模式与非装箱模式相比, 可以扩展 Z 范围。装箱的代价是降低图像分辨率。所有模式都能够以高达 30 帧/秒 (fps) 的速率运行, 但 1 兆象素 (MP) 模式除外, 它的最大运行帧速率为 15 fps。深度相机还提供被动 IR 模式。在此模式下, 照像机上的照明器不会激活, 只能观测到环境光。

## 相机性能

相机的性能以系统误差和随机误差来衡量。

### 系统误差

系统误差定义为消噪后测得的深度与正确(真实)深度之差。我们会根据静态场景的许多帧计算时态平均值, 以消除尽可能多的深度噪声。更确切地说, 系统误差定义为:

$$E_{systematic} = \frac{\sum_{t=1}^N d_t}{N} - d_{gt}$$

其中,  $d_t$  表示时间  $t$  处的测量深度,  $N$  是求平均过程使用的帧数,  $d_{gt}$  是真实深度。

深度相机的系统误差规范不包括多路径干扰 (MPI)。当某个传感器像素集成了多个对象反射的光时, 就会发生 MPI。使用较高的调制频率以及稍后将会介绍的深度失效可以在深度相机中部分缓解 MPI。

### 随机误差

假设我们在没有移动相机的情况下拍摄了同一对象的 100 张图像。在这 100 张图像中, 每张图像的对象深度略有不同。这种差异是散粒噪声造成的。发生散粒噪声的原因是, 在一段时间内, 进入传感器的光子数因某种随机因素而有变化。我们将静态场景中的这种随机误差定义为一段时间内的深度标准偏差, 其计算公式为:

$$E_{random} = \sqrt{\frac{\sum_{t=1}^N (d_t - \bar{d})^2}{N}}$$

其中,  $N$  表示深度测量值数,  $d_t$  表示时间  $t$  处的深度测量值,  $\bar{d}$  表示基于所有深度测量值  $d_t$  计算出的平均值。

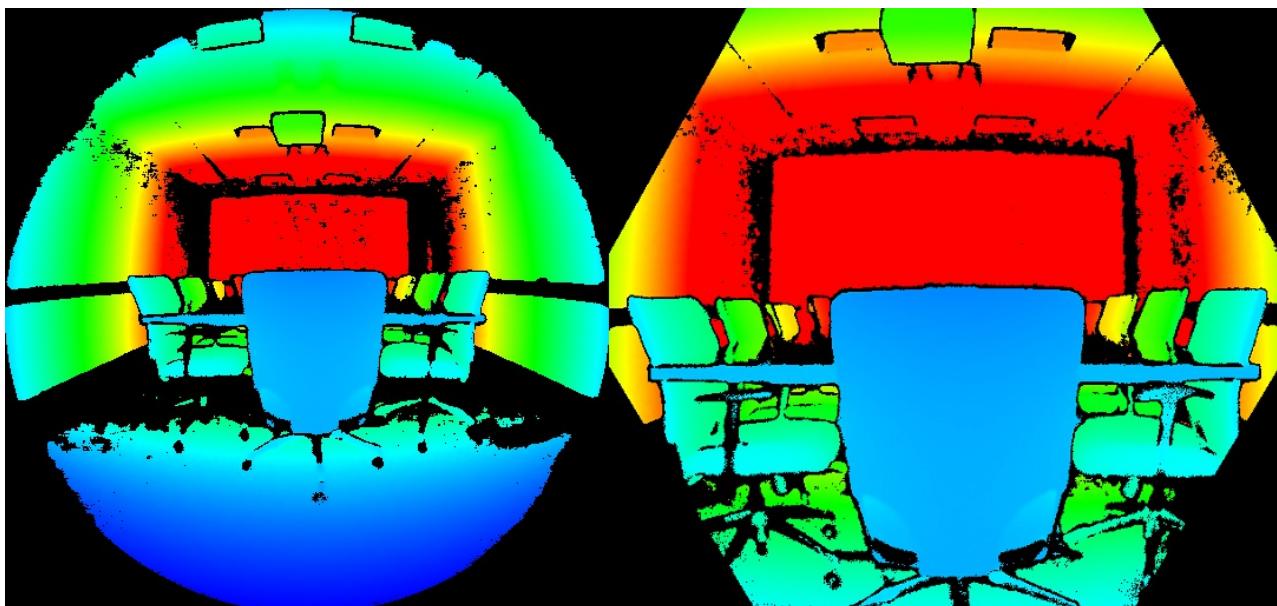
## 失效

在某些情况下, 深度相机可能无法提供某些像素的正确值。在这种情况下, 深度像素将会失效。无效的像素由深度值 0 表示。深度引擎无法生成正确值的原因包括:

- 超出活动 IR 照明遮罩范围
- IR 信号饱和
- IR 信号强度低
- 滤波异常
- 多路径干扰

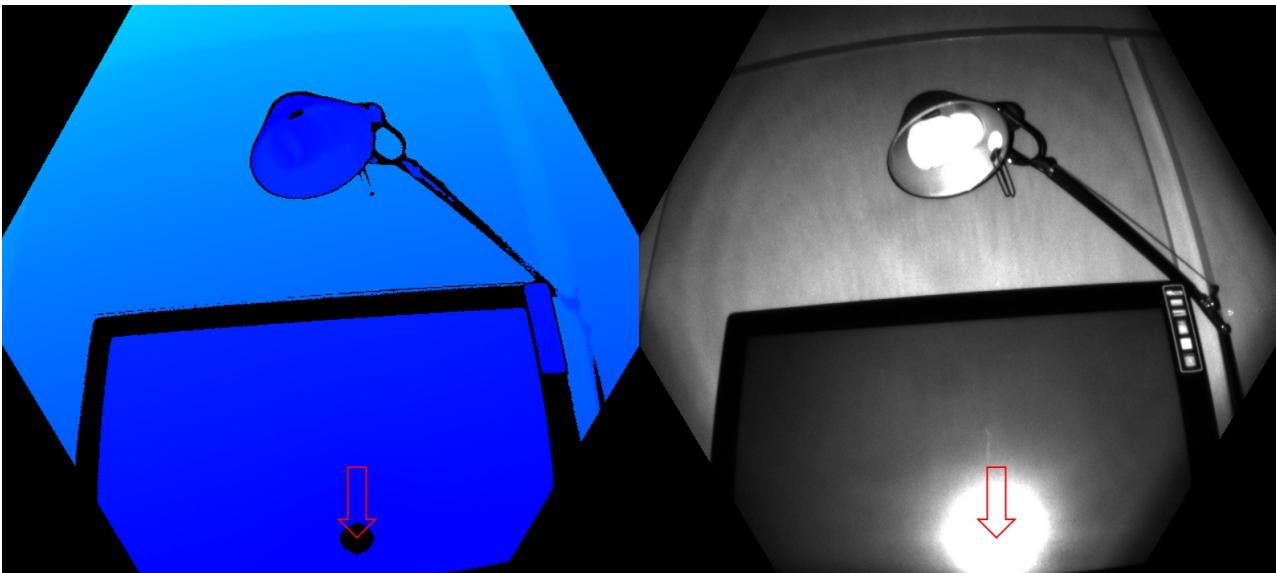
### 照明遮罩

当像素超出活动 IR 光照的遮罩范围时, 它们将会失效。我们不建议使用此类像素的信号来计算深度。下图显示了超出照明遮罩范围而导致像素失效的示例。失效的像素包括宽 FoV 模式的圆圈(左)和窄 FoV 模式的六边形(右)外部的黑色像素。

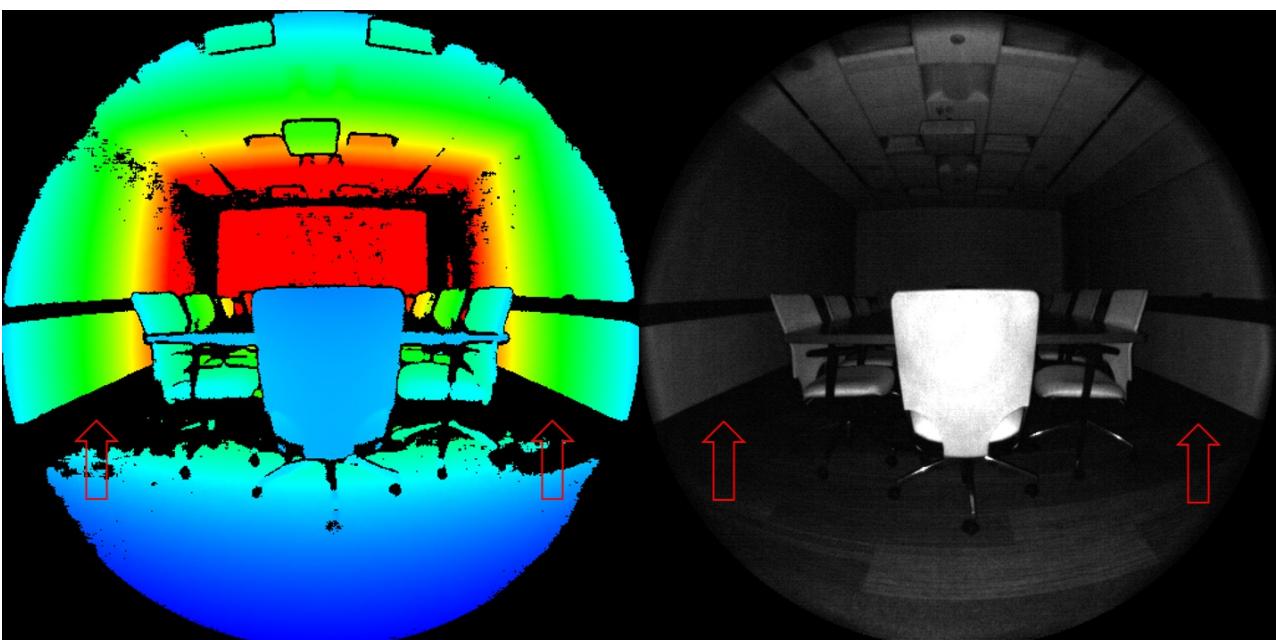


### 信号强度

当像素包含饱和的 IR 信号时, 它们将会失效。像素饱和后, 相位信息将会丢失。下图显示了 IR 信号饱和导致像素失效的示例。请查看指向深度图像和 IR 图像中的示例像素的箭头。



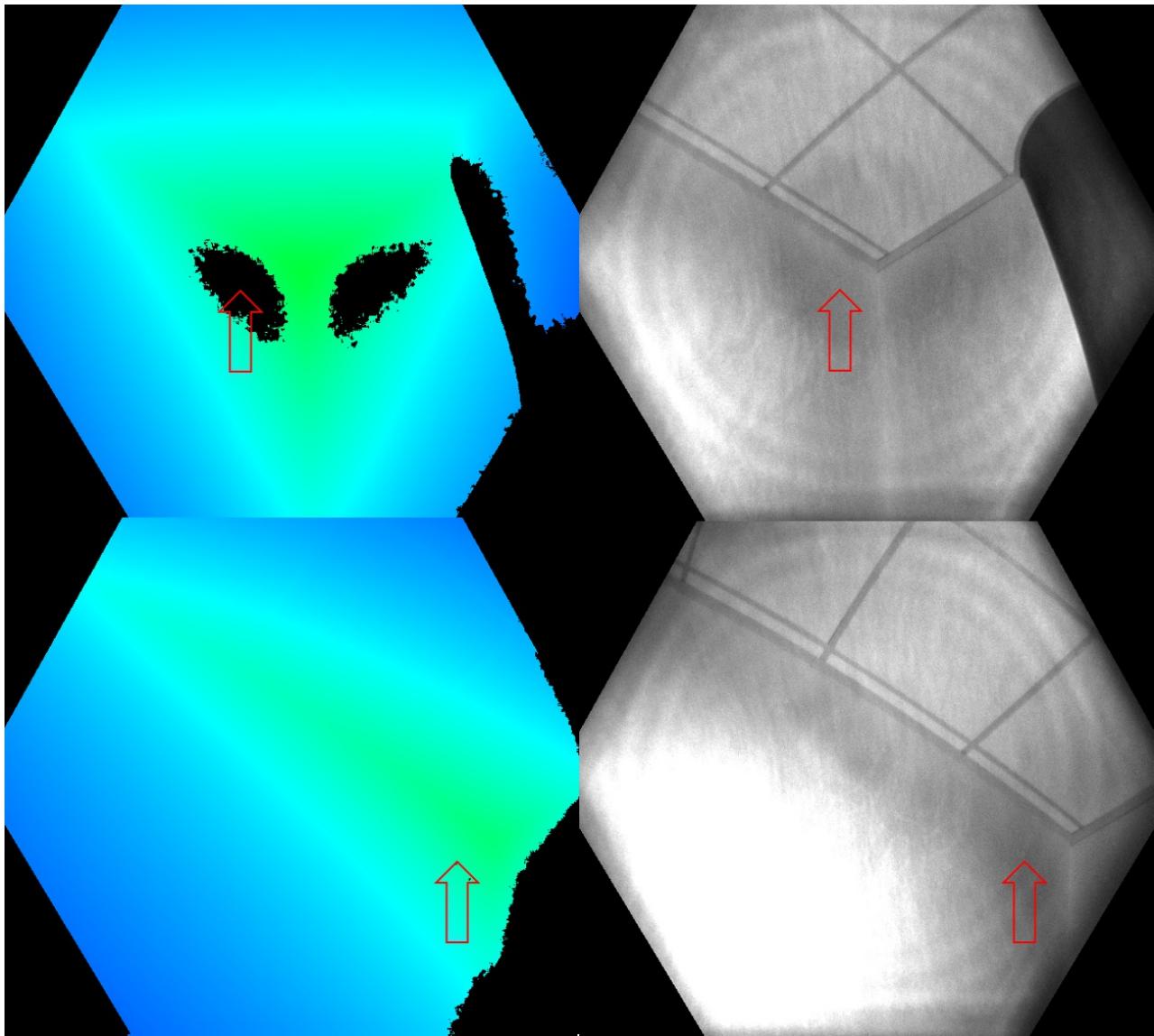
如果 IR 信号强度不够，以致无法生成深度，则也可能会发生像素失效。下图显示了 IR 信号强度低导致像素失效的示例。请查看指向深度图像和 IR 图像中的示例像素的箭头。



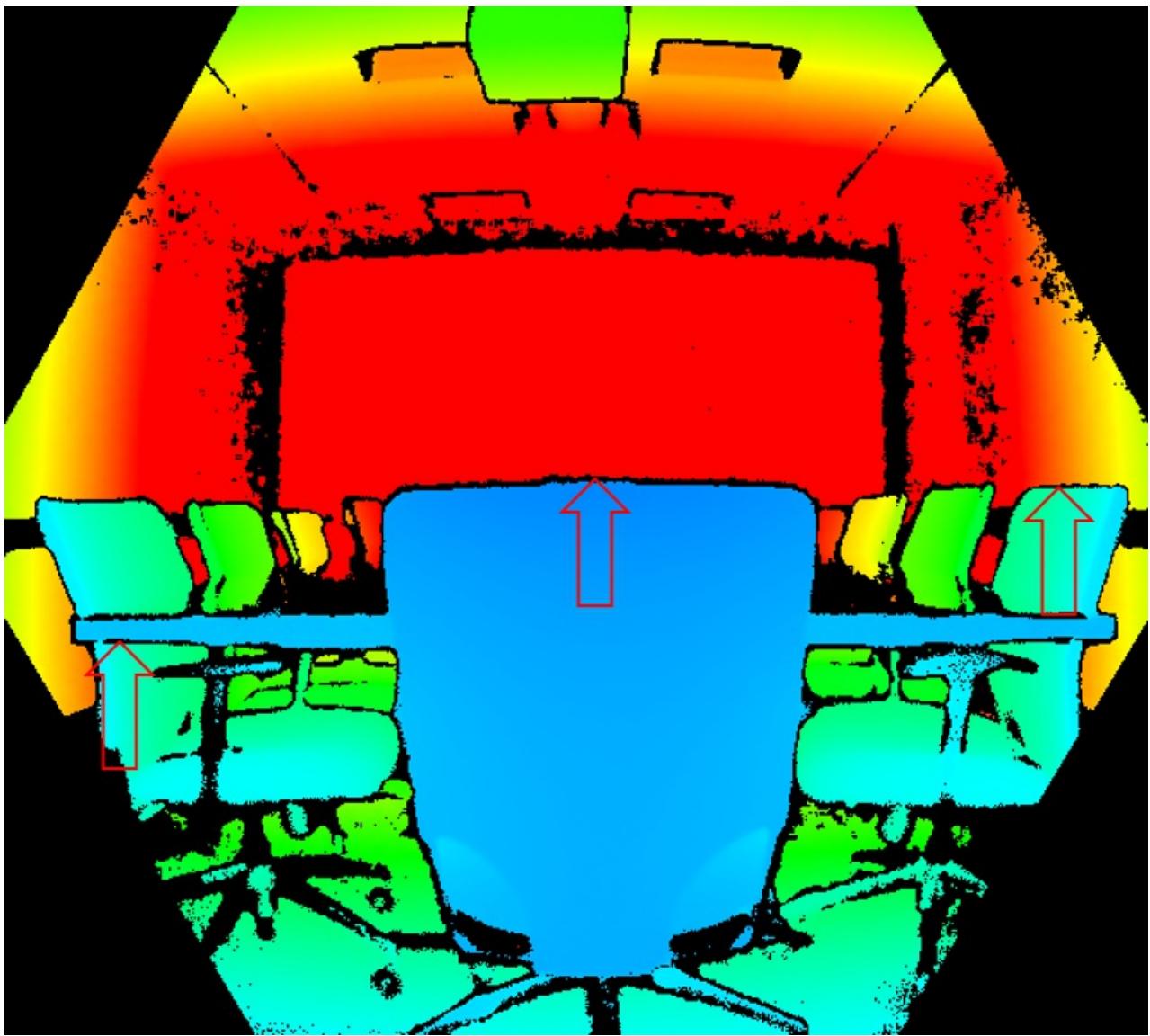
### 歧义深度

如果像素从场景中的多个对象收到了信号，则它们也可能会失效。在角落中经常会看到这种像素失效的情况。由于场景的几何结构，照相发出的 IR 光会从一堵墙反射到另一堵墙。这种反射光会导致测得的像素深度出现歧义。深度算法中的滤波器会检测这些有歧义的信号，并使像素失效。

下图显示了多路径检测导致像素失效的示例。你还可能会发现，在一个相机视图中失效的表面区域（上排）可能会在不同的相机视图中重新出现（下排）。此图演示在一个透视图中失效的表面可能会在另一个透视图中显示。



多路径的另一种常见情况是像素包含来自前景和背景的混合信号(例如围绕对象边缘)。在快动作场景中,你可能会看到边缘的四周有更多失效的像素。更多像素失效的原因在于原始深度捕获的曝光间隔。



后续步骤

坐标系

# Azure Kinect DK 坐标系

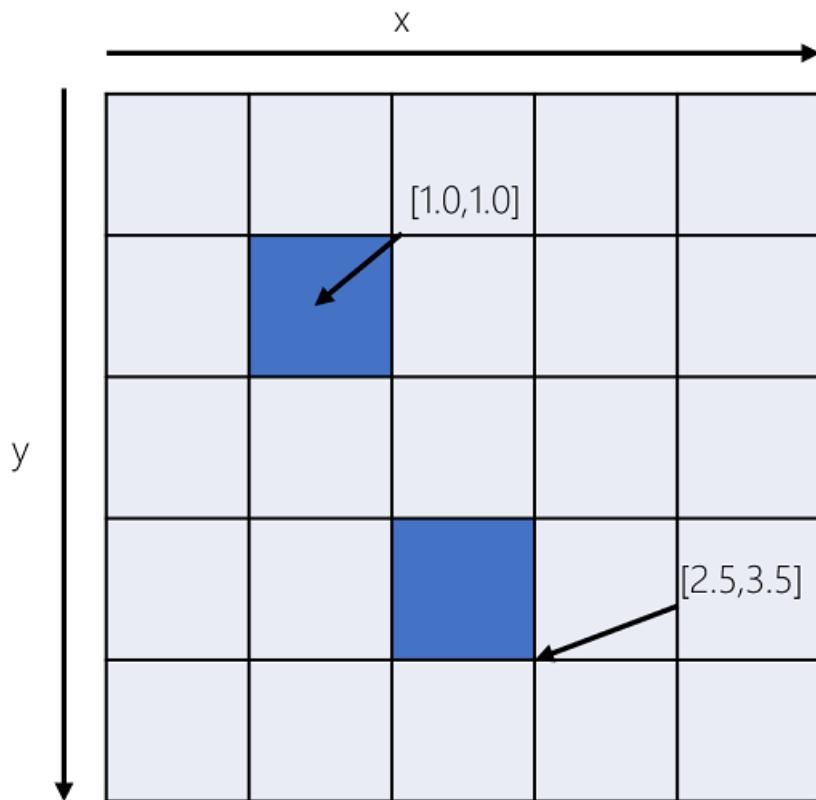
2020/8/11 • [Edit Online](#)

本文介绍用于 2D 和 3D 坐标系的约定。每个传感器设备有关联的独立坐标系，使用[校准函数](#)可以转换这些坐标系之间的点。[转换函数](#)转换坐标系之间的整个图像。

## 2D 坐标系

深度相机和彩色相机与独立的 2D 坐标系相关联。[x,y] 坐标以像素为单位，其中， $x$  的范围为 0 到宽度-1， $y$  的范围为 0 到高度-1。宽度和高度取决于所选的深度相机和彩色相机工作模式。像素坐标  $[0,0]$  对应于图像的左上角像素。像素坐标可以是表示子像素坐标的小数。

2D 坐标系以 0 为中心，即，子像素坐标  $[0.0, 0.0]$  表示中心， $[0.5,0.5]$  表示像素的右下角，如下所示。



## 3D 坐标系

每个相机、加速度传感器和陀螺仪都与独立的 3D 坐标空间系统相关联。

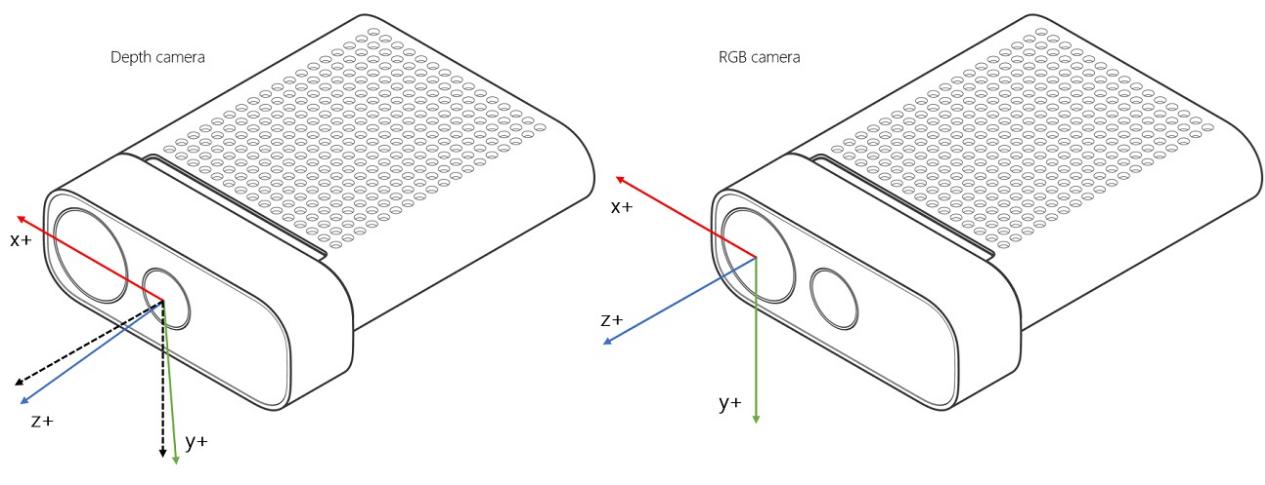
3D 坐标系中的点以指标 [X,Y,Z] 坐标三元组的形式表示，单位为毫米。

### 深度相机和彩色相机

原点  $[0,0,0]$  位于相机焦点处。坐标系的方向是正 X 轴向右，正 Y 轴向下，正 Z 轴向上。

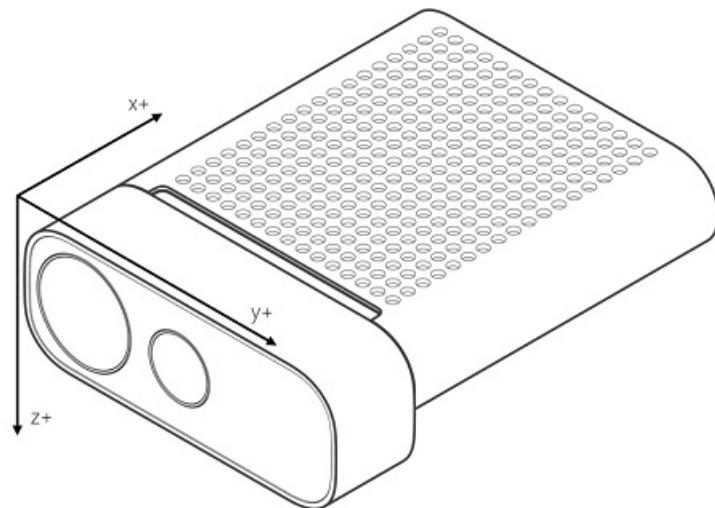
深度相机向下朝彩色相机倾斜 6 度，如下所示。

深度相机使用两个照明器。在窄视场 (NFOV) 模式下使用的照明器与深度相机的用例相符，因此照明器不倾斜。在宽视场 (WFOV) 模式下使用的照明器相对于深度相机向下额外倾斜 1.3 度。



## 陀螺仪和加速度传感器

陀螺仪的原点  $[0,0,0]$  与深度相机的原点相同。加速度传感器的原点与其物理位置相一致。加速度传感器和陀螺仪的坐标系都位于右侧。坐标系的正 X 轴向后，正 Y 轴向左，正 Z 轴向下，如下所示。



## 后续步骤

[了解 Azure Kinect 传感器 SDK](#)

# Azure Kinect 人体跟踪关节

2020/8/11 • [Edit Online](#)

Azure Kinect 人体跟踪可以同时跟踪多个人体。每个人体包括一个 ID, 用于在帧与运动骨架之间进行临时关联。可以使用 `k4abt_frame_get_num_bodies()` 获取在每个帧中检测到的人体数。

## 关节

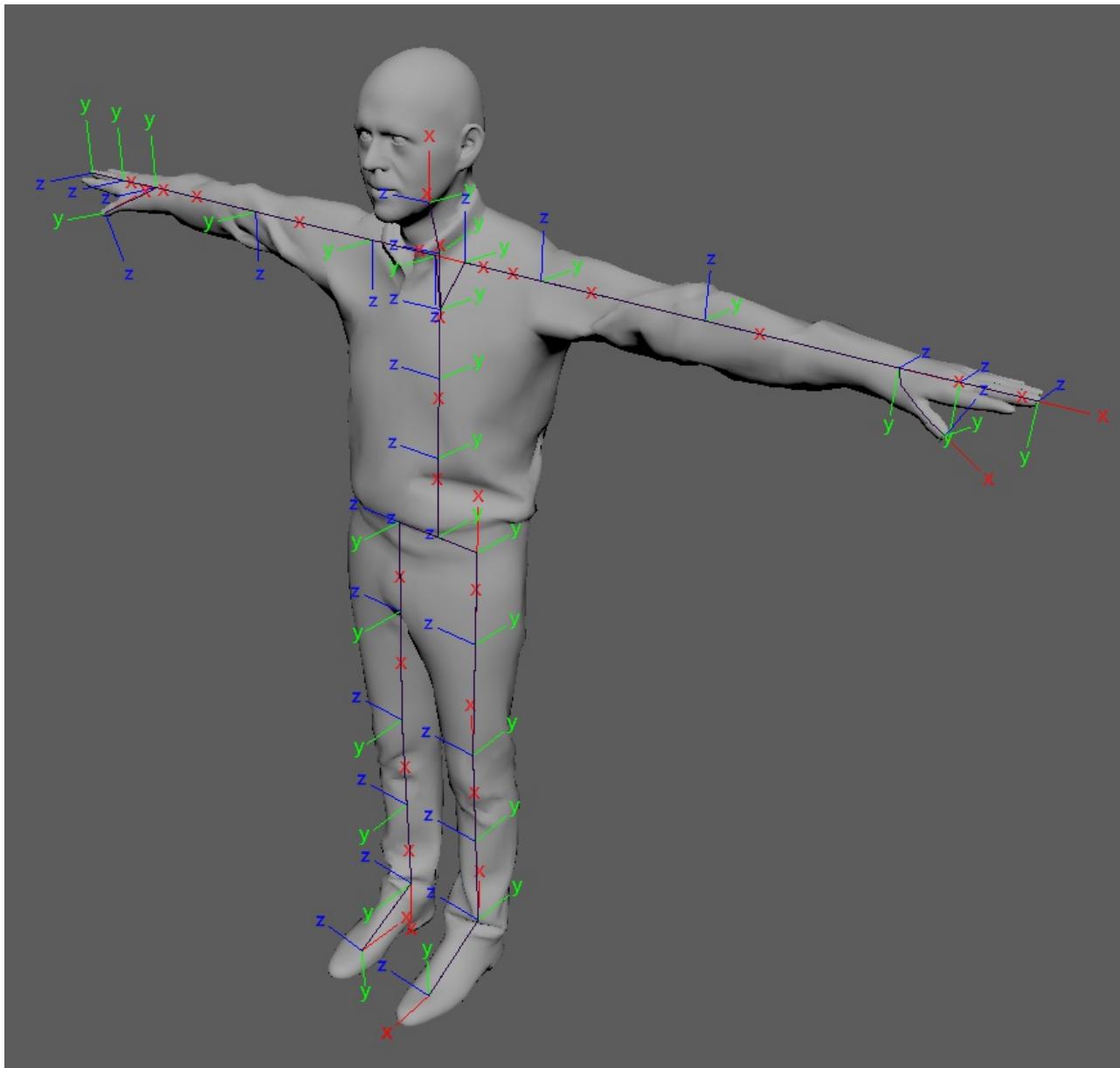
关节位置和方向是相对于全局深度传感器参考帧的估算值。位置以毫米为单位指定。方向以规范化四元数表示。

## 关节坐标

每个关节的位置和方向构成了其自身的关节坐标系。所有关节坐标系是相对于深度摄像头 3D 坐标系的绝对坐标系。

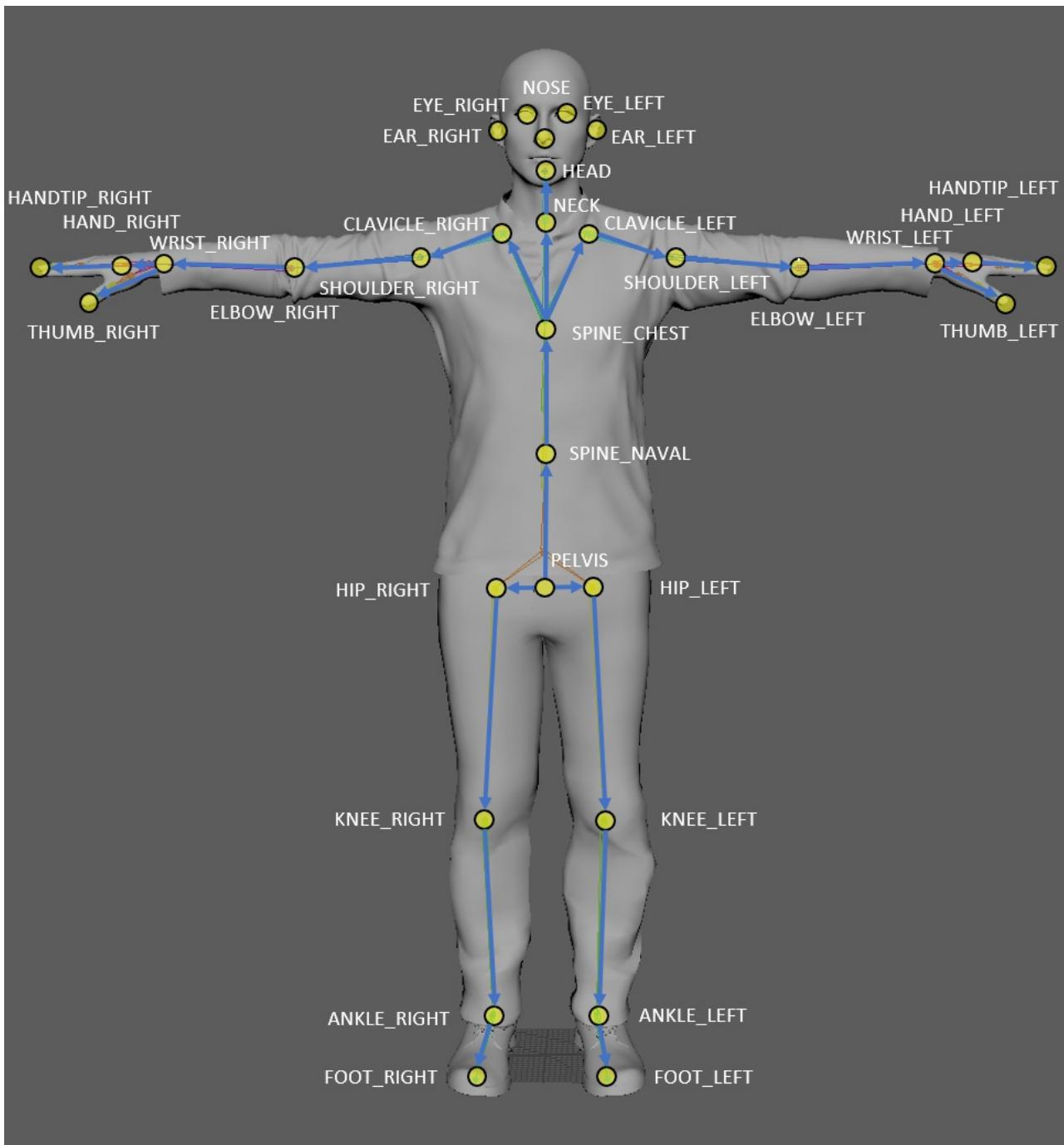
### NOTE

关节坐标是轴方向上的坐标。轴方向广泛用于商业头像、游戏引擎和渲染软件。使用轴方向可以简化镜像运动, 例如将双臂抬起 20 度。



## 关节层次结构

骨架包括 32 个关节，关节层次结构按照从人体中心向四肢的流向分布。每个连接(骨骼)将父关节与子关节链接起来。下图演示了人体上的关节位置和连接方式。



下表列举了标准的关节连接。

II	III	III
0	PELVIS	-
1	SPINE_NAVAL	PELVIS
2	SPINE_CHEST	SPINE_NAVAL
3	NECK	SPINE_CHEST
4	CLAVICLE_LEFT	SPINE_CHEST
5	SHOULDER_LEFT	CLAVICLE_LEFT
6	ELBOW_LEFT	SHOULDER_LEFT

7	WRIST_LEFT	ELBOW_LEFT
8	HAND_LEFT	WRIST_LEFT
9	HANDTIP_LEFT	HAND_LEFT
10	THUMB_LEFT	WRIST_LEFT
11	CLAVICLE_RIGHT	SPINE_CHEST
12	SHOULDER_RIGHT	CLAVICLE_RIGHT
13	ELBOW_RIGHT	SHOULDER_RIGHT
14	WRIST_RIGHT	ELBOW_RIGHT
15	HAND_RIGHT	WRIST_RIGHT
16	HANDTIP_RIGHT	HAND_RIGHT
17	THUMB_RIGHT	WRIST_RIGHT
18	HIP_LEFT	PELVIS
19	KNEE_LEFT	HIP_LEFT
20	ANKLE_LEFT	KNEE_LEFT
21	FOOT_LEFT	ANKLE_LEFT
22	HIP_RIGHT	PELVIS
23	KNEE_RIGHT	HIP_RIGHT
24	ANKLE_RIGHT	KNEE_RIGHT
25	FOOT_RIGHT	ANKLE_RIGHT
26	HEAD	NECK
27	NOSE	HEAD
28	EYE_LEFT	HEAD
29	EAR_LEFT	HEAD
30	EYE_RIGHT	HEAD
31	EAR_RIGHT	HEAD

## 后续步骤

[人体跟踪索引图](#)

# Azure Kinect 人体跟踪索引映射

2020/8/11 • [Edit Online](#)

人体索引映射包括深度机捕捉中每个人体的实例分段图。每个像素映射到深度或 IR 图像中的相应像素。每个像素的值表示该像素所属的人体。它可以是背景(值 `K4ABT_BODY_INDEX_MAP_BACKGROUND`)或检测到的 `k4abt_body_t` 的索引。



## NOTE

人体索引不同于人体 ID。可以调用 API `k4abt_frame_get_body_id()` 从给定的人体索引中查询人体 ID。

## 使用人体索引映射

人体索引映射存储为 `k4a_image_t`，使用的分辨率与深度或 IR 图像相同。每个像素都是一个 8 位值。可以调用 `k4abt_frame_get_body_index_map` 从 `k4abt_frame_t` 中查询它。开发人员负责通过调用 `k4a_image_release()` 释放人体索引映射的内存。

## 后续步骤

[生成第一个人体跟踪应用](#)

# 关于 Azure Kinect 传感器 SDK

2020/8/11 • [Edit Online](#)

本文概述 Azure Kinect 传感器软件开发工具包 (SDK) 及其功能和工具。

## 功能

Azure Kinect 传感器 SDK 提供对 Azure Kinect 设备配置和硬件传感器流的跨平台低级访问，包括：

- 深度相机访问和模式控制(被动 IR 模式, 以及宽视场和窄视场深度模式)
- RGB 相机访问和控制(例如曝光和白平衡)
- 运动传感器(陀螺仪和加速度传感器)访问
- 同步的深度 RGB 相机流, 相机之间的延迟可配置
- 外部设备同步控制, 设备之间的延迟偏移量可配置
- 用于处理图像分辨率、时间戳等的相机帧元数据访问。
- 设备校准数据访问

## 工具

- 用于监视设备数据流和配置不同模式的 [Azure Kinect 查看器](#)。
- 使用 Matroska 容器格式的 [Azure Kinect 录制器](#)和播放读取器 API。
- Azure Kinect DK [固件更新工具](#)。

## 传感器 SDK

- [下载传感器 SDK](#)。
- [GitHub 上已提供传感器 SDK 的开放源代码](#)。
- 有关用法详细信息, 请参阅[传感器 SDK API 文档](#)。

## 后续步骤

你现已了解 Azure Kinect 传感器 SDK, 接下来还可以：

[下载传感器 SDK 代码](#)

[查找和打开设备](#)

# 找到并打开 Azure Kinect 设备

2020/8/11 • [Edit Online](#)

本文介绍如何找到然后打开 Azure Kinect DK。本文将解释如何处理有多个设备连接到计算机的情况。

你还可以参考 [SDK 枚举示例](#)，其中演示了如何使用本文所述的函数。

本文将介绍以下函数：

- [k4a\\_device\\_get\\_installed\\_count\(\)](#)
- [k4a\\_device\\_open\(\)](#)
- [k4a\\_device\\_get\\_serialnum\(\)](#)
- [k4a\\_device\\_close\(\)](#)

## 发现已连接的设备数

首先使用 [k4a\\_device\\_get\\_installed\\_count\(\)](#) 获取当前已连接的 Azure Kinect 设备数。

```
uint32_t device_count = k4a_device_get_installed_count();

printf("Found %d connected devices:\n", device_count);
```

## 打开设备

若要获取设备的相关信息或从中读取数据，首先需要使用 [k4a\\_device\\_open\(\)](#) 打开该设备的句柄。

```
k4a_device_t device = NULL;

for (uint8_t deviceIndex = 0; deviceIndex < device_count; deviceIndex++)
{
    if (K4A_RESULT_SUCCEEDED != k4a_device_open(deviceIndex, &device))
    {
        printf("%d: Failed to open device\n", deviceIndex);
        continue;
    }

    ...

    k4a_device_close(device);
}
```

`index` [k4a\\_device\\_open\(\)](#) 的参数指示当连接了多个设备时要打开哪个设备。如果你预期只会连接一个设备，可以传递 [K4A\\_DEVICE\\_DEFAULT](#) 的参数或 0 来指示第一台设备。

用完句柄后，每当打开设备时，都需要调用 [k4a\\_device\\_close\(\)](#)。在关闭句柄之前，无法打开同一设备的其他句柄。

## 识别特定的设备

在附加或分离设备之前，设备按索引枚举的顺序不会更改。若要识别物理设备，应使用设备的序列号。

若要读取设备中的序列号，请在打开句柄后使用 [k4a\\_device\\_get\\_serialnum\(\)](#) 函数。

此示例演示如何分配适量内存来存储序列号。

```
char *serial_number = NULL;
size_t serial_number_length = 0;

if (K4A_BUFFER_RESULT_TOO_SMALL != k4a_device_get_serialnum(device, NULL, &serial_number_length))
{
    printf("%d: Failed to get serial number length\n", deviceIndex);
    k4a_device_close(device);
    device = NULL;
    continue;
}

serial_number = malloc(serial_number_length);
if (serial_number == NULL)
{
    printf("%d: Failed to allocate memory for serial number (%zu bytes)\n", deviceIndex,
serial_number_length);
    k4a_device_close(device);
    device = NULL;
    continue;
}

if (K4A_BUFFER_RESULT_SUCCEEDED != k4a_device_get_serialnum(device, serial_number, &serial_number_length))
{
    printf("%d: Failed to get serial number\n", deviceIndex);
    free(serial_number);
    serial_number = NULL;
    k4a_device_close(device);
    device = NULL;
    continue;
}

printf("%d: Device \"%s\"\n", deviceIndex, serial_number);
```

## 打开默认设备

在大多数应用程序中，只会将单个 Azure Kinect DK 附加到同一台计算机。如果只需连接到单个预期设备，可以结合 `K4A_DEVICE_DEFAULT` 的 `index` 调用 `k4a_device_open()` 打开第一台设备。

```
k4a_device_t device = NULL;
uint32_t device_count = k4a_device_get_installed_count();

if (device_count != 1)
{
    printf("Unexpected number of devices found (%d)\n", device_count);
    goto Exit;
}

if (K4A_RESULT_SUCCEEDED != k4a_device_open(K4A_DEVICE_DEFAULT, &device))
{
    printf("Failed to open device\n");
    goto Exit;
}
```

## 后续步骤

[检索图像](#)

[检索 IMU 样本](#)

# 检索 Azure Kinect 图像数据

2020/8/11 • [Edit Online](#)

本页提供有关如何通过 Azure Kinect 检索图像的详细信息。本文演示如何捕获和访问设备颜色和深度之间的协调图像。若要访问图像，必须先打开并配置设备，然后可以捕获图像。在配置和捕获图像之前，必须找到并打开设备。

你还可以参考 [SDK 流示例](#)，其中演示了如何使用本文所述的函数。

本文将介绍以下函数：

- [k4a\\_device\\_start\\_cameras\(\)](#)
- [k4a\\_device\\_get\\_capture\(\)](#)
- [k4a\\_capture\\_get\\_depth\\_image\(\)](#)
- [k4a\\_image\\_get\\_buffer\(\)](#)
- [k4a\\_image\\_release\(\)](#)
- [k4a\\_capture\\_release\(\)](#)
- [k4a\\_device\\_stop\\_cameras\(\)](#)

## 配置并启动设备

Kinect 设备上提供的两个相机支持多种模式、分辨率和输出格式。有关完整列表，请参考 Azure Kinect 开发工具包[硬件规格](#)。

流配置是使用 [k4a\\_device\\_configuration\\_t](#) 结构中的值设置的。

```
k4a_device_configuration_t config = K4A_DEVICE_CONFIG_INIT_DISABLE_ALL;
config.camera_fps = K4A_FRAMES_PER_SECOND_30;
config.color_format = K4A_IMAGE_FORMAT_COLOR_MJPG;
config.color_resolution = K4A_COLOR_RESOLUTION_2160P;
config.depth_mode = K4A_DEPTH_MODE_NFOV_UNBINNED;

if (K4A_RESULT_SUCCEEDED != k4a_device_start_cameras(device, &config))
{
    printf("Failed to start device\n");
    goto Exit;
}
```

相机启动后，它们将不断捕获数据，直到已调用 [k4a\\_device\\_stop\\_cameras\(\)](#) 或设备关闭。

## 稳定性

使用多设备同步功能启动设备时，我们强烈建议使用固定的曝光设置。如果设置手动曝光，最多可能需要在设备上拍摄 8 次，图像和帧速率就会才能稳定。如果使用自动曝光，最多可能需要拍摄 20 次，图像和帧速率才能稳定。

## 从设备获取捕获

图像是以关联的方式从设备捕获的。捕获的每个图像包含深度图像、IR 图像、彩色图像或图像的组合。

默认情况下，API 只会在收到流模式请求的所有图像后才返回捕获。可以通过清除 [k4a\\_device\\_configuration\\_t](#) 的 [synchronized\\_images\\_only](#) 参数，将 API 配置为在深度图像和彩色图像可用后，立即返回仅包含这些图像的部分捕获。

获。

```
// Capture a depth frame
switch (k4a_device_get_capture(device, &capture, TIMEOUT_IN_MS))
{
case K4A_WAIT_RESULT_SUCCEEDED:
    break;
case K4A_WAIT_RESULT_TIMEOUT:
    printf("Timed out waiting for a capture\n");
    continue;
    break;
case K4A_WAIT_RESULT_FAILED:
    printf("Failed to read a capture\n");
    goto Exit;
}
```

在 API 成功返回捕获后，当你用完捕获对象时，必须调用 [k4a\\_capture\\_release\(\)](#)。

## 从捕获中获取图像

若要检索捕获的图像，请针对每个图像类型调用相应的函数。可取值为：

- [k4a\\_capture\\_get\\_color\\_image\(\)](#)
- [k4a\\_capture\\_get\\_depth\\_image\(\)](#)
- [k4a\\_capture\\_get\\_ir\\_image\(\)](#)

用完图像后，必须对这些函数返回的任何 [k4a\\_image\\_t](#) 句柄调用 [k4a\\_image\\_release\(\)](#)。

## 访问图像缓冲区

[k4a\\_image\\_t](#) 提供多个访问器函数用于获取图像的属性。

若要访问图像的内存缓冲区，请使用 [k4a\\_image\\_get\\_buffer](#)。

以下示例演示如何访问捕获的深度图像。相同的原则也适用于其他图像类型。但是，请务必将图像类型变量替换为正确的图像类型，例如 IR 或彩色。

```
// Access the depth16 image
k4a_image_t image = k4a_capture_get_depth_image(capture);
if (image != NULL)
{
    printf(" | Depth16 res:%4dx%4d stride:%5d\n",
           k4a_image_get_height_pixels(image),
           k4a_image_get_width_pixels(image),
           k4a_image_get_stride_bytes(image));

    // Release the image
    k4a_image_release(image);
}

// Release the capture
k4a_capture_release(capture);
```

## 后续步骤

既然你已了解如何捕获和协调相机的彩色图像与深度图像，现在可以使用 Azure Kinect 设备。你还可以：

[检索 IMU 样本](#)

[访问麦克风](#)



# 检索 Azure Kinect IMU 样本

2020/8/11 • [Edit Online](#)

在 Azure Kinect 设备中可以访问惯性运动单元 (IMU)，其中包括加速度传感器和陀螺仪类型。若要访问 IMU 样本，必须先打开并配置设备，然后捕获 IMU 数据。有关详细信息，请参阅[查找和打开设备](#)。

生成 IMU 样本的频率要比图像高得多。向主机报告样本的频率低于采样频率。在等待接收 IMU 样本时，往往已经有多个可用的样本。

有关 IMU 报告频率的详细信息，请参阅 Azure Kinect DK [硬件规格](#)。

## 配置并启动相机

### NOTE

仅当彩色和/或深度相机正在运行时，IMU 传感器才能正常工作。IMU 传感器不能单独工作。

若要启动相机，请使用 [k4a\\_device\\_start\\_cameras\(\)](#)。

```
k4a_device_configuration_t config = K4A_DEVICE_CONFIG_INIT_DISABLE_ALL;
config.camera_fps = K4A_FRAMES_PER_SECOND_30;
config.color_format = K4A_IMAGE_FORMAT_COLOR_MJPG;
config.color_resolution = K4A_COLOR_RESOLUTION_2160P;

if (K4A_RESULT_SUCCEEDED != k4a_device_start_cameras(device, &config))
{
    printf("Failed to start cameras\n");
    goto Exit;
}

if (K4A_RESULT_SUCCEEDED != k4a_device_start_imu(device))
{
    printf("Failed to start imu\n");
    goto Exit;
}
```

## 访问 IMU 样本

每个 [k4a\\_imu\\_sample\\_t](#) 包含几乎在同一时间捕获的加速度传感器和陀螺仪读数。

可以在用于获取图像捕获的同一个线程上或者在不同的线程上获取 IMU 样本。

若要在 IMU 样本可用后立即检索它们，可在 [k4a\\_device\\_get\\_imu\\_sample\(\)](#) 自身的线程上调用该函数。API 还提供足够的内部队列，使你可以在仅返回每个图像捕获后才检查样本。

由于 IMU 样本存在某种内部队列，你可以在不丢弃任何数据的情况下使用以下模式：

1. 以任意帧速率等待捕获。
2. 处理捕获。
3. 检索所有已排队的 IMU 样本。
4. 重复等待下一个捕获。

若要检索当前已排队的所有 IMU 样本，可以在循环中结合 0 值 [timeout\\_in\\_ms](#) 调用 [k4a\\_device\\_get\\_imu\\_sample\(\)](#)，

直到函数返回 `K4A_WAIT_RESULT_TIMEOUT`。`K4A_WAIT_RESULT_TIMEOUT` 指示没有排队的样本，且在指定的超时内没有任何样本抵达。

## 用例

```
k4a_imu_sample_t imu_sample;

// Capture a imu sample
switch (k4a_device_get_imu_sample(device, &imu_sample, TIMEOUT_IN_MS))
{
case K4A_WAIT_RESULT_SUCCEEDED:
    break;
case K4A_WAIT_RESULT_TIMEOUT:
    printf("Timed out waiting for a imu sample\n");
    continue;
    break;
case K4A_WAIT_RESULT_FAILED:
    printf("Failed to read a imu sample\n");
    goto Exit;
}

// Access the accelerometer readings
if (imu_sample != NULL)
{
    printf(" | Accelerometer temperature:%.2f x:%.4f y:%.4f z: %.4f\n",
        imu_sample.temperature,
        imu_sample.acc_sample.xyz.x,
        imu_sample.acc_sample.xyz.y,
        imu_sample.acc_sample.xyz.z);
}
```

## 后续步骤

现在你已了解如何使用 IMU 样本，接下来还可以

[访问麦克风输入数据](#)

# 访问 Azure Kinect DK 麦克风输入数据

2020/8/11 • [Edit Online](#)

[语音 SDK 快速入门](#)提供了示例用于演示如何以各种编程语言使用 Azure Kinect DK 麦克风阵列。例如, 请参阅“使用语音 SDK 在 Windows 上的 C++ 中识别语音”快速入门。可以[从 GitHub 中获取该代码](#)。

同时通过 Windows API 访问麦克风阵列。有关 Windows 文档的详细信息, 请参阅以下文档:

- [Windows 音频体系结构](#)
- [Windows.Media.Capture 文档](#)
- [网络摄像头捕获教程](#)
- [USB 音频信息](#)

还可以查看[麦克风阵列硬件规格](#)。

## 后续步骤

[语音服务 SDK](#)

# 使用 Azure Kinect 传感器 SDK 图像转换

2020/8/11 • [Edit Online](#)

遵循特定的函数在 Azure Kinect DK 中的协调相机系统之间使用和转换图像。

## k4a\_transformation 函数

所有带有 *k4a\_transformation* 前缀的函数针对整个图像运行。它们需要通过 [k4a\\_transformation\\_create\(\)](#) 获取转换句柄 *k4a\_transformation\_t*, 并通过 [k4a\\_transformation\\_destroy\(\)](#) 取消分配。你还可以参考 [SDK 转换示例](#), 其中演示了如何使用本主题所述的三个函数。

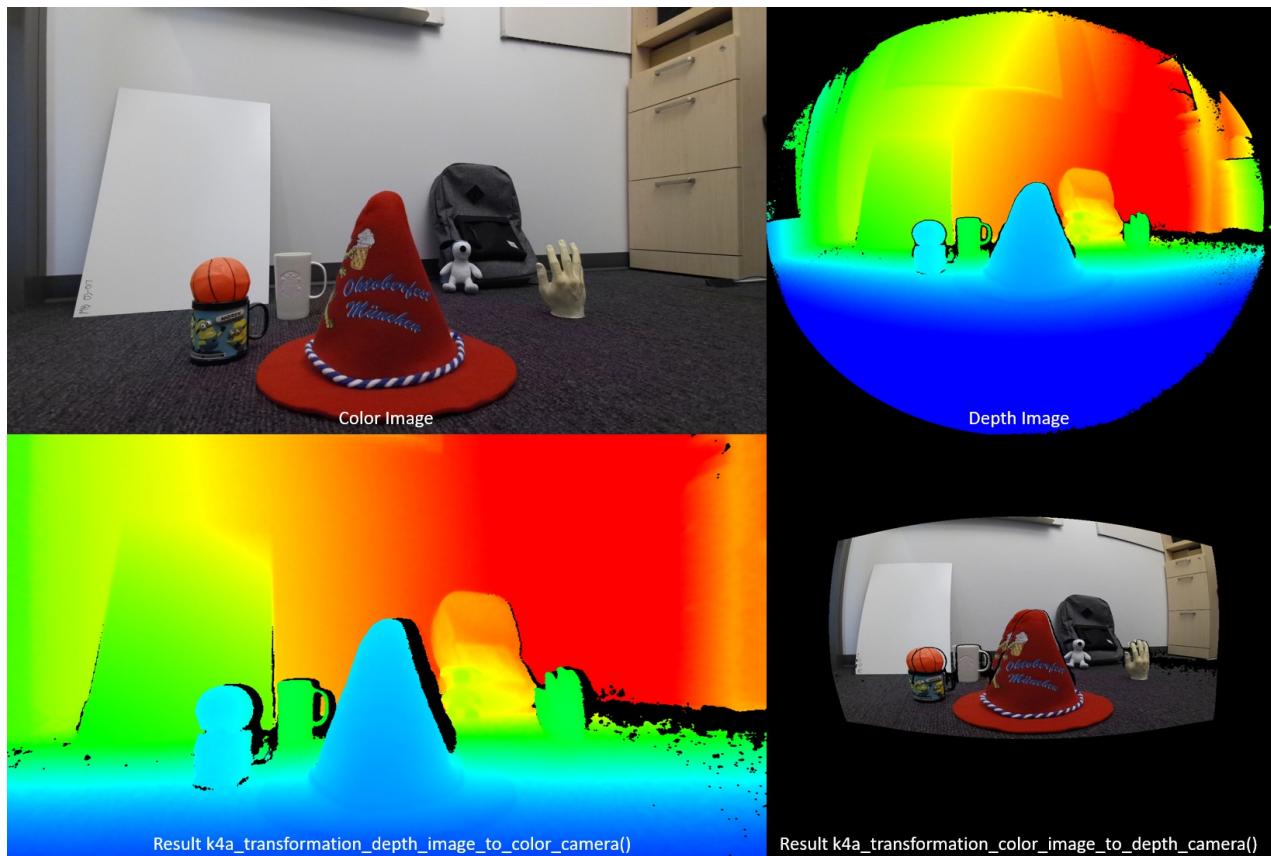
本文将介绍以下函数:

- [k4a\\_transformation\\_depth\\_image\\_to\\_color\\_camera\(\)](#)
- [k4a\\_transformation\\_depth\\_image\\_to\\_color\\_camera\\_custom\(\)](#)
- [k4a\\_transformation\\_color\\_image\\_to\\_depth\\_camera\(\)](#)
- [k4a\\_transformation\\_depth\\_image\\_to\\_point\\_cloud\(\)](#)

### **k4a\_transformation\_depth\_image\_to\_color\_camera**

#### 概述

函数 [k4a\\_transformation\\_depth\\_image\\_to\\_color\\_camera\(\)](#) 将深度图从深度相机的视点转换为彩色相机的视点。此函数旨在生成所谓的 RGB-D 图像, 其中, D 表示录制深度值的附加图像通道。在下图中可以看到, [k4a\\_transformation\\_depth\\_image\\_to\\_color\\_camera\(\)](#) 的彩色图像和输出如同它们取自同一视点(即, 彩色相机的视点)。



#### 实现

此转换函数比单纯针对每个像素调用 [k4a\\_calibration\\_2d\\_to\\_2d\(\)](#) 更为复杂。它将深度相机几何结构中的三角网格扭曲成彩色相机的几何结构。使用三角网格可以避免转换的深度图像出现孔洞。Z 缓冲区确保正确处理遮挡物。

默认已为此函数启用 GPU 加速。

#### parameters

输入参数是转换句柄和深度图像。深度图像的分辨率必须与创建转换句柄时指定的 `depth_mode` 相匹配。例如, 如果转换句柄是使用 1024x1024 `K4A_DEPTH_MODE_WFOV_UNBINNED` 模式创建的, 则深度图像的分辨率必须是 1024x1024 像素。输出是需要由用户调用 `k4a_image_create()` 分配的已转换深度图像。已转换的深度图像的分辨率必须与创建转换句柄时指定的 `color_resolution` 相匹配。例如, 如果颜色分辨率设置为 `K4A_COLOR_RESOLUTION_1080P`, 则输出图像的分辨率必须是 1920x1080 像素。输出图像步幅设置为 `width * sizeof(uint16_t)`, 因为图像存储 16 位深度值。

### `k4a_transformation_depth_image_to_color_camera_custom`

#### 概述

函数 `k4a_transformation_depth_image_to_color_camera_custom()` 将深度图和自定义图像从深度相机的视点转换为彩色相机的视点。作为 `k4a_transformation_depth_image_to_color_camera()` 的扩展, 此函数旨在生成对应的自定义图像, 其中每个像素与除了转换的深度图像之外的彩色相机的相应像素坐标相匹配。

#### 实现

此转换函数生成转换的深度图像的方式与 `k4a_transformation_depth_image_to_color_camera()` 相同。若要转换自定义图像, 此函数提供了使用线性内插或最近的邻域内插的选项。使用线性内插可以在转换后的自定义图像中创建新值。使用最近的邻域内插将防止原始图像中不存在的值出现在输出图像中, 但会导致图像不太平滑。自定义图像应为单通道 8 位或 16 位。默认已为此函数启用 GPU 加速。

#### parameters

输入参数为转换句柄、深度图像、自定义图像和内插类型。深度图像和自定义图像的分辨率必须与创建转换句柄时指定的 `depth_mode` 相匹配。例如, 如果转换句柄是使用 1024x1024 `K4A_DEPTH_MODE_WFOV_UNBINNED` 模式创建的, 则深度图像和自定义图像的分辨率必须是 1024x1024 像素。`interpolation_type` 应为 `K4A_TRANSFORMATION_INTERPOLATION_TYPE_LINEAR` 或 `K4A_TRANSFORMATION_INTERPOLATION_TYPE_NEAREST`。输出是转换后的深度图像和转换后的自定义图像, 需要由用户通过调用 `k4a_image_create()` 来分配。转换后的深度图像和转换后的自定义图像的分辨率必须与创建转换句柄时指定的 `color_resolution` 相匹配。例如, 如果颜色分辨率设置为 `K4A_COLOR_RESOLUTION_1080P`, 则输出图像的分辨率必须是 1920x1080 像素。输出深度图像步幅设置为 `width * sizeof(uint16_t)`, 因为该图像存储 16 位深度值。输入的自定义图像和转换后的自定义图像的格式必须为 `K4A_IMAGE_FORMAT_CUSTOM8` 或 `K4A_IMAGE_FORMAT_CUSTOM16`, 应相应设置对应的图像步幅。

### `k4a_transformation_color_image_to_depth_camera`

#### 概述

函数 `k4a_transformation_color_image_to_depth_camera()` 将彩色图像从彩色相机的视点转换为深度相机的视点 (参阅上图)。使用此函数可以生成 RGB-D 图像。

#### 实现

对于深度图的每个像素, 该函数使用像素的深度值来计算彩色图像中的相应子像素坐标。然后, 我们将在彩色图像中的此坐标处查找颜色值。在彩色图像中执行双线性内插, 以获取子像素精度的颜色值。没有关联的深度读数的像素将分配到输出图像中的 BGRA 值 `[0,0,0,0]`。默认已为此函数启用 GPU 加速。由于此方法会在转换的彩色图像中产生孔洞, 并且不会处理遮挡物, 因此我们建议改用函数 `k4a_transformation_depth_image_to_color_camera()`。

#### parameters

输入参数是转换句柄、深度图像和彩色图像。深度图像和彩色图像的分辨率必须与创建转换句柄时指定的 `depth_mode` 和 `color_resolution` 相匹配。输出是需要由用户调用 `k4a_image_create()` 分配的已转换彩色图像。已转换的彩色图像的分辨率必须与创建转换句柄时指定的 `depth_resolution` 相匹配。输出图像存储四个 8 位值(表示每个像素的 BGRA)。因此, 图像的步幅为 `width * 4 * sizeof(uint8_t)`。数据顺序为像素交错式, 即, 蓝色值 - 像素 0, 绿色值 - 像素 0, 红色值 - 像素 0, alpha 值 - 像素 0, 蓝色值 - 1, 依此类推。

### `k4a_transformation_depth_image_to_point_cloud`

#### 概述

函数 `k4a_transformation_depth_image_to_point_cloud()` 将相机拍摄的 2D 深度图转换为同一相机的坐标系中的

3D 点云。因此，相机可以是深度相机或彩色相机。

实现

该函数的结果与针对每个像素运行 `k4a_calibration_2d_to_2d()` 的结果相同，不过计算效率更高。调用 `k4a_transformation_create()` 时，我们会预先计算一个所谓的 xy 查找表，用于存储每个图像像素的 x 和 y 比例因子。调用 `k4a_transformation_depth_image_to_point_cloud()` 时，我们会通过将像素的 x 比例因子与像素的 Z 坐标相乘，来获取像素的 3D X 坐标。类似地，与 y 比例因子相乘可以计算出 3D Y 坐标。SDK 的[快速点云示例](#)演示了如何计算 xy 表。例如，用户可以遵循示例代码实现其自有版本的此函数，以加速其 GPU 管道。

#### parameters

输入参数是转换句柄、相机说明符和深度图像。如果相机说明符设置为深度，则深度图像的分辨率必须与创建转换句柄时指定的 `depth_mode` 相匹配。否则，如果说明符设置为彩色相机，则分辨率必须与所选 `color_resolution` 的分辨率相匹配。输出参数是需要由用户调用 `k4a_image_create()` 分配的 XYZ 图像。该 XYZ 图像的分辨率必须与输入的深度图的分辨率相匹配。我们将为每个像素存储三个带符号的 16 位坐标值（以毫米为单位）。因此，XYZ 图像步幅设置为 `width * 3 * sizeof(int16_t)`。数据顺序为像素交错式，即，X 坐标 – 像素 0, Y 坐标 – 像素 0, Z 坐标 – 像素 0, X 坐标 – 像素 1, 依此类推。如果无法将某个像素转换为 3D，该函数将为该像素分配值 `[0,0,0]`。

## 示例

[转换示例](#)

## 后续步骤

了解如何使用 Azure Kinect 传感器 SDK 图像转换函数后，接下来还可以了解

[Azure Kinect 传感器 SDK 校准函数](#)

此外，可以了解

[坐标系](#)

# 使用 Azure Kinect 校准函数

2020/8/11 • [Edit Online](#)

使用校准函数可以转换 Azure Kinect 设备上每个传感器的坐标系之间的点。需要转换整个图像的应用程序可以利用[转换函数](#)提供的加速操作。

## 检索校准数据

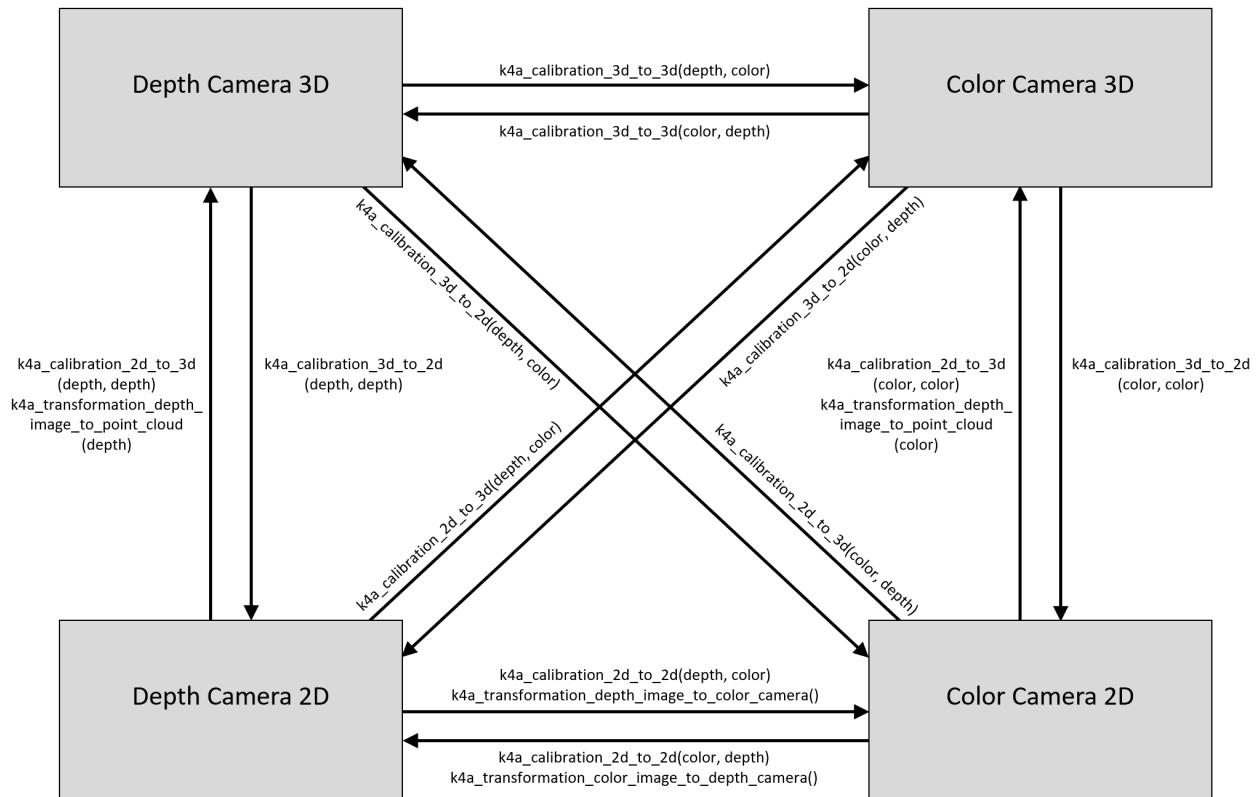
需要检索设备校准数据才能执行坐标系转换。校准数据存储在 `k4a_calibration_t` 数据类型中。该数据是通过 `k4a_device_get_calibration()` 函数从设备获取的。校准数据不仅特定于每个设备，而且还特定于相机的工作模式。因此，`k4a_device_get_calibration()` 需要使用 `depth_mode` 和 `color_resolution` 参数作为输入。

## OpenCV 兼容性

校准参数与 [OpenCV](#) 兼容。有关各个相机校正参数的详细信息，另请参阅 [OpenCV 文档](#)。另请参阅 SDK 的 [OpenCV 兼容性示例](#)，其中演示了 `k4a_calibration_t` 类型与相应 OpenCV 数据结构之间的转换。

## 坐标转换函数

下图显示了 Azure Kinect 的不同坐标系，以及用于在这些坐标系之间转换的函数。为了使插图显得简洁，我们省略了陀螺仪和加速度传感器的 3D 坐标系。



有关镜头失真的备注：2D 坐标始终引用 SDK 中失真的图像。SDK 的[失真矫正示例](#)演示了图像失真矫正。一般情况下，3D 点永远不会受到镜头失真的影响。

## 在 3D 坐标系之间转换

函数 `k4a_calibration_3d_to_3d()` 使用相机的外部校准将源坐标系的 3D 点转换为目标坐标系的 3D 点。源和目标可设置为四个 3D 坐标系中的任何一个，即，彩色相机、深度相机、陀螺仪或加速度传感器。如果源和目标相同，则返回未修改的输入 3D 点作为输出。

## 在 2D 与 3D 坐标系之间转换

函数 [k4a\\_calibration\\_3d\\_to\\_2d\(\)](#) 将源坐标系的 3D 点转换为目标相机的 2D 像素坐标。此函数通常称为投影函数。尽管源可以设置为四个 3D 坐标系中的任何一个，但目标必须是深度相机或彩色相机。如果源和目标不同，则会使用 [k4a\\_calibration\\_3d\\_to\\_3d\(\)](#) 将输入的 3D 点转换为目标相机的 3D 坐标系。以目标相机坐标系表示 3D 点后，将使用目标相机的内部校准计算相应的 2D 像素坐标。如果 3D 点超出了目标相机的可视区域，则有效值将设置为 0。

函数 [k4a\\_calibration\\_2d\\_to\\_3d\(\)](#) 将源相机的 2D 像素坐标转换为目标相机坐标系的 3D 点。源必须是彩色相机或深度相机。目标可设置为四个 3D 坐标系中的任何一个。除了 2D 像素坐标外，源相机图像中像素的深度值（以毫米为单位）也需要作为函数的输入，在彩色相机几何中推导出深度值的一种方法是使用函数

[k4a\\_transformation\\_depth\\_image\\_to\\_color\\_camera\(\)](#)。该函数使用源相机的内部校准通过指定的像素坐标计算源相机焦点引出的 3D 射线。然后，使用深度值查找此射线的确切 3D 点位置。此操作通常称为取消投影函数。如果源和目标相机不同，该函数会通过 [k4a\\_calibration\\_3d\\_to\\_3d\(\)](#) 将 3D 点转换为目的地的坐标系。如果 2D 像素坐标超出了源相机的可视区域，则有效值将设置为 0。

## 在 2D 坐标系之间转换

函数 [k4a\\_calibration\\_2d\\_to\\_2d\(\)](#) 将源相机的 2D 像素坐标转换为目标相机的 2D 像素坐标。源和目标必须设置为彩色相机或深度相机。该函数需要将源相机图像中像素的深度值（以毫米为单位）作为输入，在彩色相机几何中推导出深度值的一种方法是使用函数 [k4a\\_transformation\\_depth\\_image\\_to\\_color\\_camera\(\)](#)。它将调用 [k4a\\_calibration\\_2d\\_to\\_3d\(\)](#) 转换为源相机系统的 3D 点。然后，它将调用 [k4a\\_calibration\\_3d\\_to\\_2d\(\)](#) 转换为目标相机图像的 2D 像素坐标。如果 [k4a\\_calibration\\_2d\\_to\\_3d\(\)](#) 或 [k4a\\_calibration\\_3d\\_to\\_2d\(\)](#) 返回无效的结果，则有效值将设置为 0。

## 相关示例

- [OpenCV 兼容性示例](#)
- [失真矫正示例](#)
- [快速点云示例](#)

## 后续步骤

了解相机校正后，接下来还可以了解如何

[捕获设备同步](#)

此外，可以了解

[坐标系](#)

# 捕获 Azure Kinect 设备同步

2020/8/11 • [Edit Online](#)

Azure Kinect 硬件可以调整彩色和深度图像的捕获时间。同一设备上的相机之间的调整属于内部同步。跨多个连接设备的捕获时间调整属于外部同步。

## 设备内部同步

各个相机之间的图像捕获在硬件中同步。在包含来自彩色和深度传感器的图像的每个 `k4a_capture_t` 中，图像的时间戳根据硬件的运行模式进行调整。默认情况下，捕获的图像是调整的曝光的中心。可以使用 `k4a_device_configuration_t` 的 `depth_delay_off_color_usec` 字段调整深度和彩色捕获的相对计时。

## 设备外部同步

有关硬件设置，请参阅[设置外部同步](#)。

每个连接设备的软件必须配置为以主模式或从属模式运行。在 `k4a_device_configuration_t` 中配置此设置。

使用外部同步时，始终应该先启动从属相机，然后再启动主相机，这样才能正常调整时间戳。

### 从属模式

```
k4a_device_configuration_t deviceConfig;  
deviceConfig.wired_sync_mode = K4A_WIRED_SYNC_MODE_SUBORDINATE
```

### 主模式

```
k4a_device_configuration_t deviceConfig;  
deviceConfig.wired_sync_mode = K4A_WIRED_SYNC_MODE_MASTER;
```

### 检索同步插孔状态

若要以编程方式检索同步输入和同步输出插孔的当前状态，请使用 `k4a_device_get_sync_jack` 函数。

## 后续步骤

现在，你已了解如何启用和捕获设备同步。你还可以查看如何使用

[Azure Kinect 传感器 SDK 录制和播放 API](#)

# Azure Kinect 播放 API

2020/8/11 • [Edit Online](#)

传感器 SDK 提供一个 API 用于将设备数据记录到 Matroska (.mkv) 文件。Matroska 容器格式可存储视频篇目、IMU 样本和设备校准结果。可以使用随附的 [k4arecorder](#) 命令行实用工具生成录制内容。还可以直接使用录制 API 录制内容以及自定义录制的内容。

有关录制 API 的详细信息, 请参阅 [k4a\\_record\\_create\(\)](#)。

有关 Matroska 文件格式规范的详细信息, 请参阅[录制文件格式](#)页。

## 使用播放 API

可以使用播放 API 打开录制文件。通过播放 API 可以访问格式与其他传感器 SDK 相同的传感器数据。

### 打开录制文件

在以下示例中, 我们将使用 [k4a\\_playback\\_open\(\)](#) 打开一个录制文件, 输出录制长度, 然后使用 [k4a\\_playback\\_close\(\)](#) 关闭该文件。

```
k4a_playback_t playback_handle = NULL;
if (k4a_playback_open("recording.mkv", &playback_handle) != K4A_RESULT_SUCCEEDED)
{
    printf("Failed to open recording\n");
    return 1;
}

uint64_t recording_length = k4a_playback_get_last_timestamp_usec(playback_handle);
printf("Recording is %lld seconds long\n", recording_length / 1000000);

k4a_playback_close(playback_handle);
```

### 读取捕获

打开文件后, 可以开始读取录制内容中的捕获。以下示例将读取文件中的每个捕获。

```
k4a_capture_t capture = NULL;
k4a_stream_result_t result = K4A_STREAM_RESULT_SUCCEEDED;
while (result == K4A_STREAM_RESULT_SUCCEEDED)
{
    result = k4a_playback_get_next_capture(playback_handle, &capture);
    if (result == K4A_STREAM_RESULT_SUCCEEDED)
    {
        // Process capture here
        k4a_capture_release(capture);
    }
    else if (result == K4A_STREAM_RESULT_EOF)
    {
        // End of file reached
        break;
    }
}
if (result == K4A_STREAM_RESULT_FAILED)
{
    printf("Failed to read entire recording\n");
    return 1;
}
```

## 在录制内容中查找

到达文件末尾后，我们可能需要返回并再次读取。若要完成此过程，可以使用 [k4a\\_playback\\_get\\_previous\\_capture\(\)](#) 进行回读，但根据录制内容的长度，这种做法可能非常缓慢。我们可以改用 [k4a\\_playback\\_seek\\_timestamp\(\)](#) 函数转到文件中的特定点。

在此示例中，我们以微秒为单位指定了时间戳，以定位到文件中的各个点。

```
// Seek to the beginning of the file
if (k4a_playback_seek_timestamp(playback_handle, 0, K4A_PLAYBACK_SEEK_BEGIN) != K4A_RESULT_SUCCEEDED)
{
    return 1;
}

// Seek to the end of the file
if (k4a_playback_seek_timestamp(playback_handle, 0, K4A_PLAYBACK_SEEK_END) != K4A_RESULT_SUCCEEDED)
{
    return 1;
}

// Seek to 10 seconds from the start
if (k4a_playback_seek_timestamp(playback_handle, 10 * 1000000, K4A_PLAYBACK_SEEK_BEGIN) != K4A_RESULT_SUCCEEDED)
{
    return 1;
}

// Seek to 10 seconds from the end
if (k4a_playback_seek_timestamp(playback_handle, -10 * 1000000, K4A_PLAYBACK_SEEK_END) != K4A_RESULT_SUCCEEDED)
{
    return 1;
}
```

## 读取标记信息

录制内容还可能包含各种元数据，例如设备序列号和固件版本。此元数据存储在录制标记中，可以使用 [k4a\\_playback\\_get\\_tag\(\)](#) 函数来访问。

```
// Print the serial number of the device used to record
char serial_number[256];
size_t serial_number_size = 256;
k4a_buffer_result_t buffer_result = k4a_playback_get_tag(playback_handle, "K4A_DEVICE_SERIAL_NUMBER",
&serial_number, &serial_number_size);
if (buffer_result == K4A_BUFFER_RESULT_SUCCEEDED)
{
    printf("Device serial number: %s\n", serial_number);
}
else if (buffer_result == K4A_BUFFER_RESULT_TOO_SMALL)
{
    printf("Device serial number too long.\n");
}
else
{
    printf("Tag does not exist. Device serial number was not recorded.\n");
}
```

## 录制标记列表

下面是可以包含在录制文件中的所有默认标记的列表。其中的许多值可用作 [k4a\\_record\\_configuration\\_t](#) 结构的一部分，可以使用 [k4a\\_playback\\_get\\_record\\_configuration\(\)](#) 函数来读取。

如果某个标记不存在，则假设采用默认值。

		<code>K4A_RECORD_CONFIGURATION_T</code>	
<code>K4A_COLOR_MODE</code>	"OFF"	<code>color_format</code> / <code>color_resolution</code>	可能的值："OFF"、"MJPG_1080P"、"NV12_720P"、"YUY2_720P"等
<code>K4A_DEPTH_MODE</code>	"OFF"	<code>depth_mode</code> / <code>depth_track_enabled</code>	可能的值："OFF"、"NFOV_UNBINNED"、"PASSIVE_IR" 等
<code>K4A_IR_MODE</code>	"OFF"	<code>depth_mode</code> / <code>ir_track_enabled</code>	可能的值："OFF"、"ACTIVE"、"PASSIVE"
<code>K4A_IMU_MODE</code>	"OFF"	<code>imu_track_enabled</code>	可能的值："ON"、"OFF"
<code>K4A_CALIBRATION_FILE</code>	"calibration.json"	空值	请参阅 <a href="#">k4a_device_get_raw_calibration()</a>
<code>K4A_DEPTH_DELAY_NS</code>	"0"	<code>depth_delay_off_color_usec</code>	值以纳秒为单位存储, API 以微秒为单位。
<code>K4A_WIRED_SYNC_MODE</code>	"STANDALONE"	<code>wired_sync_mode</code>	可能的值："STANDALONE"、"MASTER"、"SUBORDINATE"
<code>K4A_SUBORDINATE_DELAY_NS</code>	"0"	<code>subordinate_delay_off_master_usec</code>	值以纳秒为单位存储, API 以微秒为单位。
<code>K4A_COLOR_FIRMWARE_VERSION</code>	""	空值	设备颜色固件版本, 例如 "1.x.xx"
<code>K4A_DEPTH_FIRMWARE_VERSION</code>	""	空值	设备深度固件版本, 例如 "1.x.xx"
<code>K4A_DEVICE_SERIAL_NUMBER</code>	""	空值	录制设备序列号
<code>K4A_START_OFFSET_NS</code>	"0"	<code>start_timestamp_offset_usec</code>	请参阅下面的 <a href="#">时间戳同步</a> 。
<code>K4A_COLOR_TRACK</code>	无	空值	请参阅 <a href="#">录制文件格式 - 识别篇目</a> 。
<code>K4A_DEPTH_TRACK</code>	无	空值	请参阅 <a href="#">录制文件格式 - 识别篇目</a> 。
<code>K4A_IR_TRACK</code>	无	空值	请参阅 <a href="#">录制文件格式 - 识别篇目</a> 。
<code>K4A_IMU_TRACK</code>	无	空值	请参阅 <a href="#">录制文件格式 - 识别篇目</a> 。

## 时间戳同步

Matroska 格式要求录制内容必须以时间戳 0 开头。在外部同步相机时, 每个设备中的第一个时间戳可以不为 0。

为了在录制和播放之间切换时保留设备的原始时间戳, 该文件会存储一个要应用到时间戳的偏移量。

`K4A_START_OFFSET_NS` 标记用于指定时间戳偏移量，以便在录制后可以重新同步文件。可将此时间戳偏移量添加到文件中的每个时间戳，以重新构造原始设备时间戳。

起始偏移量也会在 `k4a_record_configuration_t` 结构中提供。

# 获取人体跟踪结果

2020/8/11 • [Edit Online](#)

人体跟踪 SDK 使用人体跟踪器对象处理 Azure Kinect DK 捕获并生成人体跟踪结果。它还可以维护跟踪器、处理队列和输出队列的全局状态。使用人体跟踪器需要执行三个步骤：

- 创建跟踪器
- 使用 Azure Kinect DK 捕获深度图像和 IR 图像
- 将捕获排入队列并弹出结果。

## 创建跟踪器

使用人体跟踪的第一步是创建跟踪器，并需要传入传感器校准 `k4a_calibration_t` 结构。可以使用 Azure Kinect 传感器 SDK `k4a_device_get_calibration()` 函数查询传感器校准。

```
k4a_calibration_t sensor_calibration;
if (K4A_RESULT_SUCCEEDED != k4a_device_get_calibration(device, device_config.depth_mode,
K4A_COLOR_RESOLUTION_OFF, &sensor_calibration))
{
    printf("Get depth camera calibration failed!\n");
    return 0;
}

k4abt_tracker_t tracker = NULL;
k4abt_tracker_configuration_t tracker_config = K4ABT_TRACKER_CONFIG_DEFAULT;
if (K4A_RESULT_SUCCEEDED != k4abt_tracker_create(&sensor_calibration, tracker_config, &tracker))
{
    printf("Body tracker initialization failed!\n");
    return 0;
}
```

## 捕获深度图像和 IR 图像

[检索图像](#) 页中介绍了如何使用 Azure Kinect DK 捕获图像。

### NOTE

为获得最佳性能和准确度，建议使用 `K4A_DEPTH_MODE_NFOV_UNBINNED` 或 `K4A_DEPTH_MODE_WFOV_2X2BINNED` 模式。不要使用 `K4A_DEPTH_MODE_OFF` 或 `K4A_DEPTH_MODE_PASSIVE_IR` 模式。

Azure Kinect DK [硬件规格](#) 和 `k4a_depth_mode_t` 枚举中介绍了支持的 Azure Kinect DK 模式。

```
// Capture a depth frame
switch (k4a_device_get_capture(device, &capture, TIMEOUT_IN_MS))
{
case K4A_WAIT_RESULT_SUCCEEDED:
    break;
case K4A_WAIT_RESULT_TIMEOUT:
    printf("Timed out waiting for a capture\n");
    continue;
    break;
case K4A_WAIT_RESULT_FAILED:
    printf("Failed to read a capture\n");
    goto Exit;
}
```

## 将捕获排入队列并弹出结果

跟踪器在内部维护一个输入队列和一个输出队列，以便更有效地以异步方式处理 Azure Kinect DK 捕获。使用 [k4abt\\_tracker\\_enqueue\\_capture\(\)](#) 函数将新的捕获添加到输入队列。使用 [k4abt\\_tracker\\_pop\\_result\(\)](#) 函数弹出输出队列的结果。使用的超时值与应用程序相关，控制排队等待时间。

### 实时处理

对需要实时结果并且可以适应掉帧情况的单线程应用程序使用此模式。[GitHub Azure-Kinect-Samples](#) 中的 `simple_3d_viewer` 示例是一个实时处理的示例。

```
k4a_wait_result_t queue_capture_result = k4abt_tracker_enqueue_capture(tracker, sensor_capture, 0);
k4a_capture_release(sensor_capture); // Remember to release the sensor capture once you finish using it
if (queue_capture_result == K4A_WAIT_RESULT_FAILED)
{
    printf("Error! Adding capture to tracker process queue failed!\n");
    break;
}

k4abt_frame_t body_frame = NULL;
k4a_wait_result_t pop_frame_result = k4abt_tracker_pop_result(tracker, &body_frame, 0);
if (pop_frame_result == K4A_WAIT_RESULT_SUCCEEDED)
{
    // Successfully popped the body tracking result. Start your processing
    ...

    k4abt_frame_release(body_frame); // Remember to release the body frame once you finish using it
}
```

### 同步处理

对不需要实时结果或者无法适应掉帧情况的应用程序使用此模式。

处理吞吐量可能会受限制。

[GitHub Azure-Kinect-Samples](#) 中的 `simple_sample.exe` 示例是一个同步处理的示例。

```
k4a_wait_result_t queue_capture_result = k4abt_tracker_enqueue_capture(tracker, sensor_capture,
K4A_WAIT_INFINITE);
k4a_capture_release(sensor_capture); // Remember to release the sensor capture once you finish using it
if (queue_capture_result != K4A_WAIT_RESULT_SUCCEEDED)
{
    // It should never hit timeout or error when K4A_WAIT_INFINITE is set.
    printf("Error! Adding capture to tracker process queue failed!\n");
    break;
}

k4abt_frame_t body_frame = NULL;
k4a_wait_result_t pop_frame_result = k4abt_tracker_pop_result(tracker, &body_frame, K4A_WAIT_INFINITE);
if (pop_frame_result != K4A_WAIT_RESULT_SUCCEEDED)
{
    // It should never hit timeout or error when K4A_WAIT_INFINITE is set.
    printf("Error! Popping body tracking result failed!\n");
    break;
}
// Successfully popped the body tracking result. Start your processing
...
k4abt_frame_release(body_frame); // Remember to release the body frame once you finish using it
```

## 后续步骤

[访问人体帧中的数据](#)

# 访问人体帧中的数据

2020/8/11 • [Edit Online](#)

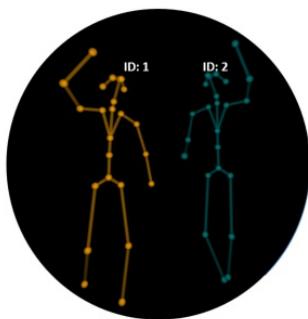
本文介绍正文框架中包含的数据，以及用于访问这些数据的函数。

本文将介绍以下函数：

- [k4abt\\_frame\\_get\\_body\\_id\(\)](#)
- [k4abt\\_frame\\_get\\_body\\_index\\_map\(\)](#)
- [k4abt\\_frame\\_get\\_body\\_skeleton\(\)](#)
- [k4abt\\_frame\\_get\\_capture\(\)](#)
- [k4abt\\_frame\\_get\\_num\\_bodies\(\)](#)
- [k4abt\\_frame\\_get\\_device\\_timestamp\\_usec\(\)](#)

## 人体帧的关键组成部分

每个人体帧包含人体结构的集合、2D 人体索引映射，以及生成了此结果的输入捕获。



A collection of body structs

- Each body struct contains:
- Body ID – the tracked ID of the body.
  - Joint Positions – the 3D position of each joint.
  - Joint Orientations – the orientation of each joint coordinate frame represented in quaternion.



2D body index map

The 2D instance segmentation map segments each body instance from the background.



Input capture

The input capture that used to generate the body tracking results.

## 访问人体结构的集合

在单个捕获中可能会检测到多个人体。可以调用 [k4abt\\_frame\\_get\\_num\\_bodies\(\)](#) 函数查询人体数目。

```
size_t num_bodies = k4abt_frame_get_num_bodies(body_frame);
```

使用 [k4abt\\_frame\\_get\\_body\\_id\(\)](#) 和 [k4abt\\_frame\\_get\\_body\\_skeleton\(\)](#) 函数可以循环访问每个人体索引，以查找人体 ID 和关节位置/方向信息。

```
for (size_t i = 0; i < num_bodies; i++)  
{  
    k4abt_skeleton_t skeleton;  
    k4abt_frame_get_body_skeleton(body_frame, i, &skeleton);  
    uint32_t id = k4abt_frame_get_body_id(body_frame, i);  
}
```

## 访问人体索引映射

使用 [k4abt\\_frame\\_get\\_body\\_index\\_map\(\)](#) 函数可以访问人体索引映射。有关人体索引映射的详细说明，请参阅[人体索引映射](#)。不再需要人体索引映射时，请务必将其释放。

```
k4a_image_t body_index_map = k4abt_frame_get_body_index_map(body_frame);
... // Do your work with the body index map
k4a_image_release(body_index_map);
```

## 访问输入捕获

人体跟踪器是一个异步 API。弹出结果时，可能已释放原始捕获。使用 [k4abt\\_frame\\_get\\_capture\(\)](#) 函数可以查询用于生成此人体跟踪结果的输入捕获。每次调用此函数，[k4a\\_capture\\_t](#) 的引用计数就会递增。不再需要捕获时，请使用 [k4a\\_capture\\_release\(\)](#) 函数。

```
k4a_capture_t input_capture = k4abt_frame_get_capture(body_frame);
... // Do your work with the input capture
k4a_capture_release(input_capture);
```

## 后续步骤

[Azure Kinect 人体跟踪 SDK](#)

# 将 Azure Kinect 库添加到 Visual Studio 项目

2020/8/11 • [Edit Online](#)

本文引导你完成将 Azure Kinect NuGet 包添加到 Visual Studio 项目的过程。

## 安装 Azure Kinect NuGet 包

若要安装 Azure Kinect NuGet 包：

- 可在以下文章中找到有关在 Visual Studio 中安装 NuGet 包的详细说明：[快速入门：在 Visual Studio 中安装和使用包](#)。
- 若要添加该包，可以使用程序包管理器 UI：右键单击“引用”，然后从解决方案资源管理器中选择“管理 NuGet 包”。
- 选择[nuget.org](#) 作为包源，选择“浏览”选项卡，然后搜索 `Microsoft.Azure.Kinect.Sensor`。
- 从列表中选择该包并安装。

## 使用 Azure Kinect NuGet 包

添加该包后，将标头文件中的 include 语句添加到源代码，例如：

- `#include <k4a/k4a.h>`
- `#include <k4arecord/record.h>`
- `#include <k4arecord/playback.h>`

## 后续步骤

[现已准备好生成第一个应用程序](#)

# 更新 Azure Kinect DK 固件

2020/8/11 • [Edit Online](#)

本文档提供有关如何更新 Azure Kinect DK 上的设备固件的指导。

Azure Kinect DK 不会自动更新固件。可以使用 [Azure Kinect 固件工具](#) 手动将固件更新到最新可用版本。

## 准备固件更新

1. [下载 SDK](#)。
2. 安装 SDK。
3. 在 SDK 安装位置的 tools 文件夹中，应该可以看到：
  - AzureKinectFirmwareTool.exe
  - firmware 文件夹中的固件 .bin 文件，例如 *AzureKinectDK\_Fw\_1.5.926614.bin*。
4. 将设备连接到主机电脑并将其打开。

### IMPORTANT

在更新固件期间，请保持连接 USB 端口和电源。在更新期间断开连接可能会导致固件损坏。

## 更新设备固件

1. 在 (SDK 安装位置)\tools\ 文件夹中打开命令提示符。
2. 使用 Azure Kinect 固件工具更新固件

```
AzureKinectFirmwareTool.exe -u <device_firmware_file.bin>
```

示例：

```
AzureKinectFirmwareTool.exe -u firmware\AzureKinectDK_Fw_1.5.926614.bin
```

3. 等到固件更新完成。这可能需要几分钟时间，具体取决于映像大小。

### 验证设备固件版本

1. 验证固件是否已更新。

```
AzureKinectFirmwareTool.exe -q
```

2. 查看以下示例。

```
>AzureKinectFirmwareTool.exe -q
```

```
== Azure Kinect Firmware Tool == Device Serial Number: 000805192412 Current Firmware Versions: RGB  
camera firmware: 1.6.102 Depth camera firmware: 1.6.75 Depth config file: 6109.7 Audio firmware: 1.6.14 Build  
Config: Production Certificate Type: Microsoft ````
```

3. 如果看到上面的输出，则表示固件已更新。
4. 更新固件后，可以运行 [Azure Kinect 查看器](#) 来验证所有传感器是否按预期方式工作。

## 故障排除

固件更新可能出于多种原因而失败。如果固件更新失败, 请尝试以下缓解步骤:

1. 再次尝试运行固件更新命令。
2. 通过查询固件版本来确认设备是否仍处于连接状态。AzureKinectFirmareTool.exe
3. 如果所有其他方法均失败, 请遵循[恢复步骤](#)将固件还原为出厂设置, 然后重试。

如有其他任何问题, 请参阅 [Microsoft 支持页](#)

## 后续步骤

[Azure Kinect 固件工具](#)

# 将 Azure Kinect 录制器与外部同步设备配合使用

2020/8/11 • [Edit Online](#)

本文提供有关 [Azure Kinect 录制器](#) 如何结合配置的外部同步设备记录数据的指导。

## 必备条件

- [设置多个 Azure Kinect DK 单元进行外部同步。](#)

## 外部同步约束

- 主设备不能连接 SYNC IN 线缆。
- 主设备必须流式传输 RGB 相机数据才能启用同步。
- 所有单元必须使用相同的相机配置(帧速率和分辨率)。
- 所有单元必须运行相同的设备固件([更新固件](#)说明)。
- 必须先启动所有从属设备, 然后启动主设备。
- 应在所有设备上设置相同的曝光值。
- 每个从属设备的“主控延迟关闭”设置相对于主设备。

## 当每个单元都有一台主机电脑时进行录制

在以下示例中, 每个设备都有自身专用的主机电脑。我们建议将设备连接到专用电脑, 以防止 USB 带宽和 CPU/GPU 使用率出现问题。

### Subordinate-1

1. 设置第一个单元的录制器

```
k4arecorder.exe --external-sync sub -e -8 -r 5 -l 10 sub1.mkv
```

2. 设备开始等待

```
Device serial number: 000011590212
Device version: Rel; C: 1.5.78; D: 1.5.60[6109.6109]; A: 1.5.13
Device started
[subordinate mode] Waiting for signal from master
```

### Subordinate-2

1. 设置第二个单元的录制器

```
k4arecorder.exe --external-sync sub -e -8 -r 5 -l 10 sub2.mkv
```

2. 设备开始等待

```
Device serial number: 000011590212
Device version: Rel; C: 1.5.78; D: 1.5.60[6109.6109]; A: 1.5.13
Device started
[subordinate mode] Waiting for signal from master
```

## 主设备

1. 在主设备上开始录制

```
>k4arecorder.exe --external-sync master -e -8 -r 5 -l 10 master.mkv
```

## 2. 等到录制完成

# 当多个单元连接到单个主机电脑时进行录制

可将多个 Azure Kinect DK 连接到单个主机电脑。但是，这可能需要满足很高的 USB 带宽和主机算力要求。若要降低需求：

- 请将每个设备连接到其自身的 USB 主控制器。
- 使用强大的 GPU，它可以处理每个设备的深度引擎。
- 仅记录所需的传感器，并使用较低的帧速率。

始终先启动从属设备，最后再启动主设备。

## Subordinate-1

### 1. 在从属设备上启动录制器

```
>k4arecorder.exe --device 1 --external-sync subordinate --imu OFF -e -8 -r 5 -l 5 output-2.mkv
```

### 2. 设备进入等待状态

## 主设备

### 1. 启动主设备

```
>k4arecorder.exe --device 0 --external-sync master --imu OFF -e -8 -r 5 -l 5 output-1.mkv
```

### 2. 等待录制完成

# 播放录制内容

可以使用 [Azure Kinect 查看器](#) 播放录制内容。

## 提示

- 使用手动曝光在同步相机上进行录制。RGB 相机自动曝光可能会影响时间同步。
- 重启从属设备会导致失去同步。
- 某些[相机模式](#)最大支持 15 fps 的帧速率。我们建议不要在各个设备上使用不同的模式/帧速率
- 将多个单元连接到单个电脑很容易使 USB 带宽达到饱和，请考虑为每个设备使用单独的主机电脑。另请注意 CPU/GPU 算力。
- 如果不需要使用麦克风和 IMU 来提高可靠性，请将其禁用。

如有任何问题，请参阅[故障排除](#)

## 另请参阅

- [设置外部同步](#)
- [Azure Kinect 录制器](#)的设置及其他信息。
- 使用 [Azure Kinect 查看器](#)播放录制内容，或者设置无法通过录制器使用的 RGB 相机属性。
- 使用 [Azure Kinect 固件工具](#)更新设备固件。

# Azure Kinect 查看器

2020/8/11 • [Edit Online](#)

使用 Azure Kinect 查看器(可在工具的安装目录中找到, 其文件名为 `k4aviewer.exe`, 例如, 其路径为 `C:\Program Files\Azure Kinect SDK vX.Y.Z\tools\k4aviewer.exe`, 其中 `X.Y.Z` 是安装的 SDK 版本)可将所有设备数据流可视化, 以便:

- 验证传感器是否正常工作。
- 帮助定位设备。
- 体验不同的相机设置。
- 读取设备配置。
- 播放使用 [Azure Kinect 录制器](#)录制的内容。

有关 Azure Kinect 查看器的详细信息, 请观看[如何使用 Azure Kinect 视频](#)。

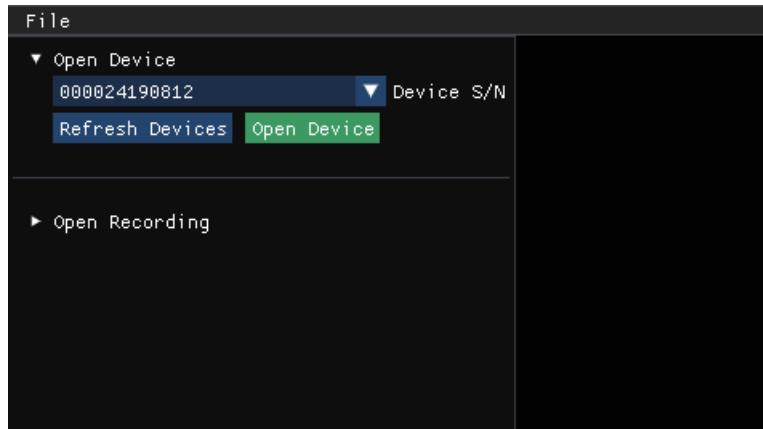
Azure Kinect 查看器是[开源的](#), 可作为一个例子来演示如何使用 API。

## 使用查看器

该查看器能够以两种模式运行:结合传感器发送的实时数据, 或者结合录制的数据([Azure Kinect 录制器](#))。

### 启动应用程序

运行 `k4aviewer.exe` 启动应用程序。



### 结合实时数据使用查看器

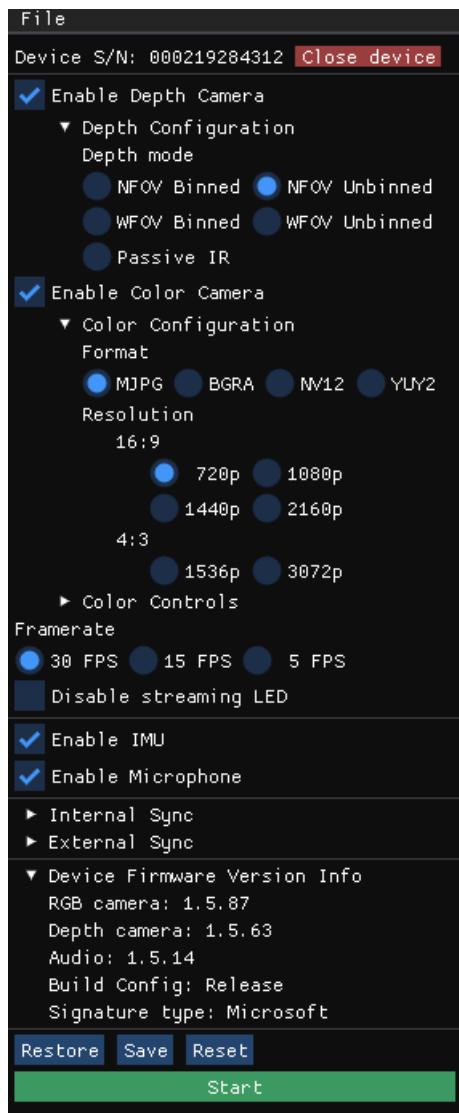
1. 在“打开设备”部分, 选择设备的“序列号”将其打开。如果该设备未列出, 请选择“刷新”。
2. 选择“打开设备”按钮。
3. 选择“启动”, 开始使用默认设置流式传输数据。

### 结合录制的数据使用查看器

在“打开录制内容”部分, 导航到录制的文件并将其选中。

## 检查设备固件版本

如下图所示, 在配置窗口中访问设备固件版本。



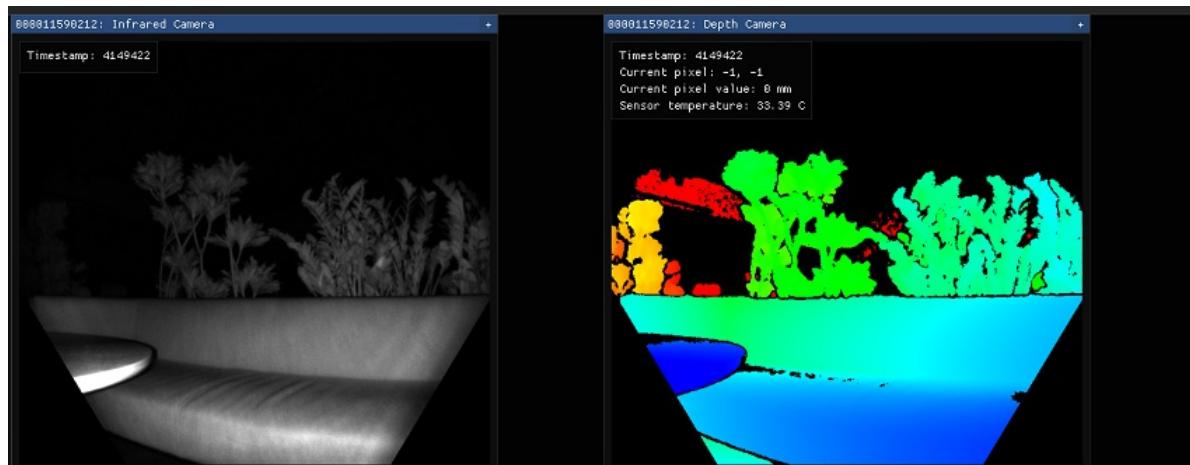
例如，在本例中，深度相机 ISP 运行的是 FW 1.5.63。

## 深度相机

深度相机查看器将显示两个窗口：

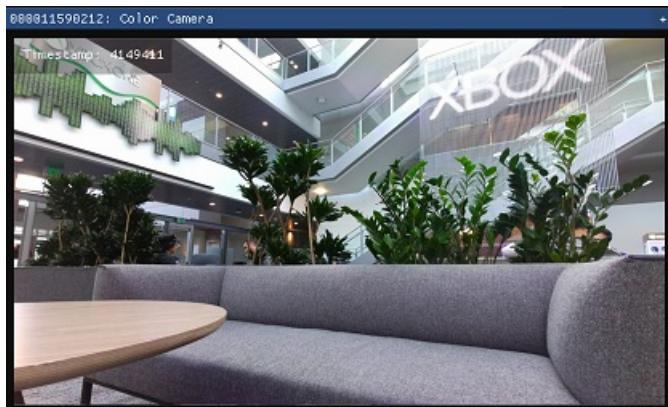
- 一个窗口的名称为“有效亮度”，即，显示 IR 亮度的灰度图像。
- 另一个窗口的名称为“深度”，以不同的颜色表示深度数据。

将鼠标悬停在深度窗口中的像素上可以查看深度传感器的值，如下所示。

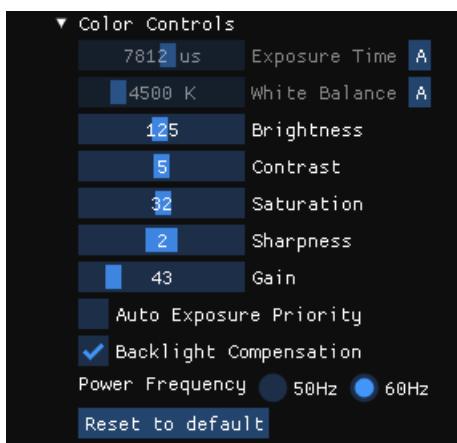


## RGB 相机

下图显示了彩色相机视图。



在流式传输期间，可以通过配置窗口控制 RGB 相机设置。

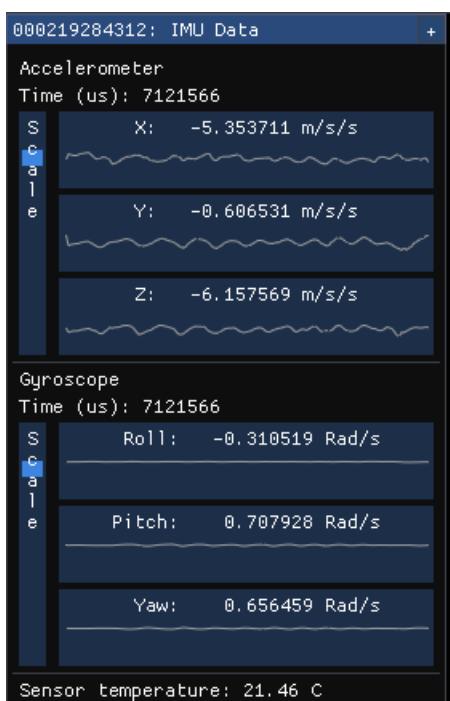


## 惯性测量单元 (IMU)

IMU 窗口包含两个组件：加速度传感器和陀螺仪。

上半部分是加速度传感器，以米/秒<sup>2</sup>为单位显示线性加速度。它包含了重力加速度，因此，如果将它平稳地放在桌面上，Z 轴可能会显示大约 -9.8 米/秒<sup>2</sup>。

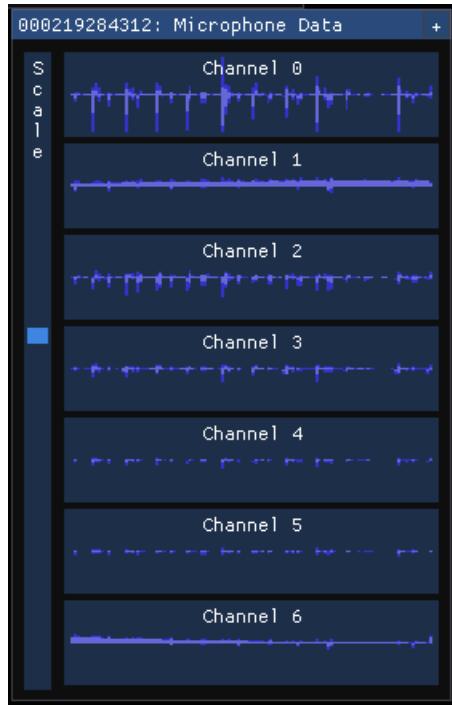
下半部分是陀螺仪，以弧度/秒为单位显示旋转运动



## 麦克风输入

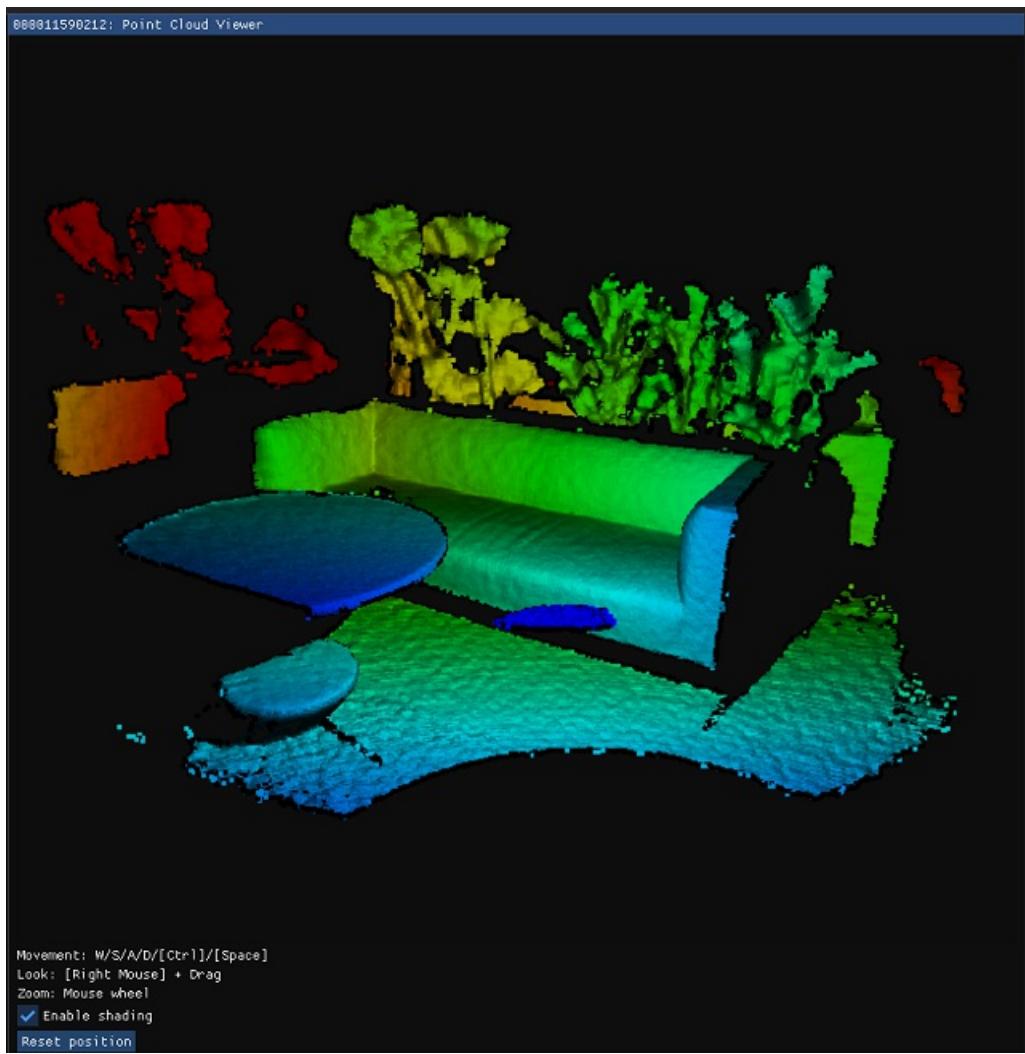
麦克风视图显示了在每个麦克风中听到的声音的表示形式。如果没有声音，图形是空白的；否则，你会看到深蓝色的波形，其顶部叠加了浅蓝色的波形。

深色波形表示麦克风在该时间切片内观测到的最小值和最大值。浅色波形表示麦克风在该时间切片内观测到的值的均方根。



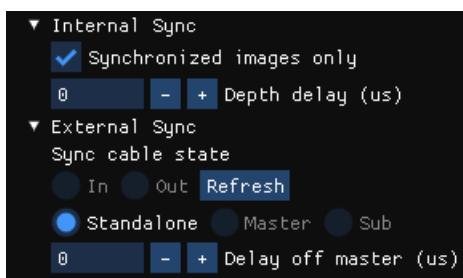
## 点云可视化

借助 3D 中可视化的深度，可以使用指示键在图像中移动。



## 同步控制

配置多设备同步时，可以使用查看器将设备配置为独立(默认)、主控或从属模式。更改配置或插入/移除同步线缆时，请选择“刷新”进行更新。



## 后续步骤

[外部同步设置指南](#)

# Azure Kinect DK 录制器

2020/8/11 • [Edit Online](#)

本文介绍如何使用 `k4arecorder` 命令行实用工具将传感器 SDK 发出的数据流录制到文件中。

## NOTE

Azure Kinect 录制器不会录制音频。

## 录制器选项

`k4arecorder` 提供不同的命令行参数用于指定输出文件和录制模式。

录制内容以 [Matroska.mkv 格式](#) 存储。录制内容为彩色和深度图像使用多个视频轨道，并包含相机校准和元数据等其他信息。

```
k4arecorder [options] output.mkv

Options:
-h, --help                  Prints this help
--list                      List the currently connected K4A devices
--device                     Specify the device index to use (default: 0)
-l, --record-length         Limit the recording to N seconds (default: infinite)
-c, --color-mode             Set the color sensor mode (default: 1080p), Available options:
                             3072p, 2160p, 1536p, 1440p, 1080p, 720p, 720p_NV12, 720p_YUY2, OFF
-d, --depth-mode            Set the depth sensor mode (default: NFOV_UNBINNED), Available options:
                             NFOV_2X2BINNED, NFOV_UNBINNED, WFOV_2X2BINNED, WFOV_UNBINNED, PASSIVE_IR, OFF
--depth-delay                Set the time offset between color and depth frames in microseconds (default: 0)
                             A negative value means depth frames will arrive before color frames.
                             The delay must be less than 1 frame period.
-r, --rate                   Set the camera frame rate in Frames per Second
                             Default is the maximum rate supported by the camera modes.
                             Available options: 30, 15, 5
--imu                       Set the IMU recording mode (ON, OFF, default: ON)
--external-sync              Set the external sync mode (Master, Subordinate, Standalone default: Standalone)
--sync-delay                 Set the external sync delay off the master camera in microseconds (default: 0)
                             This setting is only valid if the camera is in Subordinate mode.
-e, --exposure-control      Set manual exposure value (-11 to 1) for the RGB camera (default: auto exposure)
```

## 录制文件

示例 1。录制深度 NFOV 非装箱 (640x576) 模式 RGB 1080p @ 30 fps(包括 IMU) 内容。按 **CTRL-C** 键可停止录制。

```
k4arecorder.exe output.mkv
```

示例 2。录制 WFOV 非装箱 (1MP) RGB 3072p @ 15 fps(不包括 IMU) 内容 10 秒。

```
k4arecorder.exe -d WFOV_UNBINNED -c 3072p -r 15 -l 10 --imu OFF output.mkv
```

示例 3。录制 WFOV 2x2 装箱 30 fps 内容 5 秒，并保存到 output.mkv。

```
k4arecorder.exe -d WFOV_2X2BINNED -c OFF --imu OFF -l 5 output.mkv
```

#### TIP

在录制之前，可以使用 [Azure Kinect 查看器](#) 配置 RGB 相机控件（例如设置手动白平衡）。

## 验证录制

可以使用 [Azure Kinect 查看器](#) 打开输出的 .mkv 文件。

若要提取轨道或查看文件信息，可以使用 [MKVToolNix](#) 工具包中的 `mkvinfo` 等工具。

## 后续步骤

[将录制器与外部同步单元配合使用](#)

# Azure Kinect DK 固件工具

2020/8/11 • [Edit Online](#)

Azure Kinect 固件工具可用于查询和更新 Azure Kinect DK 的设备固件。

## 列出已连接的设备

可以使用 `-l` 选项获取已连接的设备列表。 `AzureKinectFirmwareTool.exe -l`

```
== Azure Kinect DK Firmware Tool ==
Found 2 connected devices:
0: Device "000036590812"
1: Device "000274185112"
```

## 检查设备固件版本

可以使用 `-q` 选项检查第一个附加设备的当前固件版本，例如 `AzureKinectFirmwareTool.exe -q`。

```
== Azure Kinect DK Firmware Tool ==
Device Serial Number: 000036590812
Current Firmware Versions:
RGB camera firmware: 1.5.92
Depth camera firmware: 1.5.66
Depth config file: 6109.7
Audio firmware: 1.5.14
Build Config: Production
Certificate Type: Microsoft
```

如果附加了多个设备，可以通过将完整的序列号添加到命令，来指定要查询的设备，例如：

```
AzureKinectFirmwareTool.exe -q 000036590812
```

## 更新设备固件

此工具的最常见用途是更新设备固件。使用 `-u` 选项调用该工具可以执行更新。固件更新可能需要几分钟时间，具体取决于必须更新哪些固件文件。

有关固件更新的分步说明，请参阅 [Azure Kinect 固件更新](#)。

```
AzureKinectFirmwareTool.exe -u firmware\AzureKinectDK_Fw_1.5.926614.bin
```

如果附加了多个设备，可以通过将完整的序列号添加到命令，来指定要查询的设备。

```
AzureKinectFirmwareTool.exe -u firmware\AzureKinectDK_Fw_1.5.926614.bin 000036590812
```

## 重置设备

如果必须将设备置于某种已知状态，可以使用 `-r` 选项重置附加的 Azure Kinect DK。

如果附加了多个设备，可以通过将完整的序列号添加到命令，来指定要查询的设备。

```
AzureKinectFirmwareTool.exe -r 000036590812
```

## 检查固件

在更新实际设备之前，可以通过检查固件从固件 bin 文件中获取版本信息。

```
AzureKinectFirmwareTool.exe -i firmware\AzureKinectDK_Fw_1.5.926614.bin
```

```
== Azure Kinect DK Firmware Tool ==
Loading firmware package ..\tools\update\firmware\AzureKinectDK_Fw_1.5.926614.bin.
File size: 1228844 bytes
This package contains:
    RGB camera firmware:      1.5.92
    Depth camera firmware:    1.5.66
    Depth config files: 6109.7 5006.27
    Audio firmware:          1.5.14
    Build Config:            Production
    Certificate Type:        Microsoft
    Signature Type:          Microsoft
```

## 固件更新工具选项

```
== Azure Kinect DK Firmware Tool ==
* Usage Info *
    AzureKinectFirmwareTool.exe <Command> <Arguments>

Commands:
    List Devices: -List, -l
    Query Device: -Query, -q
        Arguments: [Serial Number]
    Update Device: -Update, -u
        Arguments: <Firmware Package Path and FileName> [Serial Number]
    Reset Device: -Reset, -r
        Arguments: [Serial Number]
    Inspect Firmware: -Inspect, -i
        Arguments: <Firmware Package Path and FileName>

    If no Serial Number is provided, the tool will just connect to the first device.

Examples:
    AzureKinectFirmwareTool.exe -List
    AzureKinectFirmwareTool.exe -Update c:\data\firmware.bin 0123456
```

## 后续步骤

[有关更新设备固件的分步说明](#)

# Azure Kinect 传感器 SDK 下载

2020/8/11 • [Edit Online](#)

本页提供各版本 Azure Kinect 传感器 SDK 的下载链接。安装程序提供了 Azure Kinect 开发所需的所有文件。

## Azure Kinect 传感器 SDK 的内容

- 标头和库，用于通过 Azure Kinect DK 生成应用程序。
- 使用 Azure Kinect DK 的应用程序所需的可分发 DLL。
- [Azure Kinect 查看器](#)。
- [Azure Kinect 录制器](#)。
- [Azure Kinect 固件工具](#)。

## Windows 下载链接

[Microsoft 安装程序](#) | [GitHub 源代码](#)

### NOTE

安装 SDK 时，请记住要安装到的路径。例如，“C:\Program Files\Azure Kinect SDK 1.2”。你将要在此路径中查找文章中参考的工具。

可以在 [GitHub](#) 上找到以前版本的 Azure Kinect 传感器 SDK 和固件。

## Linux 安装说明

目前，唯一支持的分发版是 Ubuntu 18.04。若要请求对其他分发版的支持，请参阅[此页](#)。

首先，需要遵照[此处](#)的说明，配置 Microsoft 的包存储库。

现在，可以安装所需的包。`k4a-tools` 包中包含 [Azure Kinect 查看器](#)、[Azure Kinect 录制器](#) 和 [Azure Kinect 固件工具](#)。若要安装该包，请运行

```
sudo apt install k4a-tools
```

`libk4a<major>.<minor>-dev` 包中包含针对 `libk4a` 生成的头文件以及 CMake 文件。`libk4a<major>.<minor>` 包中包含共享对象，运行依赖于 `libk4a` 的可执行文件时需要这些对象。

基本教程需要 `libk4a<major>.<minor>-dev` 包。若要安装该包，请运行

```
sudo apt install libk4a1.1-dev
```

如果该命令成功，则表示 SDK 可供使用。

## 更改日志和早期版本

可在[此处](#)找到 Azure Kinect 传感器 SDK 的更改日志。

如果需要早期版本的 Azure Kinect 传感器 SDK，可在[此处](#)找到。

## 后续步骤

## 设置 Azure Kinect DK

# 下载 Azure Kinect 人体跟踪 SDK

2020/8/11 · · [Edit Online](#)

本文档提供各版本 Azure Kinect 人体跟踪 SDK 的安装链接。

## Azure Kinect 人体跟踪 SDK 的内容

- 使用 Azure Kinect DK 生成人体跟踪应用程序的头文件和库。
- 使用 Azure Kinect DK 的人体跟踪应用程序所需的可再发行 DLL。
- 人体跟踪应用程序示例。

## Windows 下载链接

版本	安装方式
1.0.1	<a href="#">msi</a> <a href="#">nuget</a>
1.0.0	<a href="#">msi</a> <a href="#">nuget</a>
0.9.5	<a href="#">msi</a> <a href="#">nuget</a>
0.9.4	<a href="#">msi</a> <a href="#">nuget</a>
0.9.3	<a href="#">msi</a> <a href="#">nuget</a>
0.9.2	<a href="#">msi</a> <a href="#">nuget</a>
0.9.1	<a href="#">msi</a> <a href="#">nuget</a>
0.9.0	<a href="#">msi</a> <a href="#">nuget</a>

## Linux 安装说明

目前，唯一支持的分发版是 Ubuntu 18.04。若要请求对其他分发版的支持，请参阅[此页](#)。

首先，需要遵照[此处](#)的说明，配置 Microsoft 的包存储库。

`libk4abt<major>.<minor>-dev` 包中包含针对 `libk4abt` 生成的头文件以及 CMake 文件。`libk4abt<major>.<minor>` 包中包含运行依赖于 `libk4abt` 的可执行文件以及示例查看器所需的共享对象。

基本教程需要 `libk4abt<major>.<minor>-dev` 包。若要安装该包，请运行

```
sudo apt install libk4abt1.0-dev
```

如果该命令成功，则表示 SDK 可供使用。

## NOTE

安装 SDK 时,请记住要安装到的路径。例如,“C:\Program Files\Azure Kinect Body Tracking SDK 1.0.0”。你将要在此路径中查找文章中参考的示例。人体跟踪示例位于 Azure-Kinect-Samples 存储库的 [body-tracking-samples](#) 文件夹中。你将找到这些文章中参考的示例。

# 更改日志

## v1.0.1

- [Bug 修复] 修复了从 Windows 内部版本 19025 或更高版本的路径加载 onnxruntime.dll 时 SDK 崩溃的问题:[链接](#)。

## v1.0.0

- [功能] 为 msi 安装程序添加了 C# 包装器。
- [Bug 修复] 修复了无法正确检测头旋转的问题:[链接](#)。
- [Bug 修复] 修复了在 Linux 计算机上 CPU 使用率高达 100% 的问题:[链接](#)。
- [示例] 将两个示例添加到示例存储库中。示例 1 演示了如何将深度空间中的人体跟踪结果转换为颜色空间:[链接](#);示例 2 演示了如何检测地面:[链接](#)。

## v0.9.5

- [功能] C# 支持。C# 包装器已打包在 nuget 包中。
- [功能] 多跟踪器支持。允许创建多个跟踪器。现在, 用户可以创建多个跟踪器, 以从不同的 Azure Kinect 设备跟踪人体。
- [功能] CPU 模式的多线程处理支持。在 CPU 模式下运行时, 将使用所有核心来最大程度地提高速度。
- [功能] 将 `gpu_device_id` 添加到 `k4abt_tracker_configuration_t` 结构。允许用户指定非默认的 GPU 设备来运行人体跟踪算法。
- [Bug 修复/中断性变更] 修正了关节名称的拼写错误。将关节名称从 `K4ABT_JOINT_SPINE_NAVAL` 更改为 `K4ABT_JOINT_SPINE_NAVAL`。

## v0.9.4

- [功能] 添加手关节支持。SDK 将为每只手提供三个附加关节的信息:手、手指尖、拇指。
- [功能] 为每个检测到的关节添加预测置信度。
- [功能] 添加 CPU 模式支持。通过更改 `k4abt_tracker_configuration_t` 中的 `cpu_only_mode` 值, 现在 SDK 可以在 CPU 模式下运行, 这不需要用户具有功能强大的图形卡。

## v0.9.3

- [功能] 发布新的 DNN 模型 `dnn_model_2_0.onnx`, 该模型大大提高了人体跟踪的可靠性。
- [功能] 默认情况下禁用时间平滑。跟踪的关节会响应更迅速。
- [功能] 提高了人体索引映射的准确性。
- [Bug 修复] 修复了传感器方向设置无效的 bug。
- [Bug 修复] 将 `body_index_map` 类型从 `K4A_IMAGE_FORMAT_CUSTOM` 更改为 `K4A_IMAGE_FORMAT_CUSTOM8`。
- [已知问题] 两个相近的身体可能合并成单个实体部分。

## v0.9.2

- [中断性变更] 更新为依赖最新的 Azure Kinect 传感器 SDK 1.2.0。
- [API 更改] `k4abt_tracker_create` 函数将开始接受 `k4abt_tracker_configuration_t` 输入。
- [API 更改] 将 `k4abt_frame_get_timestamp_usec` API 更改为 `k4abt_frame_get_device_timestamp_usec`, 使其更具体且与传感器 SDK 1.2.0 保持一致。
- [功能] 允许用户在创建跟踪器时指定传感器安装方向, 以不同角度安装可获得更准确的人体跟踪结果。

- [功能] 提供了新的 API `k4abt_tracker_set_temporal_smoothing` 以更改用户想要应用的时间平滑量。
- [功能] 添加了 C++ 包装器 `k4abt.hpp`。
- [功能] 添加了版本定义头文件 `k4abtversion.h`。
- [Bug 修复] 修复了导致 CPU 使用率极高的 bug。
- [Bug 修复] 修复了记录器崩溃 bug。

#### v0.9.1

- [Bug 修复] 修复了销毁跟踪器时的内存泄漏
- [Bug 修复] 改进了缺少依赖项的错误消息
- [Bug 修复] 创建第二个跟踪器实例时失败而不崩溃
- [Bug 修复] 记录器环境变量现在可正常工作
- Linux 支持

#### v0.9.0

- [中断性变更] 将 SDK 依赖项降级到 CUDA 10.0(从 CUDA 10.1)。ONNX 运行时最高仅正式支持 CUDA 10.0 版本。
- [中断性变更] 已切换到 ONNX 运行时，弃用了 Tensorflow 运行时。这可以减少第一帧的启动时间和内存用量。此外还可以减小 SDK 二进制大小。
- [API 更改] 已将 `k4abt_tracker_queue_capture()` 重命名为 `k4abt_tracker_enqueue_capture()`。
- [API 更改] 已将 `k4abt_frame_get_body()` 划分为两个单独的函数：`k4abt_frame_get_body_skeleton()` 和 `k4abt_frame_get_body_id()`。现在，无需复制整个主干结构即可查询人体 ID。
- [API 更改] 添加了 `k4abt_frame_get_timestamp_usec()` 函数用于简化用户查询人体帧时间戳的步骤。
- 进一步改进了人体跟踪算法的准确性和跟踪可靠性。

## 后续步骤

- [Azure Kinect DK 概述](#)
- [设置 Azure Kinect DK](#)
- [设置 Azure Kinect 人体跟踪](#)

# Azure Kinect 传感器 SDK 系统要求

2020/8/11 • [Edit Online](#)

本文档详细说明安装传感器 SDK 以及成功部署 Azure Kinect DK 所要满足的系统要求。

## 支持的操作系统和体系结构

- Windows 10 的 2018 年 4 月(版本 1803, 操作系统内部版本 17134)发行版(x64)或更高版本
- Linux Ubuntu 18.04(x64), 其中包含使用 OpenGLv 4.4 或更高版本的 GPU 驱动程序

传感器 SDK 适用于本机 C/C++ Windows 应用程序的 Windows API (Win32)。该 SDK 目前不适用于 UWP 应用程序。S 模式的 Windows 10 不支持 Azure Kinect DK。

## 开发环境要求

若要为传感器 SDK 开发供稿, 请访问 [GitHub](#)。

## 主机电脑的最低硬件要求

电脑主机的硬件要求取决于在主机电脑上执行的应用程序/算法/传感器帧速率/分辨率。对 Windows 建议的最低传感器 SDK 配置为:

- 第七代 Intel® Core™ i3 处理器(双核 2.4 GHz, 搭载 HD620 GPU 或更快的 GPU)
- 4 GB 内存
- 专用 USB3 端口
- 支持 OpenGL 4.4 或 DirectX 11.0 的图形驱动程序

根据具体的用例, 低端或早期的 CPU 可能也适用。

此外, 性能根据所用的 Windows/Linux 操作系统和图形驱动程序而异。

## 人体跟踪主机电脑的硬件要求

人体跟踪电脑主机的要求比一般电脑主机的要求更高。对 Windows 建议的最低人体跟踪 SDK 配置为:

- 第七代 Intel® Core™ i5 处理器(四核 2.4 GHz 或更快)
- 4 GB 内存
- NVIDIA GEFORCE GTX 1070 或更佳
- 专用 USB3 端口

建议的最低配置假定 K4A\_DEPTH\_MODE\_NFOV\_UNBINNED 深度模式以 30fps 的速度跟踪 5 个人。根据具体的用例, 低端或早期的 CPU 和 NVIDIA GPU 可能也适用。

## USB3

USB 主控制器存在一些已知的兼容性问题。可以在[故障排除页](#)上找到详细信息

## 后续步骤

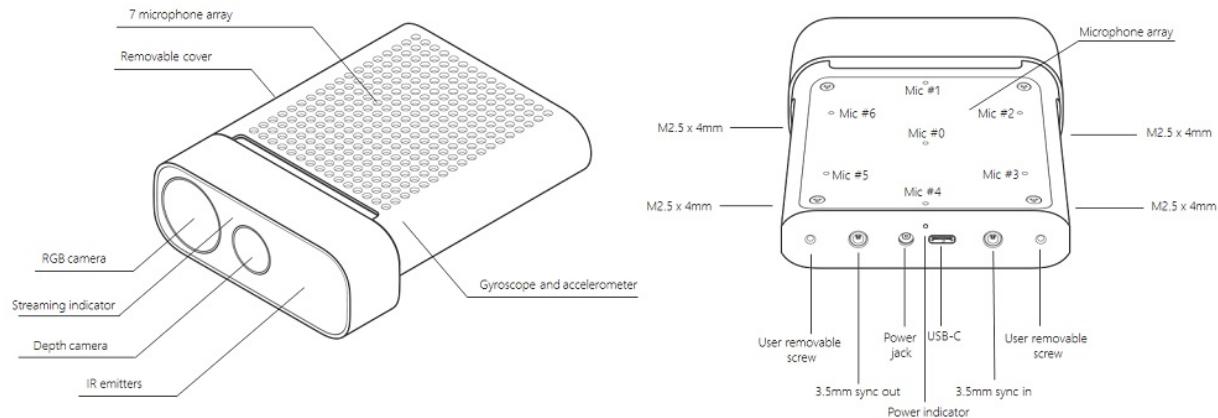
- [Azure Kinect DK 概述](#)
- [设置 Azure Kinect DK](#)

- 设置 Azure Kinect 人体跟踪

# Azure Kinect DK 硬件规格

2020/4/28 • [Edit Online](#)

本文详细说明 Azure Kinect 硬件如何将 Microsoft 的最新传感器技术集成到单个已连接 USB 的附件。



## 术语

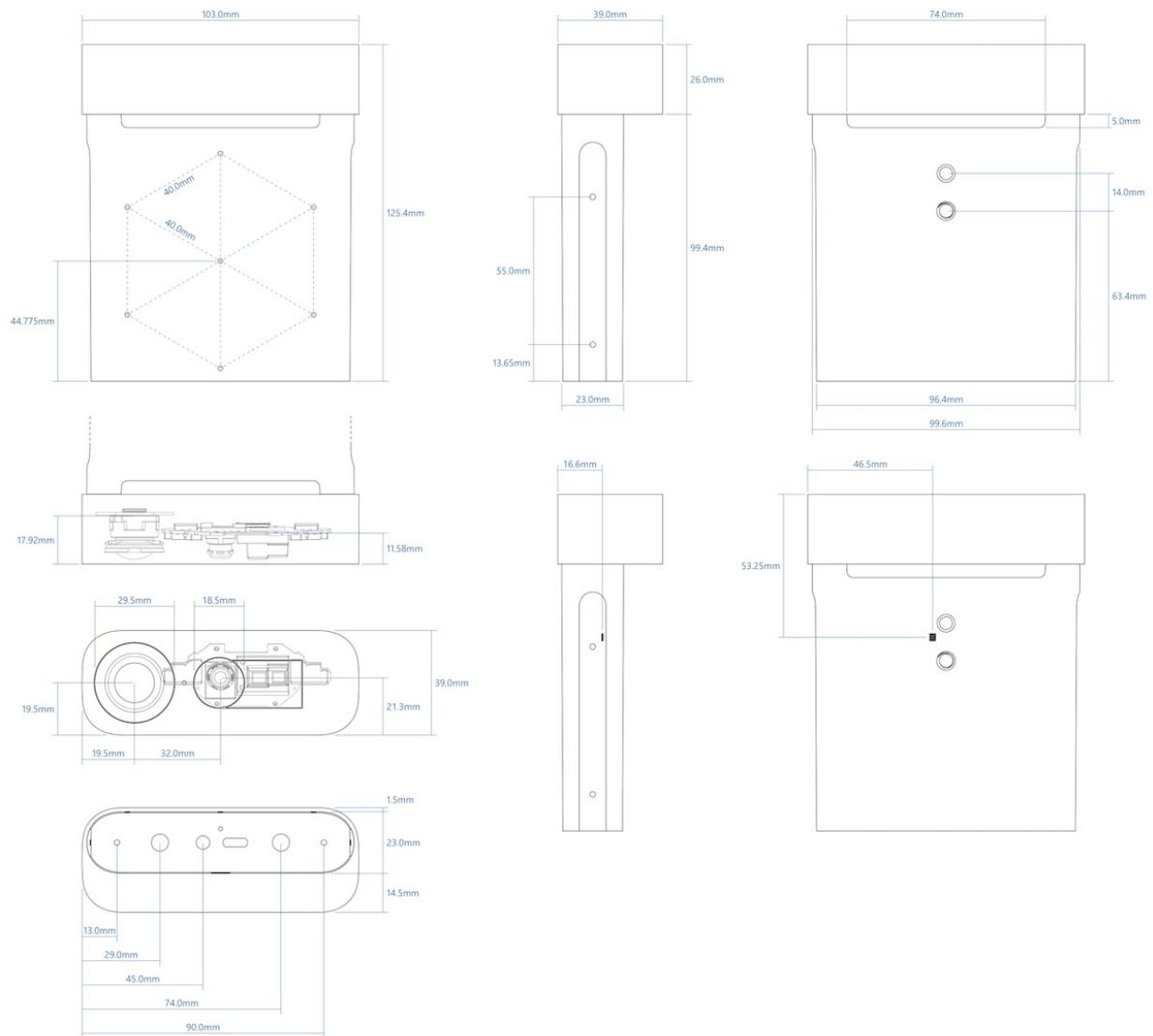
整篇文章使用了以下缩写术语。

- NFOV(窄视场深度模式)
- WFOV(宽视场深度模式)
- FOV(视场)
- FPS(每秒帧数)
- IMU(惯性测量单元)
- FoI(感兴趣的字段)

## 产品尺寸和重量

Azure Kinect 设备的尺寸和重量如下。

- **维度**: 103 x 39 x 126 毫米
- **重量**: 440 克



## 工作环境

Azure Kinect DK 适用于在以下环境条件下工作的开发人员和商业单位：

- **温度** : 10-25°C
- **湿度** : 8-90% (非冷凝) 相对湿度

### NOTE

不符合上述环境条件可能会导致设备出现故障和/或不正常运行。这些环境条件适用于设备在所有工作条件下运行时的邻近周边环境。如果配合外部机箱使用，我们建议使用有效的温度控制和/或其他散热解决方案，来确保设备工作条件保持在这些范围内。本设备在前端型面与后端套管之间设计了一个散热通道。使用本设备时，请确保不要阻挡此散热通道。

请参阅附加的产品[安全信息](#)。

## 深度相机支持的工作模式

Azure Kinect DK 集成了 Microsoft 设计的 1 兆像素时差测距 (ToF) 深度相机，该相机使用[符合 ISSCC 2018 的图像传感器](#)。深度相机支持如下所述的模式：

"ff"	ffff	FOI	FPS	ffff*	ffff
NFOV 非装箱	640x576	75°x65°	0、5、15、30	0.5 - 3.86 米	12.8 毫秒

“ <b>分辨率</b>	<b>帧率</b>	<b>FOV</b>	<b>FPS</b>	<b>深度*</b>	<b>延迟</b>
NFOV 2x2 装箱 (SW)	320x288	75°x65°	0、5、15、30	0.5 - 5.46 米	12.8 毫秒
WFOV 2x2 装箱	512x512	120°x120°	0、5、15、30	0.25 - 2.88 米	12.8 毫秒
WFOV 非装箱	1024x1024	120°x120°	0、5、15	0.25 - 2.21 米	20.3 毫秒
被动 IR	1024x1024	空值	0、5、15、30	空值	1.6 毫秒

\*850nm 时 15% 到 95% 的反射率,  $2.2 \mu\text{W}/\text{cm}^2/\text{nm}$ , 随机误差标准偏差  $\leq 17 \text{ mm}$ , 典型系统误差  $< 11 \text{ mm} + 0.1\%$  的距离(无多路径干扰)。可以在上面指示的操作范围之外提供深度。这取决于对象的反射率。

## 彩色相机支持的工作模式

Azure Kinect DK 包含 OV12A10 12MP CMOS 滚动快门传感器。下面列出了本机工作模式：

RGB 分辨率 (HxV)	宽	高	帧率 (FPS)	FOV (HxV) (mm)
3840x2160	16:9	MJPEG	0、5、15、30	90°x59°
2560x1440	16:9	MJPEG	0、5、15、30	90°x59°
1920x1080	16:9	MJPEG	0、5、15、30	90°x59°
1280x720	16:9	MJPEG/YUY2/NV12	0、5、15、30	90°x59°
4096x3072	4:3	MJPEG	0、5、15	90°x74.3°
2048x1536	4:3	MJPEG	0、5、15、30	90°x74.3°

RGB 相机与 USB 视频类兼容, 可以在未安装传感器 SDK 的情况下使用。RGB 相机颜色空间: BT.601 全范围 [0..255]。

### NOTE

传感器 SDK 能够以 BGRA 像素格式提供彩色图像。这并非设备支持的本机模式, 如果使用, 会导致 CPU 负载增大。主机 CPU 用于转换从设备收到的 MJPEG 图像。

## RGB 相机曝光时间值

以下是可接受的 RGB 相机手动曝光值的映射:

EXP	$2^{\text{EXP}}$	50Hz	60Hz
-11	488	500	500
-10	977	1250	1250
-9	1953	2500	2500

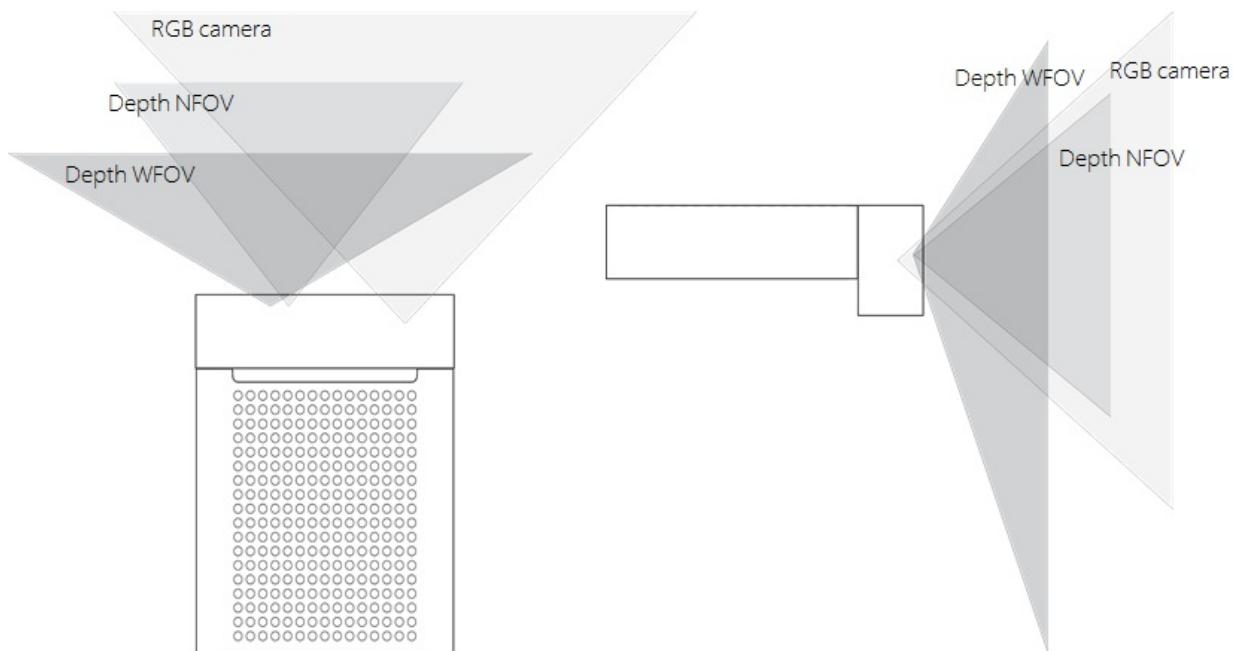
EXP	$2^{\wedge} EXP$	50HZ	60HZ
-8	3906	10000	8330
-7	7813	20000	16670
-6	15625	30000	33330
-5	31250	40000	41670
-4	62500	50000	50000
-3	125000	60000	66670
-2	250000	80000	83330
-1	500000	100000	100000
0	1000000	120000	116670
1	2000000	130000	133330

## 深度传感器原始计时

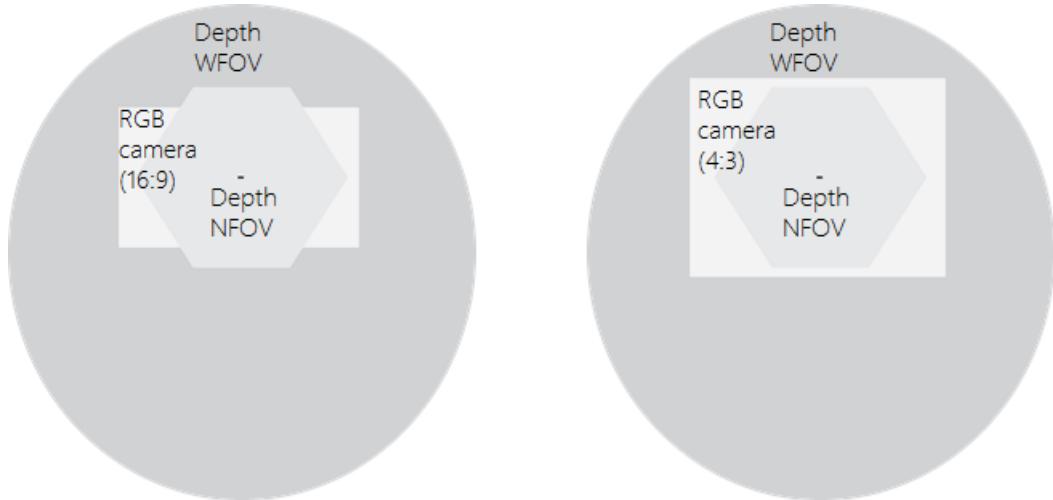
	IR 				
NFOV 非装箱 NFOV 2xx 装箱 WFOV 2x2 装箱	9	125 微秒	8	1450 微秒	12.8 毫秒
WFOV 非装箱	9	125 微秒	8	2390 微秒	20.3 毫秒

## 相机视场

下图显示了深度和 RGB 相机视场(传感器“看到”的视角)。下图显示了 4:3 模式的 RGB 相机。



此图显示了相机在正面 2000 mm 距离的视场。



#### NOTE

如果深度采用 NFOV 模式, RGB 相机在 4:3 模式下的像素重叠性能优于 16:9 模式。

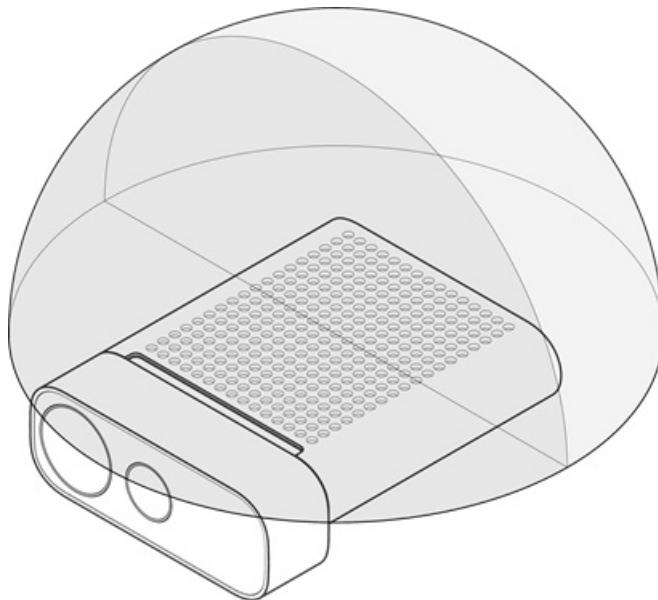
## 运动传感器 (IMU)

嵌入式惯性测量单元 (IMU) 为 LSM6DSMUS, 包含加速度传感器和陀螺仪。加速度传感器和陀螺仪同时按 1.6 kHz 采样。样本以 208 Hz 的频率报告给主机。

## 麦克风阵列

Azure Kinect DK 中嵌入了被视为标准 USB 音频类 2.0 设备的优质七麦克风环形阵列。可以访问所有 7 个通道。  
性能规格：

- 灵敏度 : -22 dBFS (94 dB SPL, 1 kHz)
- 信噪比 > 65 dB
- 声学过载点 : 116 dB



## USB

Azure Kinect DK 是一个 USB3 组合设备，它会向操作系统公开以下硬件终结点：

供应商 ID 为 0x045E (Microsoft)。产品 ID 表如下所示：

USB ID	PNP ID	
USB3.1 第 1 代集线器	0x097A	主集线器
USB2.0 集线器	0x097B	HS USB
深度相机	0x097C	USB3.0
彩色相机	0x097D	USB3.0
麦克风	0x097E	HS USB

## 指示灯

设备的正面配备了相机流指示灯，可以使用传感器 SDK 以编程方式将其禁用。

设备背面的状态 LED 指示设备状态：

LED 状态	含义
稳定白色	设备已打开并正常工作。
闪烁白色	设备已打开，但未建立 USB 3.0 数据连接。
闪烁琥珀色	设备电量不足，无法正常运行。
交替闪烁琥珀色和白色	正在进行固件更新或恢复

## 设备供电

可通过两种方式为设备供电：

1. 使用随附的电源。电源连接器的外径为 4.5 毫米，内径为 3.0 毫米，引脚直径为 0.6 毫米。
2. 使用 Type-C 转 Type-C 线缆供电和传输数据。

Azure Kinect DK 未随附 Type-C 转 Type-C 线缆。

#### NOTE

- 随附的电源线是 USB Type-A 转桶形单柱连接器。使用此线缆时请结合随附的墙上电源。两个标准 USB Type-A 端口提供的电量并不足以满足本设备的消耗。
- USB 线缆非常重要，我们建议使用优质线缆，并在远程部署本单元之前验证功能。

#### TIP

选择良好的 Type-C 转 Type-C 线缆：

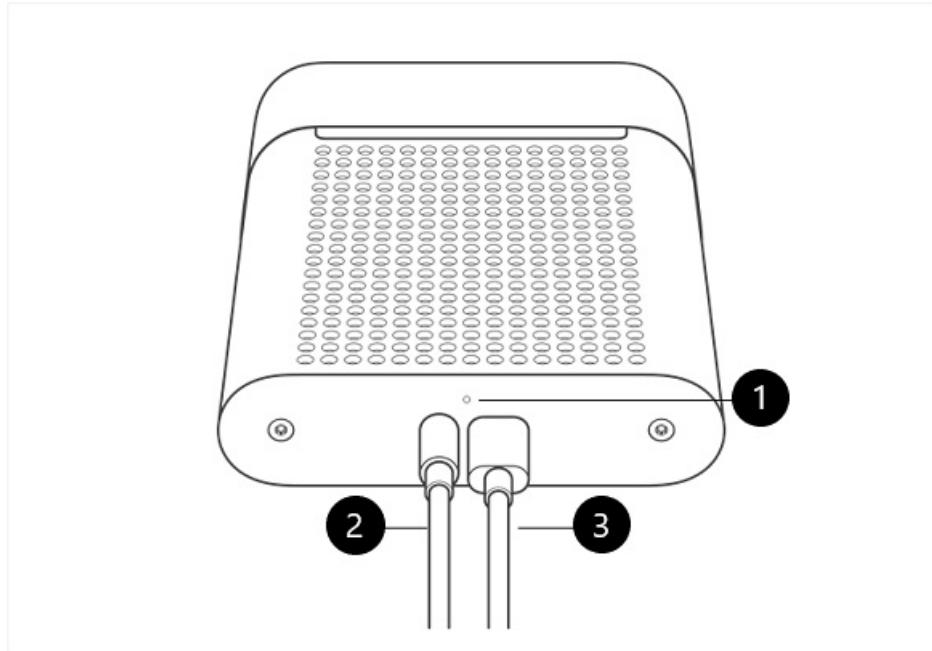
- **USB 认证的线缆**必须支持供电和数据传输。
- 无源线缆的长度应小于 1.5 米。如果更长，请使用有源线缆。
- 线缆至少需要能够支持 1.5A 的电流。否则，需要连接外部电源。

检查线缆：

- 使用线缆将设备连接到主机电脑。
- 验证所有设备是否在 Windows 设备管理器中正确列出。深度相机和 RGB 相机应如以下示例中所示列出。
  - ▼  Universal Serial Bus devices
    -  Azure Kinect 4K Camera
    -  Azure Kinect Depth Camera
- 在 Azure Kinect 查看器中使用以下设置，验证线缆是否能够可靠地流式传输所有传感器的数据：
  - 深度相机：NFOV 非装箱
  - RGB 相机：2160p
  - 麦克风和 IMU 已启用

## 指示灯的含义是什么？

电源指示灯是 Azure Kinect DK 背面的 LED。该 LED 的颜色根据设备的状态而变化。



此图标记了以下组件：

1. 电源指示灯
2. 电源线(连接电源)
3. USB-C 数据线(连接电脑)

请确保如图所示连接线缆。然后，查看下表了解电源指示灯各种状态的含义。

状态:	含义:	操作:
稳定白色	设备已打开电源且在工作正常。	使用设备。
未点亮	设备未连接到电脑。	<p>确保圆形电源连接器线缆已连接到设备和 USB 电源适配器。</p> <p>确保将 USB 数据线连接到设备和电脑。</p>
闪烁白色	设备已打开电源，但尚未建立 USB 3.0 数据连接。	<p>确保圆形电源连接器线缆已连接到设备和 USB 电源适配器。</p> <p>确保将 USB-C 数据线连接到设备和电脑上的 USB 3.0 端口。</p> <p>将设备连接到电脑上的另一个 USB 3.0 端口。</p> <p>在电脑上打开设备管理器（“启动”&gt;“控制面板”&gt;“设备管理器”），检查电脑上是否装有受支持的 USB 3.0 主机控制器。</p>
闪烁琥珀色	设备电量不足，无法正常运行。	<p>确保圆形电源连接器线缆已连接到设备和 USB 电源适配器。</p> <p>确保将 USB 数据线连接到设备和电脑。</p>
琥珀色，然后闪烁白色	设备已打开电源并正在接收固件更新，或者设备正在还原出厂设置。	等待电源指示灯变为稳定白色。有关详细信息，请参阅 <a href="#">重置 Azure Kinect DK</a> 。

# 功耗

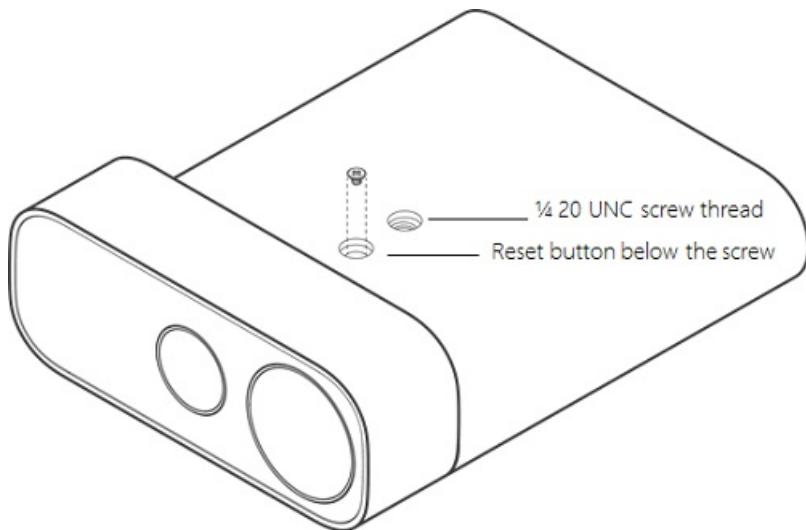
Azure Kinect DK 的最大功率为 5.9W; 具体的功耗与用例相关。

# 校准

Azure Kinect DK 在出厂之前已校准。可通过传感器 SDK 以编程方式查询视觉传感器和惯性传感器的校准参数。

# 设备恢复

可以使用定位销下方的按钮将设备固件重置为原始固件。



若要恢复设备，请参阅[此处的说明](#)。

# 后续步骤

- [使用 Azure Kinect 传感器 SDK](#)
- [设置硬件](#)

# 同步多个 Azure Kinect DK 设备

2020/8/11 • [Edit Online](#)

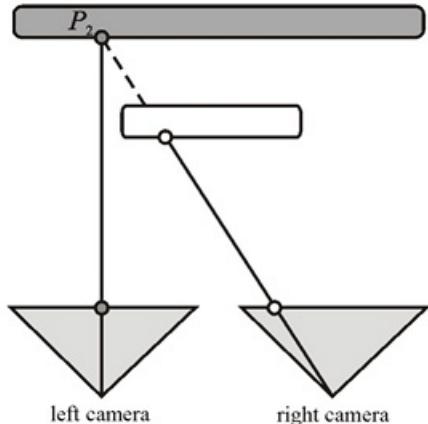
每个 Azure Kinect DK 设备附带 3.5 毫米同步端口(输入同步和输出同步), 可将多个设备链接在一起。连接设备后, 软件可以协调设备之间的触发定时。

本文将介绍如何连接和同步设备。

## 使用多个 Azure Kinect DK 设备的好处

使用多个 Azure Kinect DK 设备的原因有很多, 包括:

- 填补遮挡区域。尽管 Azure Kinect DK 数据转换生成的是单个图像, 但两个相机(深度和 RGB 相机)实际上保持着较小的一段距离。这种偏移使得遮挡成为可能。遮挡是指前景对象阻挡了设备上两个相机之一的背景对象的部分视角。在生成的彩色图像中, 前景对象看上去像是在背景对象上投射了一个阴影。  
例如, 在下图中, 左侧相机可看到灰色像素“P2”。但是, 白色的前景对象阻挡了右侧相机的红外光束。右侧相机无法获取“P2”的数据。



附加的同步设备可以提供遮挡的数据。

- 扫描三维对象。
- 将有效帧速率提升至 30 帧/秒 (FPS) 以上的值。
- 捕获同一场景的多个 4K 彩色图像, 所有图像都在曝光中心时间点的 100 微秒 ( $\mu$ s) 内对齐。
- 增大相机的空间覆盖范围。

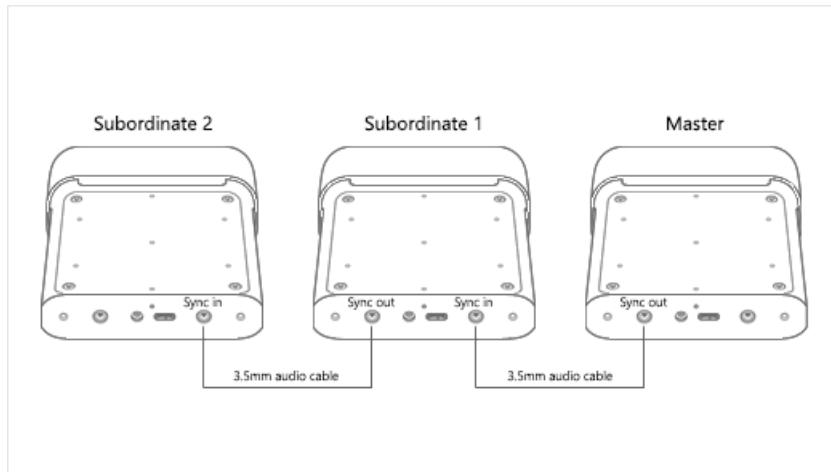
## 规划多设备配置

在开始之前, 请务必查看 [Azure Kinect DK 硬件规格](#) 和 [Azure Kinect DK 深度相机](#)。

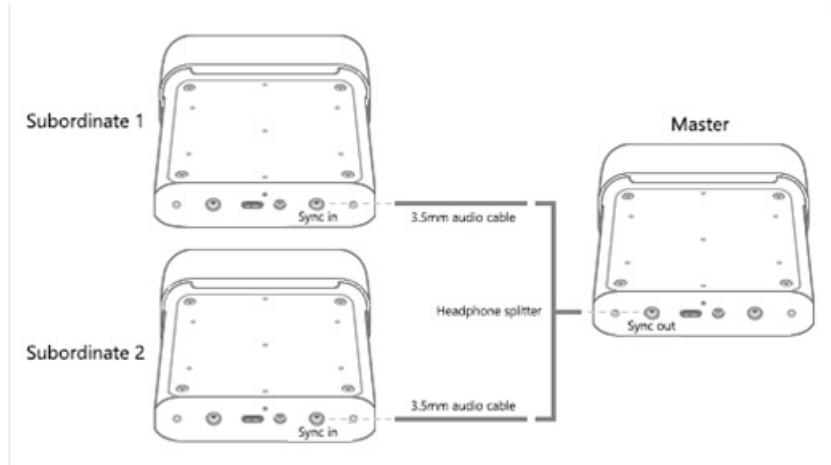
### 选择设备配置

可使用以下任一方法来完成设备配置:

- **菊花链配置。** 同步一个主设备以及最多八个从属设备。



- 星形配置。同步一个主设备以及最多两个从属设备。



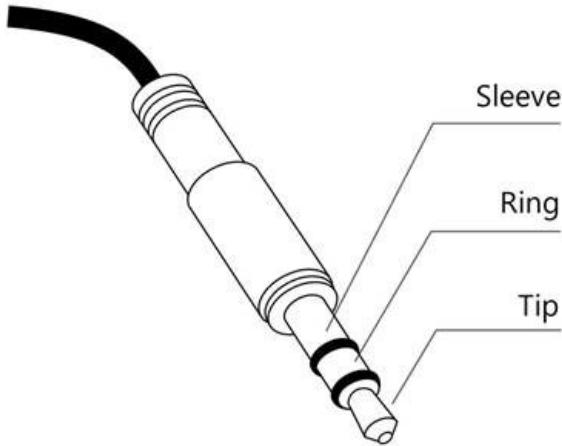
#### 使用外部同步触发器

在两种配置中，主设备提供从属设备的触发信号。但是，可将自定义的外部源用于同步触发器。例如，可以使用此选项同步其他设备的图像捕获信号。在菊花链配置或星形配置中，外部触发器源将连接到主设备。

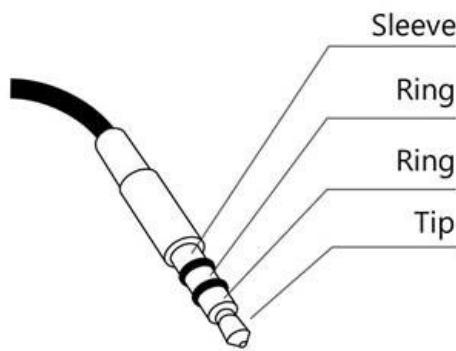
外部触发器源的工作方式必须与主设备相同。它必须提供一个具有以下特征的同步信号：

- 活动程度高
- 脉冲宽度：大于 8 $\mu$ s
- 5V TTL/CMOS
- 最大驱动容量：不小于 8 毫安 (mA)
- 频率支持：精确到 30 FPS、15 FPS 和 5 FPS (彩色相机主 VSYNC 信号的频率)

触发器源必须使用 3.5 毫米音频线将信号传送到主设备的输入同步端口。可以使用立体声或单声道音频线。Azure Kinect DK 会将音频线连接器的所有套管和套环短接到一起，并将其接地。如下图所示，设备只从连接器尖端接收同步信号。



TRS Audio Plug



TRRS Audio Plug

有关如何使用外部设备的详细信息, 请参阅[将 Azure Kinect 录制器与外部同步设备配合使用](#)

### 规划相机设置和软件配置

有关如何设置软件来控制相机以及如何使用图像数据的信息, 请参阅[Azure Kinect 传感器 SDK](#)。

本部分讨论影响已同步设备(而不是单独的设备)的多种因素。软件应考虑到这些因素。

#### 曝光考虑因素

若要控制每个设备的精确定时, 我们建议使用手动曝光设置。使用自动曝光设置时, 每个彩色相机可能会动态更改实际曝光。由于曝光会影响定时, 此类更改很快就会使相机失去同步。

避免在图像捕获循环中重复指定相同的曝光设置。必要时, 请只调用 API 一次。

#### 避免多个深度相机之间产生干扰

如果多个深度相机对重叠的视场成像, 每个相机必须对其自身关联的激光成像。为了防止激光相互干扰, 相机捕获应相互偏离 160 $\mu$ s 或以上。

对于每次深度相机捕获, 激光会打开 9 次, 每次只保持活动状态 125 $\mu$ s。然后, 激光将空闲 14505 $\mu$ s 或 23905 $\mu$ s, 具体取决于工作模式。此行为意味着, 偏移量计算的起点为 125 $\mu$ s。

此外, 相机时钟与设备固件时钟之差会将最小偏移量增大至 160 $\mu$ s。若要根据配置计算出更精确的偏移量, 请注意所用的深度模式, 并参考[深度传感器原始定时表](#)。参考此表中的数据可以使用以下公式计算最小偏移量(每个相机的曝光时间):

$$\text{曝光时间} = (\text{红外脉冲} \times \text{脉冲宽度}) + (\text{空闲周期} \times \text{空闲时间})$$

使用 160 $\mu$ s 偏移量时, 最多可以配置 9 个额外的深度相机, 以便在每束激光打开时, 其他激光保持空闲状态。

在软件中, 使用 `depth_delay_off_color_usec` 或 `subordinate_delay_off_master_usec` 来确保每束红外激光在其自身的 160 $\mu$ s 时限内触发, 或者提供不同的视场。

## 准备设备和其他硬件

除了配置多个 Azure Kinect DK 设备以外, 可能还需要获得其他主机和其他硬件才能支持你要构建的配置。在开始设置之前, 请使用本部分中的信息来确保所有设备和硬件已准备就绪。

### Azure Kinect DK 设备

对于要同步的每个 Azure Kinect DK 设备, 请采取以下措施:

- 确保设备上已安装最新的固件。有关如何更新设备的详细信息, 请参阅[更新 Azure Kinect DK 固件](#)。

- 卸下设备护盖，露出同步端口。
- 记下每个设备的序列号。在稍后的设置过程中需使用此编号。

## 主机

每个 Azure Kinect DK 通常使用自身的主机。可以根据设备的使用方式以及通过 USB 连接传输的数据量使用专用主机控制器。

确保每台主机上安装了 Azure Kinect 传感器 SDK。有关如何安装传感器 SDK 的详细信息，请参阅[快速入门：设置 Azure Kinect DK](#)。

### Linux 计算机：Ubuntu 上的 USB 内存

默认情况下，基于 Linux 的主机只为 USB 控制器分配 16 MB 的内核内存用于处理 USB 传输。此内存量通常足以支持单个 Azure Kinect DK。但是，若要支持多个设备，USB 控制器必须有更多的内存。若要增大内存，请执行以下步骤：

1. 编辑 `/etc/default/grub`。
2. 找到以下行：

```
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash"
```

替换为以下行：

```
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash usbcore.usbfs_memory_mb=32"
```

#### NOTE

这些命令会将 USB 内存设置为 32 MB。此示例设置是默认值的两倍。可以设置一个大得多的值，只要适合解决方案即可。

3. 运行 `sudo update-grub`。

4. 重新启动计算机。

## 电缆

若要将设备相互连接并连接到主机，必须使用 3.5 毫米公对公线缆（也称为 3.5 毫米音频线）。线缆长度应小于 10 米，可以是立体声或单声道音频线。

所需的线缆数取决于所用的设备数以及具体的设备配置。Azure Kinect DK 机箱中未随附线缆。必须单独购买线缆。

如果在星形配置中连接设备，则还需要一个耳机分线器。

## 连接数据

### 在菊花链配置中连接 Azure Kinect DK 设备

1. 将每个 Azure Kinect DK 连接到电源。
2. 将每个设备连接到其自身的主机。
3. 选择一个设备充当主设备，并将 3.5 毫米音频线插入该设备的输出同步端口。
4. 将该线缆的另一端插入第一个从属设备的输入同步端口。
5. 若要连接另一个设备，请将另一根线缆插入第一个从属设备的输出同步端口，以及下一个设备的输入同步端口。
6. 重复上述步骤，直到所有设备都已连接。最后一个设备应连接了一根线缆。其输出同步端口应该是空的。

### 在星形配置中连接 Azure Kinect DK 设备

1. 将每个 Azure Kinect DK 连接到电源。
2. 将每个设备连接到其自身的主机。
3. 选择一个设备充当主设备，将耳机分线器的单体端插入其输出同步端口。
4. 将 3.5 毫米音频线连接到耳机分线器的“分接”端。
5. 将每根线缆的另一端插入某个从属设备的输入同步端口。

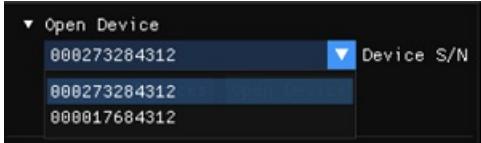
## 验证设备是否已连接并可通信

若要验证设备是否已正确连接，请使用 [Azure Kinect 查看器](#)。根据需要重复此过程，以结合主设备测试每个从属设备

### IMPORTANT

在测试过程中，必须知道每个 Azure Kinect DK 的序列号。

1. 打开 Azure Kinect 查看器的两个实例。
2. 在“打开设备”下，选择要测试的从属设备的序列号。



### IMPORTANT

若要使所有设备的图像捕获保持精确同步，必须最后启动主设备。

3. 在“外部同步”下，选择“从属设备”。



4. 选择“开始”。

**NOTE**

由于这是一个从属设备，在设备启动后，Azure Kinect 查看器不会显示图像。在从属设备收到主设备发出的同步信号之前不会显示图像。

5. 启动从属设备后，使用 Azure Kinect 查看器的另一实例打开主设备。

6. 在“外部同步”下，选择“主设备”。

7. 选择“开始”。

Azure Kinect 主设备启动后，Azure Kinect 查看器的两个实例应显示图像。

## 根据同步设置校准设备

验证设备可正确通信后，接下来可对其进行校准，以便在单个域中生成图像。

在单个设备中，深度相机和 RGB 相机已经过出厂校准，可以协同工作。但是，如果必须一同使用多个设备，则必须对这些设备进行校准，以确定如何将图像从捕获它时所在的相机域转换为用于处理图像的相机域。

可以使用多个选项来交叉校准设备。Microsoft 提供了使用 OpenCV 方法的 [GitHub 绿屏代码示例](#)。此代码示例的自述文件提供了有关校准设备的更多详细信息和说明。

有关校准的详细信息，请参阅[使用 Azure Kinect 校准功能](#)。

## 后续步骤

设置已同步的设备后，还可以了解如何使用

Azure Kinect 传感器 SDK 录制和播放 API

## 相关主题

- [关于 Azure Kinect 传感器 SDK](#)
- [Azure Kinect DK 硬件规格](#)
- [快速入门：设置 Azure Kinect DK](#)
- [更新 Azure Kinect DK 固件](#)
- [重置 Azure Kinect DK](#)
- [Azure Kinect 查看器](#)

# Azure Kinect 与 Kinect Windows v2 的比较

2020/8/11 • [Edit Online](#)

Azure Kinect DK 硬件和软件开发工具包与 Kinect for Windows v2 之间有差别。任何现有的 Kinect for Windows v2 应用程序不能直接与 Azure Kinect DK 配合工作，需要移植到新的 SDK。

## 硬件

下表列出了 Azure Kinect 开发工具包与 Kinect for Windows v2 之间的大致差别。

		AZURE KINECT DK	KINECT FOR WINDOWS V2
■	详细信息	7 麦克风环形阵列	4 麦克风线性相控阵列
■	详细信息	3 轴加速度传感器, 3 轴陀螺仪	3 轴加速度传感器
RGB ■	详细信息	3840 x 2160 像素 @30 fps	1920 x 1080 像素 @30 fps
■	方法	时差测距	时差测距
	解决方法	640 x 576 像素 @30 fps	512 x 424 像素 @ 30 fps
		512 x 512 像素 @30 fps	
		1024x1024 像素 @15 fps	
■	数据	USB3.1 Gen 1, 附带 USB Type-C	USB 3.1 Gen 1
	电源	外部 PSU 或 USB-C	外部 PSU
	同步	RGB 和深度内部同步, 外部设备到设备同步	仅限 RGB 和深度内部同步
■	维度	103 x 39 x 126 毫米	249 x 66 x 67 毫米
	重量	440 克	970 克
	安装	一颗 1/4-20 UNC 螺丝。四个内部螺丝固定点	一颗 1/4-20 UNC 螺丝

在 [Azure Kinect 硬件](#) 文档中查找更多详细信息。

## 传感器访问

下表提供了低级别设备传感器访问功能的比较。

AZURE KINECT	KINECT FOR WINDOWS	
Depth	✓	
IR	✓	
■	✓	颜色格式支持差异, Azure Kinect 深色支持以下相机控件:曝光度、白平衡、亮度、对比度、饱和度、清晰度和增益控制
■:	✓	通过语音 SDK 或 Windows 本机 API 访问 Azure Kinect DK 麦克风
IMU	✓	Azure Kinect DK 有完整的 6 轴 IMU, Kinect for Windows 仅提供单轴 IMU
■	✓	OpenCV 兼容的相机型号校准
■ RGB ■	✓	
■	✓	Azure Kinect DK 允许对外部同步延迟进行编程
■■	✓	Azure Kinect 传感器 SDK 依赖于使用 WinUSB/libUSB 来访问设备, 不会实现某个服务来启用与多个进程共享设备访问
■/■	✓	Azure Kinect DK 使用基于 Matroska 容器的开源实现

## 功能

Azure Kinect SDK 的功能集与 Kinect for Windows v2 不同, 详述如下:

KINECT V2	KINECT V2	AZURE KINECT SDK
传感器数据访问	DepthFrame	传感器 SDK - 检索图像
	InfraredFrame	传感器 SDK - 检索图像
	ColorFrame	传感器 SDK - 检索图像

KINECT V2	KINECT V2	AZURE KINECT SDK/
	AudioBeamFrame	目前不支持
人体跟踪	BodyFrame	人体跟踪 SDK
	BodyIndexFrame	人体跟踪 SDK
协调映射	CoordinateMapper	传感器 SDK - 图像转换
人脸跟踪	FaceFrame	认知服务:面部
语音识别	空值	认知服务:语音

## 后续步骤

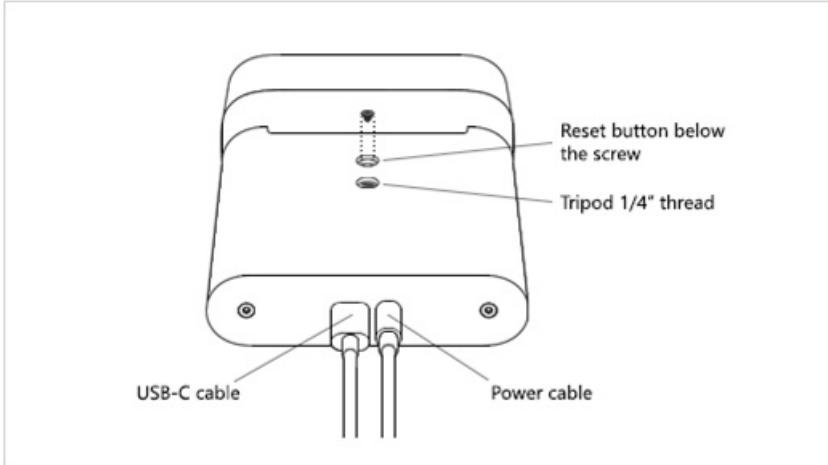
[Kinect for Windows 开发人员页](#)

# 重置 Azure Kinect DK

2020/3/27 • [Edit Online](#)

你可能遇到过需要将 Azure Kinect DK 重置为出厂映像的情况(例如, 在未正确安装固件更新时)。

1. 关闭 Azure Kinect DK 的电源。为此, 请拔下 USB 数据线和电源线。



2. 若要找到重置按钮, 请拆下三脚架安装锁孔中的螺丝。
3. 重新连接电源线。
4. 将拉直的回形针的尖端插入三脚架安装锁孔中的空螺丝孔。
5. 使用回形针轻按重置按钮并按住不放。
6. 在按住重置按钮的同时, 重新连接 USB 数据线。
7. 大约 3 秒后, 电源指示灯会变为琥珀色。指示灯变色后, 松开重置按钮。  
松开重置按钮后, 电源指示灯会以白色和琥珀色闪烁, 此时设备正在重置。
8. 等待电源指示灯变为稳定白色。
9. 在三脚架安装锁孔中装回螺丝, 盖住重置按钮。
10. 使用 Azure Kinect 查看器验证固件是否已重置。为此, 请启动 [Azure Kinect 查看器](#), 选择“设备固件版本信息”查看安装在 Azure Kinect DK 上的固件版本。

请始终确保在设备上安装最新的固件。若要获取最新的固件版本, 请使用 Azure Kinect 固件工具。有关如何检查固件状态的详细信息, 请参阅[检查设备固件版本](#)。

## 相关主题

- [关于 Azure Kinect DK](#)
- [设置 Azure Kinect DK](#)
- [Azure Kinect DK 硬件规格: 工作环境](#)
- [Azure Kinect 固件工具](#)
- [Azure Kinect 查看器](#)
- [跨多个 Azure Kinect DK 设备进行同步](#)

# Azure Kinect 支持选项和资源

2020/8/11 • [Edit Online](#)

本文介绍各个支持选项。

## 社区支持

可以使用多种方式通过公共论坛获得问题的解答：

- [StackOverflow](#): 可在其中提问或搜索现有的问题库。
- [GitHub](#): 可在其中提问、提出新 bug 或者为 Azure Kinect 传感器 SDK 的开发做贡献。
- [提供反馈](#): 可为将来的的产品开发分享看法，以及为现有的看法投票。

## 辅助支持

可通过多种方式获取 Azure Kinect 的支持。

### Microsoft 问答

要快速从 Microsoft 工程师、Azure 最有价值专家 (MVP) 或我们的专家社区那里获得技术产品问题的可靠答案，请在 [Microsoft 问答](#) 上与我们联系 - 这是 Azure 的首选社区支持位置。

- [Microsoft Q&A 适用于 Azure Kinect](#), 你可以在其中提问或搜索现有的问题库。

### Azure 上的开发 Azure Kinect

Azure 订阅者可在 Azure 门户中创建和管理支持请求。Azure 订阅者可以使用与其订阅关联的 [Azure 支持计划](#) 获得人体跟踪、传感器 SDK、语音设备 SDK 或 Azure 认知服务方面的一对一开发支持。

- 已有一个与 Azure 订阅关联的 [Azure 支持计划](#)？请[登录 Azure 门户](#)提交事件。
- 需要 Azure 订阅？[Azure 订阅选项](#)会提供有关不同选项的详细信息。
- 需要支持计划？[选择支持计划](#)

### Azure Kinect 本地或其他云服务

在本地使用传感器 SDK 和人体跟踪 SDK 时如需技术支持，请在 [Microsoft 支持门户](#) 中开具专业支持票证。

### Azure Kinect DK 设备

在联系硬件支持人员之前，请确保已设置并更新 Azure Kinect DK。若要测试设备是否正常工作，请使用 [Azure Kinect 查看器](#)。在我们的 [Azure Kinect DK 帮助](#) 页上可以找到更多信息。另外建议查看[已知问题和故障排除](#)。

获取有关设备或传感器功能、固件更新或购买选项的[帮助](#)。

有关支持产品/服务的详细信息，请参阅[面向企业的 Microsoft 支持](#)。

## 后续步骤

[Azure Kinect 故障排除](#)

# Azure Kinect 已知问题和故障排除

2020/8/11 • [Edit Online](#)

本页包含在 Azure Kinect DK 中使用传感器 SDK 时可能会遇到的已知问题及其故障排除提示。另请参阅[产品支持页](#)了解特定于产品硬件的问题。

## 已知问题

- ASMedia USB 主控制器(例如 ASM1142 芯片组)的兼容性问题
  - 使用 Microsoft USB 驱动程序可以解决某些问题
  - 许多电脑提供备选的主控制器, 更改 USB3 端口可能会有所帮助

有关其他传感器 SDK 相关问题, 请查看 [GitHub 问题](#)

## 收集日志

可通过环境变量启用 K4A.dll 的日志记录。默认情况下, 日志将发送到 stdout, 并且只会生成错误和关键消息。可以更改这些设置, 以将日志发送到文件。还可以按需调整详细级别。以下示例适用于 Windows, 演示如何将日志记录到名为 k4a.log 的文件, 并捕获警告和更高级别的消息。

1. `set K4A_ENABLE_LOG_TO_A_FILE=k4a.log`
2. `set K4A_LOG_LEVEL=w`
3. 从命令提示符运行方案(例如启动查看器)
4. 导航到 k4a.log 并共享该文件。

有关详细信息, 请参阅头文件中的以下片段:

```
/**  
 * environment variables  
 * K4A_ENABLE_LOG_TO_A_FILE =  
 *     0      - completely disable logging to a file  
 *     log\custom.log - log all messages to the path and file specified - must end in '.log' to  
 *                      be considered a valid entry  
 *     ** When enabled this takes precedence over the value of K4A_ENABLE_LOG_TO_STDOUT  
 *  
 * K4A_ENABLE_LOG_TO_STDOUT =  
 *     0      - disable logging to stdout  
 *     all else - log all messages to stdout  
 *  
 * K4A_LOG_LEVEL =  
 *     'c'    - log all messages of level 'critical' criticality  
 *     'e'    - log all messages of level 'error' or higher criticality  
 *     'w'    - log all messages of level 'warning' or higher criticality  
 *     'i'    - log all messages of level 'info' or higher criticality  
 *     't'    - log all messages of level 'trace' or higher criticality  
 *     DEFAULT - log all message of level 'error' or higher criticality  
 */
```

人体跟踪 SDK K4ABT.dll 的日志记录是类似的, 不同之处在于用户应修改一组不同的环境变量名称:

```

/**
 * environment variables
 * K4ABT_ENABLE_LOG_TO_A_FILE =
 *   0      - completely disable logging to a file
 *   log\custom.log - log all messages to the path and file specified - must end in '.log' to
 *                   be considered a valid entry
 *   ** When enabled this takes precedence over the value of K4A_ENABLE_LOG_TO_STDOUT
 *
 * K4ABT_ENABLE_LOG_TO_STDOUT =
 *   0      - disable logging to stdout
 *   all else - log all messages to stdout
 *
 * K4ABT_LOG_LEVEL =
 *   'c'  - log all messages of level 'critical' criticality
 *   'e'  - log all messages of level 'error' or higher criticality
 *   'w'  - log all messages of level 'warning' or higher criticality
 *   'i'  - log all messages of level 'info' or higher criticality
 *   't'  - log all messages of level 'trace' or higher criticality
 *   DEFAULT - log all message of level 'error' or higher criticality
*/

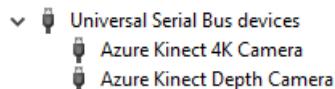
```

## 设备未列在设备管理器中

- 检查设备背面的状态 LED，如果该 LED 闪烁琥珀色，则表示 USB 连接有问题，或者 USB 的供电不足。应将电源线插入随附的电源适配器。尽管电源线已连接到 USB Type-A 端口，但电脑的 USB 端口无法提供设备所需的电量。因此，请不要将设备连接到电脑端口或 USB 集线器。
- 检查是否已连接电源线并使用 USB3 端口来传输数据。
- 尝试改用 USB3 端口来建立数据连接(建议使用靠近主板的 USB 端口，例如，电脑背面的 USB 端口)。
- 检查线缆的状态，受损或劣质的线缆会导致列出的信息不可靠(设备在设备管理器中不断“闪烁”)。
- 如果已连接到笔记本电脑并且该电脑以电池运行，则端口的电量可能会受到限制。
- 重新启动主机电脑。
- 如果问题仍然存在，则可能是存在兼容性问题。
- 如果故障是在固件更新期间发生的，而设备无法自行恢复，请执行[出厂重置](#)。

## Azure Kinect 查看器无法打开

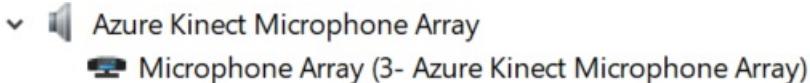
- 先检查设备是否列在 Windows 设备管理器中。



- 检查是否有任何其他应用程序正在使用该设备(例如 Windows 相机应用程序)。每次只能有一个应用程序访问该设备。
- 检查 k4aviewer.err 日志中的错误消息。
- 打开 Windows 相机应用程序，检查它是否可正常工作。
- 关闭再打开设备的电源，等待流 LED 熄灭，然后再使用设备。
- 重新启动主机电脑。
- 确保在电脑上使用最新的图形驱动程序。
- 如果你使用自己的 SDK 版本，请尝试使用正式发布的版本(如果可以解决问题)。
- 如果问题仍然存在，请[收集日志](#)并提交反馈。

## 找不到麦克风

- 先检查麦克风阵列是否列在设备管理器中。
- 如果设备已列出并且在 Windows 中可正常工作，则问题可能在于，在更新固件后，Windows 将不同的容器 ID 分配给了深度相机。
- 可以转到设备管理器，右键单击“Azure Kinect Microphone Array”并选择“卸载设备”来重置设备。完成该操作后，卸下再重新装上传感器。



- 然后，重启 Azure Kinect 查看器并重试。

## 设备固件更新问题

- 如果更新后未报告正确的版本号，则可能需要关闭再打开设备。
- 如果固件更新中断，设备可能会进入错误状态，因此无法列出。卸下再重新装上设备，等待 60 秒再看看它能否恢复。如果不能，请执行[出厂重置](#)。

## 图像质量问题

- 启动 [Azure Kinect 查看器](#)，检查设备的位置，以确定是否存在干扰、传感器被遮挡，或者镜头脏污。
- 如果问题在特定的模式下发生，请尝试不同的运行模式，以缩小问题的检查范围。
- 若要与团队配合解决图像质量问题，可以：
  - 抓取 [Azure Kinect 查看器](#) 的暂停视图并抓取屏幕截图，或
  - 使用 [Azure Kinect 录制器](#) 录制，例如 `k4arecorder.exe -l 5 -r 5 output.mkv`

## 不一致或意外的设备时间戳

调用 `k4a_device_set_color_control` 可能会暂时引发设备的计时更改，这些更改可能需要几个捕获才能稳定。避免在图像捕获循环中调用 API，以免重置每个新图像的内部计时计算。可以改为在启动相机之前或需要更改图像捕获循环内的值时调用 API。具体而言，请避免调用

```
k4a_device_set_color_control(K4A_COLOR_CONTROL_AUTO_EXPOSURE_PRIORITY)。
```

## USB3 主控制器兼容性

如果设备未列在设备管理器中，原因可能是将它插入到了不受支持的 USB3 控制器。

Windows 上的 Azure Kinect DK 仅支持 Intel、Texas Instruments (TI) 和 Renesas 的主控制器。Windows 平台上的 Azure Kinect SDK 依赖于统一的容器 ID，它必须与 USB 2.0 和 3.0 设备兼容，这样，该 SDK 才能找到实际定位在同一设备上的深度、颜色和音频设备。在 Linux 上，可能会支持更多的主控制器，因为该平台对容器 ID 的依赖性较小，而更多地依赖于设备序列号。

当电脑上安装了多个主控制器时，USB 主控制器的话题就会变得更复杂。如果混合使用主控制器，用户可能会遇到问题，有些端口可以正常工作，而其他一些端口则根本无法工作。根据端口在机箱上的布线方式，在使用 Azure Kinect 时，你可能会发现所有(机箱)正面端口都出现问题。

**Windows：** 若要找出现有的主控制器，请打开设备管理器

- “查看”->“依类型排序设备”
- 在连接 Azure Kinect 后，连接“相机”->“Azure Kinect 4K 相机”
- “查看”->“依连接排序设备”



若要更好地了解电脑上已连接哪个 USB 端口, 请在将 Azure Kinect DK 连接到电脑上的不同 USB 端口时重复上述步骤。

## 深度相机自动关机

深度相机用来计算图像深度数据的激光, 其寿命有限。为了最大限度地提高激光寿命, 深度相机会检测何时不会使用深度数据。如果设备流式传输了数据几分钟, 但主机电脑并未读取数据, 则深度相机将会关机。这也会影响多设备同步。此时, 附属设备启动时所处的状态是深度相机正在流式传输数据, 而深度帧已有效挂起, 正在等待主设备开始同步捕获内容。为避免在多设备捕获方案中出现此问题, 请确保主设备在第一个附属设备启动后的一分钟内启动。

## 将人体跟踪 SDK 与 Unreal 配合使用

若要将人体跟踪 SDK 与 Unreal 配合使用, 请确保已将 <SDK Installation Path>\tools 添加到环境变量 PATH, 并已将 dnn\_model\_2\_0.onnx 和 cudnn64\_7.dll 复制到 Program Files/Epic Games/UE\_4.23/Engine/Binaries/Win64。

## 后续步骤

[更多支持信息](#)

# 使用 Azure Kinect 传感器 SDK 录制文件格式

2020/8/11 • [Edit Online](#)

为了记录传感器数据，将使用 Matroska (.mkv) 容器格式，这允许使用各种编解码器存储多个曲目。录制文件包含用于存储颜色、深度、IR 图像和 IMU 的轨道。

在 [Matroska 网站](#) 上可以找到 .mkv 容器格式的大致详细信息。

轨道	描述
COLOR	与模式相关 (MJPEG、NV12 或 YUY2)
DEPTH	b16g(16 位灰度, 大字节序)
IR	b16g(16 位灰度, 大字节序)
IMU	自定义结构。请参阅下面的 <a href="#">IMU 示例结构</a> 。

## 使用第三方工具

可以使用 `ffmpeg` 等工具或者 [MKVToolNix](#) 工具包中的 `mkvinfo` 命令查看和提取录制文件中的信息。

例如，以下命令将深度轨道作为 16 位 PNG 序列提取到同一个文件夹：

```
ffmpeg -i output.mkv -map 0:1 -vsync 0 depth%04d.png
```

`-map 0:1` 参数将提取轨道索引 1，对于大多数录制内容而言，此索引是深度。如果录制内容不包含颜色轨道，则会使用 `-map 0:0`。

`-vsync 0` 参数强制 `ffmpeg` 按原样提取帧，而不是尝试匹配 30 fps、15 fps 或 5 fps 的帧速率。

## IMU 示例结构

如果在不使用播放 API 的情况下从文件中提取 IMU 数据，则数据将采用二进制格式。下面是 IMU 数据的结构。所有字段均为小字节序。

字段	类型
加速度传感器时间戳 (μs)	uint64
加速度传感器数据 (x, y, z)	float[3]
陀螺仪时间戳 (μs)	uint64
陀螺仪数据 (x, y, z)	float[3]

## 识别轨道

可能需要识别哪个轨道包含颜色、深度、IR 等属性。使用第三方工具读取 Matroska 文件时，需要识别轨道。轨道

编号根据相机模式和已启用的轨道集而异。标记用于标识每个轨道的含义。

下面所列的每个标记将附加到特定的 Matroska 元素，可用于查找相应的轨道或附件。

可以使用 `ffmpeg` 和 `mkvinfo` 等工具查看这些标记。[录制和播放](#)页上提供了完整的标记列表。

名称	含义	Matroska 轨道 UID
K4A_COLOR_TRACK	颜色轨道	Matroska 轨道 UID
K4A_DEPTH_TRACK	深度轨道	Matroska 轨道 UID
K4A_IR_TRACK	IR 轨道	Matroska 轨道 UID
K4A_IMU_TRACK	IMU 轨道	Matroska 轨道 UID
K4A_CALIBRATION_FILE	校准附件	附件文件名

## 后续步骤

[录制和播放](#)