
基于KINECT-DK的多 聚焦图像融合

2021.7.16

H3 本机环境

Visual Studio2019 (需安装c++拓展)

kinectSDK1.4.1

新建空白**C++**控制台工程并
添加源文件**MAIN.CPP**

H3 使用kinect-dk

1、下载SDK

此处为Github仓库提供下载，选择最新版本1.4.1最新版即可。

Using Azure Kinect SDK

Installation

If you aren't making changes to the SDK itself, you should use a binary distribution. On Windows, binaries are currently available as an MSI and a NuGet package. On Linux, binaries are currently available as debian packages. See below for installation instructions for each distribution mechanism. If you would like a package that is not available please file an [issue](#).

If you are making changes to the SDK, you can build your own copy of the SDK from source. Follow the instruction in [building](#) for how to build from source.







MSIs

The latest stable binaries are available for download as MSIs.

Tag	MSI	Firmware
v1.4.1	Azure Kinect SDK 1.4.1.exe	1.6.110079014
v1.4.0	Azure Kinect SDK 1.4.0.exe	1.6.108079014
v1.3.0	Azure Kinect SDK 1.3.0.exe	1.6.102075014
v1.2.0	Azure Kinect SDK 1.2.0.msi	1.6.102075014
v1.1.1	Azure Kinect SDK 1.1.1.msi	1.6.987014
v1.1.0	Azure Kinect SDK 1.1.0.msi	1.6.987014
v1.0.2	Azure Kinect SDK 1.0.2.msi	1.6.987014

这里我建议安装到C盘，会直接出现在根目录下

Azure Kinect SDK v1.4.1

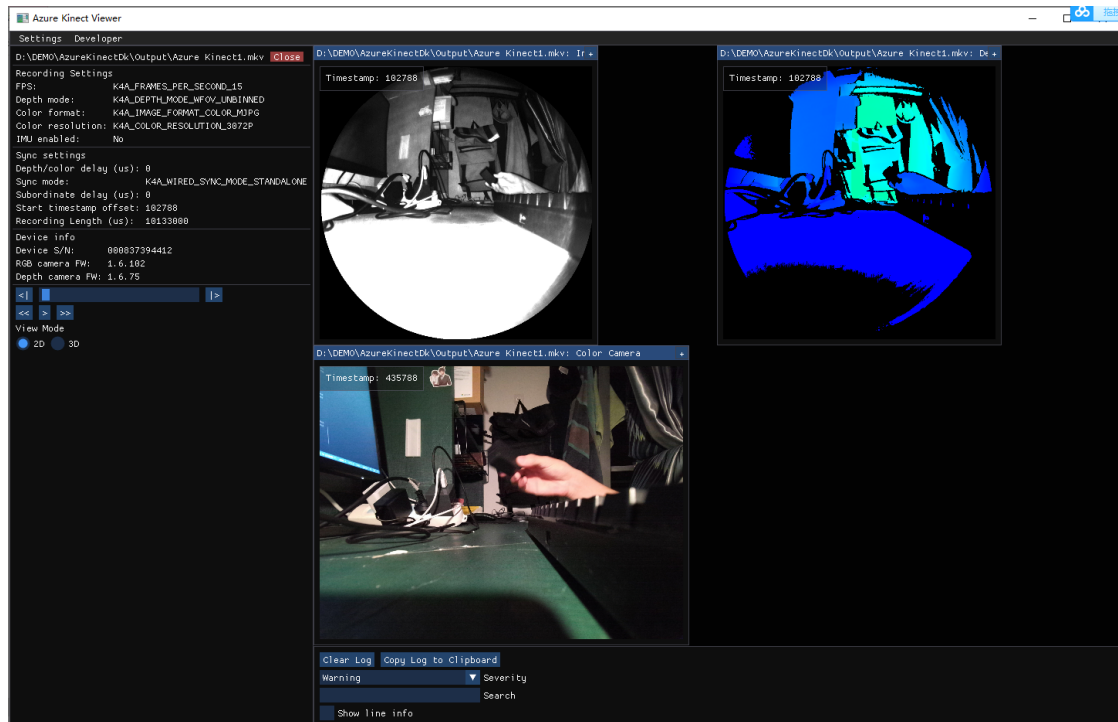
 sdk	2021/7/5 17:48	文件夹	
 tools	2021/7/7 23:48	文件夹	
 LICENSE.txt	2020/6/15 22:55	文本文档	13 KB
 REDIST.txt	2020/6/15 23:01	文本文档	1 KB
 ThirdPartyNotices.txt	2020/6/15 22:55	文本文档	116 KB
 version.txt	2020/6/15 23:01	文本文档	1 KB

SDK目录下是一些头文件

tools目录下则是工具，总共有三种

-
- Azure Kinect 查看器
- Azure Kinect 录制器
- Azure Kinect 固件工具

Kinect查看器可以启动相机，或者打开已经录制好的视频。并且展示KinectDk的三种模式，RGB模式，红外模式，深度模式。



录制器

主要使用命令行来执行，可以录制一定长度的视频。

命令行切换到tool目录下

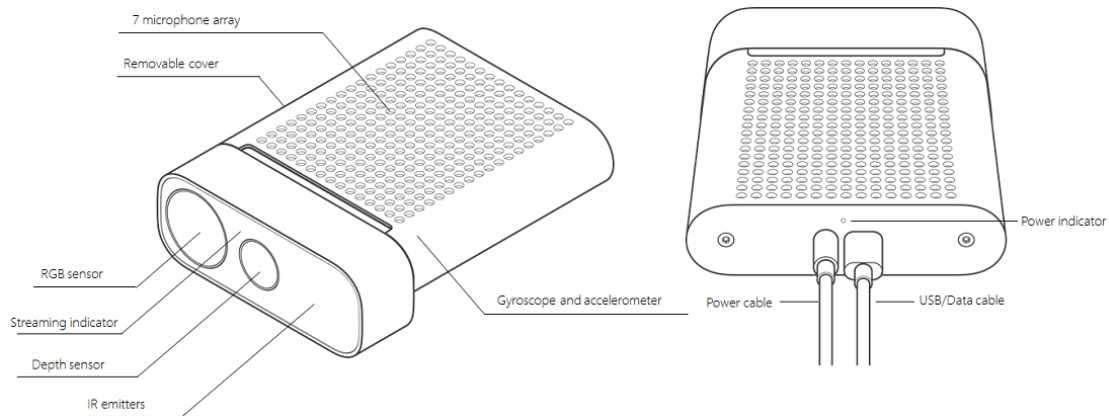
```
k4arecorder.exe -l 5 %TEMP%\output.mkv
```

即可录制五秒视频

完整参数介绍详见此页

<https://docs.microsoft.com/zh-cn/azure/kinect-dk/azure-kinect-recorder>

2、连接电源



在图片中，type-c接口的线是数据接口，要连接到电脑上，要求USB3.0（如果连接不良可以检查一下是否是使用了拓展坞，尝试使用电脑自带的USB3.0接口）

圆形接口的线是电源线，需要连接到电源。

H3 项目部署

1、新建空白C++控制台工程并添加源文件Main.cpp

配置新项目

空项目 C++ Windows 控制台

项目名称(N)

Kinect_first_app

位置(L)

D:\LHB\code\Kinect

解决方案(S)

创建新解决方案

解决方案名称(M) i

Kinect_first_app

☒ 将解决方案和项目放在同一目录中(D)

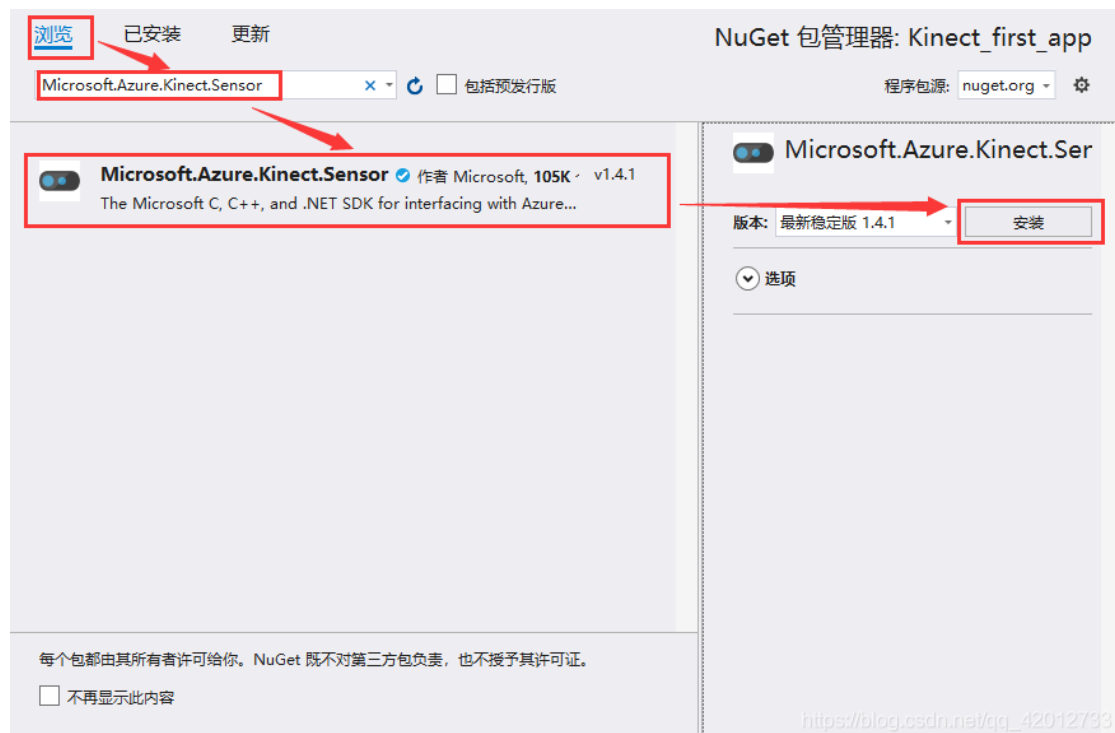
https://blog.csdn.net/qq_42012733

2、安装 Azure Kinect NuGet 包

右键 引用 - 管理NuGet程序包



3、搜索Microsoft.Azure.Kinect.Sensor，从列表中选择该包并安装。



4、搜索cv

5、添加头文件和库文件

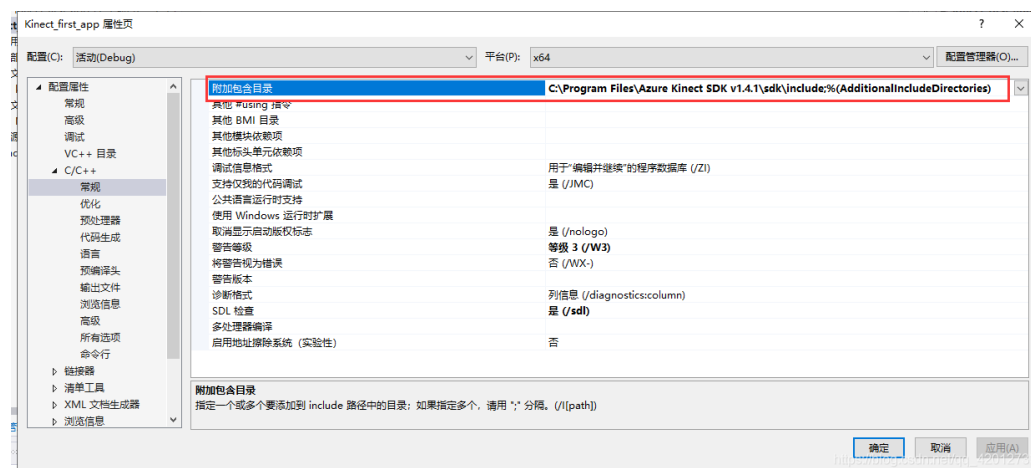
5.1 加入头文件k4a.h

直接右键 头文件 - 添加 - 现有项，找到k4a.h导入。默认路径为C:\Program Files\Azure Kinect SDK v1.4.1\sdk\include\k4a

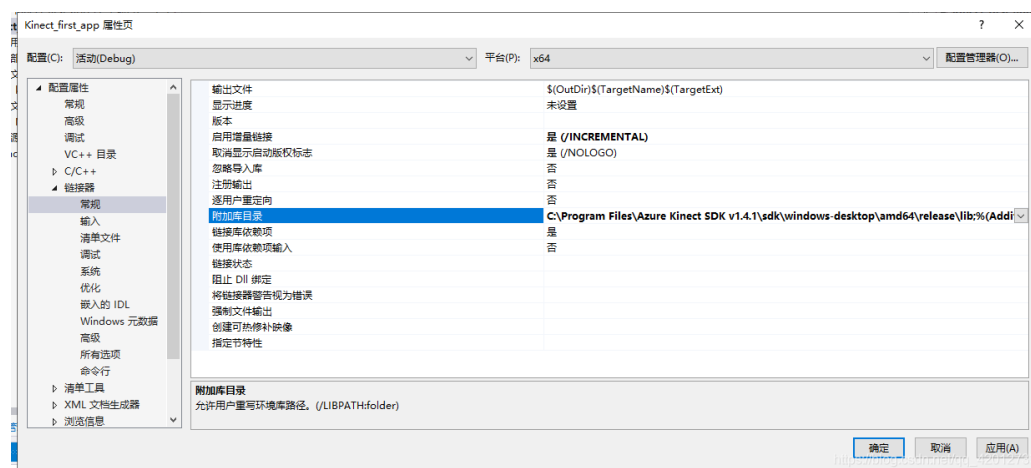
5.2 配置头文件目录和库文件目录

右键自己的项目，选择属性，上方的平台选择 x64

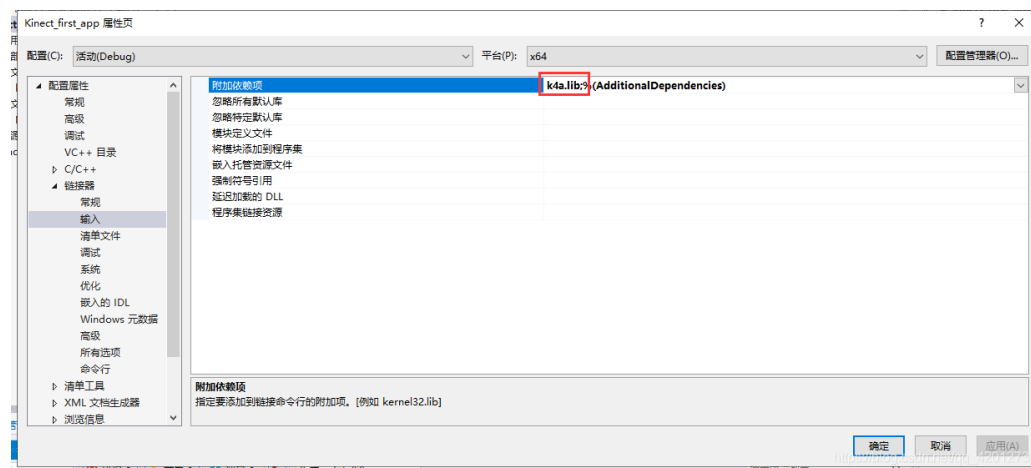
- C/C++ - 常规 - 附加包含目录，加入SDK的include路径



- 链接器 - 常规 - 附加库目录，加入SDK的lib路径



- 链接器 - 输入 - 附加依赖项，加入k4a.lib

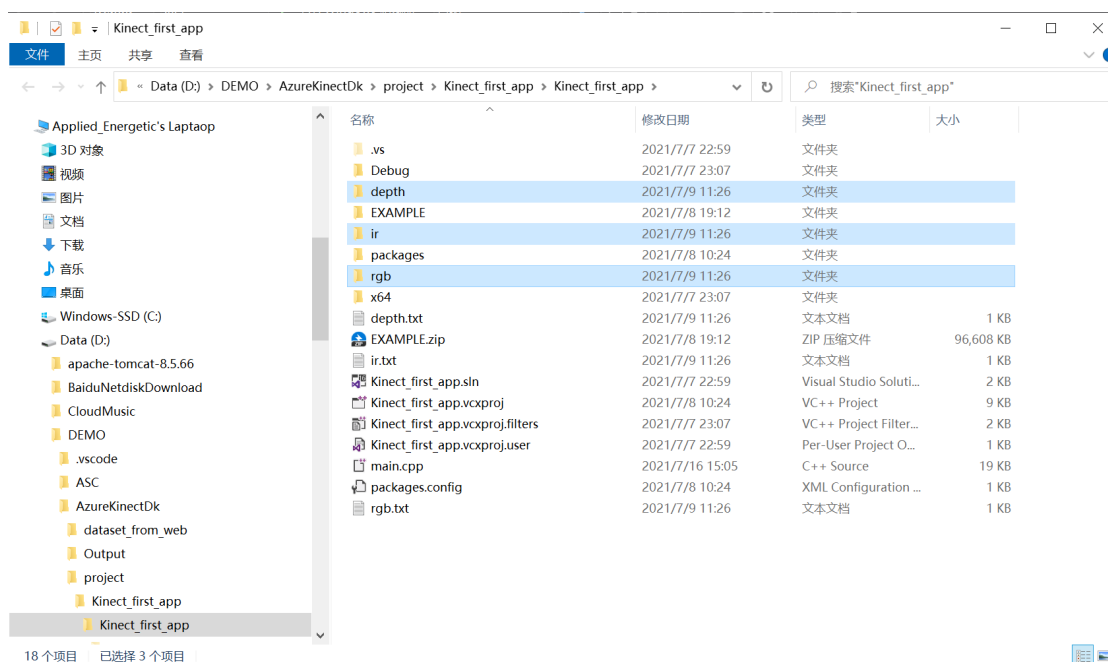


注意：每添加一项后，点击右下角应用。

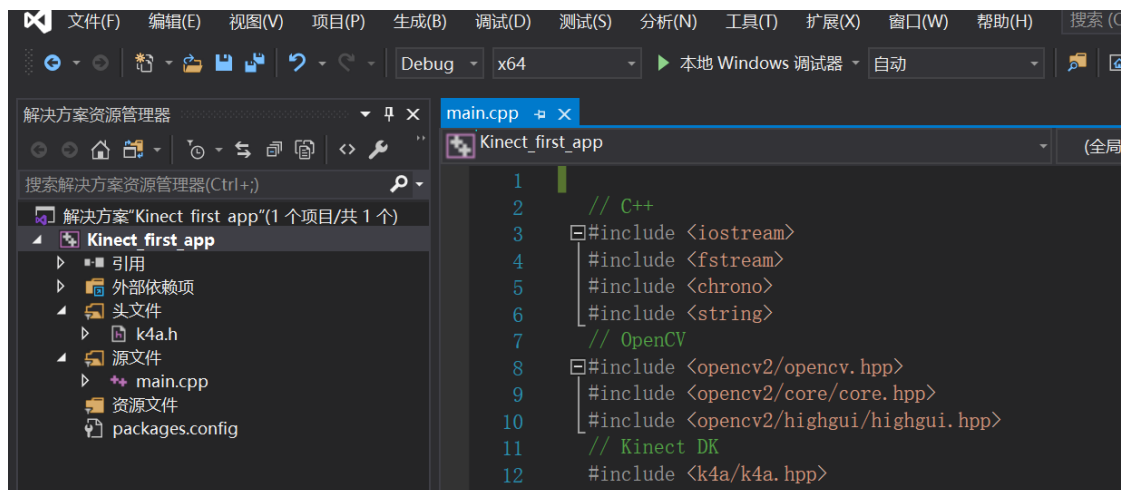
H3 项目运行

项目地址 <https://github.com/Applied-Energetic/KinectDK>

在VS项目目录下新建三个文件夹——rgb、depth、ir，输出的图片会存储到这里。



运行时点击本地Windows调试器即可。



H3 相机内参获取

KinectDk内置深度相机，其焦距等参数为一个固定值，因此无法多聚焦，下面提供一种获取焦距的方法。

获取内参矩阵

```
#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() {

    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;
    }

```

```

    k4a_calibration_t sensor_calibration;

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

    k4a_transformation_create(&sensor_calibration);
    cout <<
k4a_transformation_create(&sensor_calibration)<<endl;

    // ... 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;
}

#include <k4a/k4a.h>

#include <stdio.h>

#include <vector>
using namespace std;

#include "opencv2/core.hpp"
#include "opencv2/calib3d.hpp"
using namespace cv;

static void clean_up(k4a_device_t device)
{
    if (device != NULL)
    {
        k4a_device_close(device);
    }
}

```

```

    }
}

int main(int argc, char** /*argv*/)
{
    uint32_t device_count = 0;
    k4a_device_t device = NULL;
    k4a_device_configuration_t config =
K4A_DEVICE_CONFIG_INIT_DISABLE_ALL;

    if (argc ≠ 1)
    {
        printf("Usage: opencv_example.exe\n");
        return 2;
    }

    device_count = k4a_device_get_installed_count();

    if (device_count == 0)
    {
        printf("No K4A devices found\n");
        return 1;
    }

    if (K4A_RESULT_SUCCEEDED ≠
k4a_device_open(K4A_DEVICE_DEFAULT, &device))
    {
        printf("Failed to open device\n");
        clean_up(device);
        return 1;
    }

    config.depth_mode = K4A_DEPTH_MODE_WFOV_2X2BINNED;
    config.color_resolution = K4A_COLOR_RESOLUTION_1080P;
    config.camera_fps = K4A_FRAMES_PER_SECOND_30;

    k4a_calibration_t calibration;
    if (K4A_RESULT_SUCCEEDED ≠
        k4a_device_get_calibration(device,
config.depth_mode, config.color_resolution,
&calibration))
    {
        printf("Failed to get calibration\n");
    }
}

```

```

        clean_up(device);
        return 1;
    }

    vector<k4a_float3_t> points_3d = { { { 0.f, 0.f,
1000.f } },          // color camera center
                                     { { -1000.f,
-1000.f, 1000.f } }, // color camera top left
                                     { { 1000.f,
-1000.f, 1000.f } }, // color camera top right
                                     { { 1000.f,
1000.f, 1000.f } }, // color camera bottom right
                                     { { -1000.f,
1000.f, 1000.f } } }; // color camera bottom left

    // k4a project function
    vector<k4a_float2_t> k4a_points_2d(points_3d.size());
    for (size_t i = 0; i < points_3d.size(); i++)
    {
        int valid = 0;
        k4a_calibration_3d_to_2d(&calibration,
                                &points_3d[i],
                                K4A_CALIBRATION_TYPE_COLOR,
                                K4A_CALIBRATION_TYPE_DEPTH,
                                &k4a_points_2d[i],
                                &valid);
    }

    // converting the calibration data to OpenCV format
    // extrinsic transformation from color to depth
    camera
    Mat se3 =
        Mat(3, 3, CV_32FC1,
calibration.extrinsics[K4A_CALIBRATION_TYPE_COLOR]
[K4A_CALIBRATION_TYPE_DEPTH].rotation);
    Mat r_vec = Mat(3, 1, CV_32FC1);
    Rodrigues(se3, r_vec);
    Mat t_vec =
        Mat(3, 1, CV_32F,
calibration.extrinsics[K4A_CALIBRATION_TYPE_COLOR]
[K4A_CALIBRATION_TYPE_DEPTH].translation);

    // intrinsic parameters of the depth camera

```

```

k4a_calibration_intrinsic_parameters_t* intrinsics =
&calibration.depth_camera_calibration.intrinsics.parameters;

vector<float> _camera_matrix = {
    intrinsics->param.fx, 0.f, intrinsics->param.cx,
    0.f, intrinsics->param.fy, intrinsics->param.cy, 0.f,
    0.f, 1.f
};
Mat camera_matrix = Mat(3, 3, CV_32F,
&_camera_matrix[0]);
vector<float> _dist_coeffs = { intrinsics->param.k1,
intrinsics->param.k2, intrinsics->param.p1,
                                intrinsics->param.p2,
intrinsics->param.k3, intrinsics->param.k4,
                                intrinsics->param.k5,
intrinsics->param.k6 };
Mat dist_coeffs = Mat(8, 1, CV_32F,
&_dist_coeffs[0]);

// OpenCV project function
vector<Point2f> cv_points_2d(points_3d.size());
projectPoints(*reinterpret_cast<vector<Point3f*>*>
(&points_3d),
    r_vec,
    t_vec,
    camera_matrix,
    dist_coeffs,
    cv_points_2d);

for (size_t i = 0; i < points_3d.size(); i++)
{
    printf("3d point:\t\t\t(%.5f, %.5f, %.5f)\n",
points_3d[i].v[0], points_3d[i].v[1], points_3d[i].v[2]);
    printf("OpenCV projectPoints:\t\t(%.5f, %.5f)\n",
cv_points_2d[i].x, cv_points_2d[i].y);
    printf("k4a_calibration_3d_to_2d:\t(%.5f,
%.5f)\n\n", k4a_points_2d[i].v[0],
k4a_points_2d[i].v[1]);
}

cout << camera_matrix << endl;
//cout << intrinsics->param.fx << endl;
clean_up(device);

```

```
    return 0;  
}
```

H3 结论

1、目前还没找到修改相机的方法，官方人员的回复是不能修改内参。

issue链接如下

<https://github.com/microsoft/Azure-Kinect-Sensor-SDK/issues/1640>

The calibration matrices (intrin, extrin) are unchangable for a given mode (wide pov, narrow pov, etc.). There is no variable lens focus.

[Dale Phurrough diablodale](#)

Berlin, Germany

2、如果想要获取多聚焦RGBD图像的话应该考虑更换一种设备。

3、不更换设备的前提下想要利用RGBD数据在图像融合领域进行训练可能需要一种新的图像处理算法。

H3 引用

<https://github.com/microsoft/Azure-Kinect-Sensor-SDK>

此项目为KinectDk开发者仓库，有问题可以在这里询问。