Evaluating Intel's RealSense SDK 2.0 for 3D Computer Vision Using the RealSense D415/D435 Depth Cameras

By the staff of

BDTi

May 2018

OVERVIEW

Today's 2D computer vision applications have been limited by the lack of an important third dimension: depth. Unlike 2D, 3D vision enables machines to accurately understand shapes, sizes, distances and to maneuver in the real 3D world. But, historically, depth-sensing cameras have been expensive and difficult to use. Intel's RealSense D400 Series depth camera technology represents an important milestone in that it introduces a suite of inexpensive, easy-to-use 3D cameras for both indoor and outdoor use. Furthermore, the technology is offered in a range of scalable hardware configurations to meet market demands from one unit (for developers) to millions of units (for volume production).

BDTI, an independent technology analysis firm, performed a technical evaluation of the Intel RealSense Software Development Kit (SDK) 2.0 for developing 3D vision applications using the RealSense D415/D435 depth cameras. In our evaluation, we sought to understand the ease of use and developer efficiency for developing 3D vision applications using this SDK. To get hands-on with the SDK, we developed a simple, real-world computer vision application that uses depth information to produce real-time information about the physical size of objects in a video stream generated by the depth camera. The application leveraged commonly used computer vision packages, such as OpenCV, and deep learning algorithms for object localization and detection.

Overall we had a very positive development experience and found the RealSense SDK 2.0 to be a complete and easy-to-use development platform. The SDK supported all of the needed building blocks to develop our application, including good support for OpenCV. Interfacing with the depth cameras was straightforward using the SDK's API. Contributing to the ease of use, the entire SDK is available in open source form, enabling developers to review and experiment with the code.

Contents

1. Introduction

This report presents an independent evaluation of the Intel RealSense SDK 2.0 using the RealSense D415 and D435 depth-sensing cameras. The focus of the evaluation was on the ease of use and developer efficiency for developing 3D-vision applications.

As a key component of the evaluation, BDTI implemented a simple 3D application that uses depth information to measure the height of objects (in this case people) in a live video stream generated by the depth camera. In addition to the RealSense SDK 2.0, we leveraged commonly used computer vision packages, such as OpenCV, and deep learning algorithms for object localization and detection.

The target audience for this report is application developers and managers interested in an independent perspective on the development experience using the RealSense SDK 2.0 and the RealSense D415/D435 depth cameras. [1]

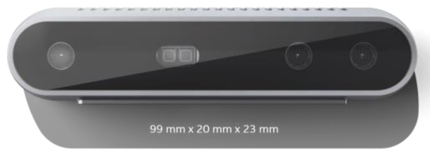
2. About BDTI

This independent evaluation was performed by BDTI, a technology analysis and software development firm specializing in embedded computer vision and deep learning applications. BDTI has extensive experience developing, optimizing and deploying computer vision applications across many different platforms. In addition to its software development work, for more than 25 years BDTI has performed in-depth, hands-on evaluations of numerous processors, development kits and tools. For more information about BDTI, please visit https://www.bdti.com/contact. For questions about this report, please email us at info@bdti.com.

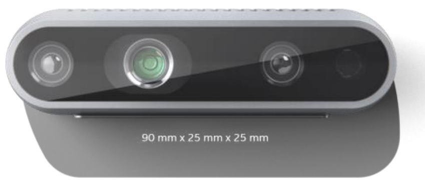
3. The Intel RealSense D415/D435 Depth Cameras

Today's 2D vision-based applications have been limited by the lack of an important third dimension: depth. Unlike 2D, 3D vision enables machines to accurately understand shapes, sizes, distances and to maneuver in the real 3D world. Historically, 3D cameras have been expensive and difficult to use. Intel's RealSense D400 Series depth camera technology represents an important milestone in that it introduces a suite of inexpensive, easy-to-use 3D cameras for both indoor and outdoor use. Furthermore, the technology is offered in a range of scalable hardware configurations to meet market demands from one unit (for developers) to millions of units (for volume production).

For developers to get started quickly, the RealSense D415/D435 depth cameras are ready-to-use cameras that plug into a USB port.



RealSense D415 Depth Camera



RealSense D435 Depth Camera

Figure 1: The RealSense D415 and D435 Depth Cameras

For higher levels of hardware integration, Intel also offers the RealSense Depth Module D400 Series. These modules feature the same camera technology as the D400 cameras but are offered in a sub-assembly module, enabling developers to integrate depth sensing into their hardware product. For higher volume applications, developers can choose to integrate the RealSense Vision Processor D4 Series chip directly into their board-level hardware design. The D4 Series chip is also found in the D400 Series Cameras and Modules and computes a 3D depth map without the use of a GPU or host processor.

Feature	D415 Depth Camera	D435 Depth Camera
Image sensor technology	Rolling shutter	Global shutter
Depth Field of View (H x V) for 16:9	Narrow: 63.4° x 40.4°	Wide: 85.2° x 58°
Camera dimensions	99 mm x 20 mm x 23 mm	90 mm x 25 mm x 25 mm
Intended use case	Precise measurement. Narrower FOV results in higher depth resolution.	Low light and wide field of view. Wider FOV enables coverage of more area, resulting in fewer "blind spots."
Use environment	Indoor/Outdoor	
Depth technology	Active infrared stereo	
Depth resolution	Up to 1280 x 720 at 30 frames per second (fps)	
Maximum range	Approximately 10 meters	

Figure 2: Specifications for the RealSense Depth Cameras [2]

4. What's in the Intel RealSense SDK 2.0?

The purpose of the RealSense SDK 2.0 is to ease the development of computer vision applications using depth information provided by the RealSense depth cameras. We expect the majority of RealSense SDK 2.0 users will have previous experience developing computer vision applications, but little to no experience with the use of depth information. These developers will want to leverage familiar tools, libraries, and frameworks (e.g., OpenCV).

There are four major components of the SDK: the librealsense2 library, tools, sample code and wrappers:

- **librealsense2**. This library is the core component of the SDK and provides an API to configure, control and access the streaming data from the depth cameras. The API allows getting started with the camera basic functionality using the high-level API, or get full control of all camera settings using the low-level API. [3]
- **Tools.** Provided in the SDK are two application tools and a set of five debug tools. The application tools include the RealSense Viewer (enabling easy visualization of the video and depth streams and setting the camera's configuration), and the Depth Quality Tool (enabling developers to test the camera's depth quality).
- **Sample code.** There are approximately twenty code examples demonstrating how to use the SDK and the depth cameras for various tasks, including aligning depth frames to their corresponding color frames, displaying the distance from the camera to the object in the center of the image and how to use the depth cameras with existing deep neural network (DNN) algorithms.
- **Wrappers.** Support for a broad range of languages and software platforms is provided by the included wrappers, including python,

.NET, Node.js, Robot Operating System (ROS), LabView, Point-Cloud Library (PCL) and the Unity gaming platform.

Intel has made the entire SDK and all of its major components available in source code form under the Apache 2.0 license and hosted the SDK on GitHub.

5. BDTI's Evaluation Methodology

The focus of BDTI's evaluation was to explore the overall development experience, including ease of use and developer efficiency. The evaluation explored topics such as:

- How hard is it to get things working out of the box?
- How hard is it to develop a new 3D vision application using the SDK?
- What are the SDK's strengths and weaknesses viewed through the eyes of someone trying to develop a depth-sensing computer vision application?

We did not focus on the technical characteristics of the cameras. For example, we did not measure things like the RMS depth sensing error of the cameras, their sensitivity under different lighting conditions or their color correctness.

For a realistic assessment of the overall development experience, we followed a typical developer's journey to develop a new application with a new SDK. The journey included installation, exploration, application design, development and testing. To put the SDK through its paces in a real-world use case, BDTI developed a simple 3D application that uses depth information to measure the height of people in a live video stream generated by the depth camera. The people detector we implemented is deep-learning based and could be retrained to detect other objects; one can imagine this application being useful in several types of use cases, such as in a warehouse setting where packages over a certain height are flagged for a different type of transport, or in an amusement park setting where children shorter than 36" are not permitted on the ride.

Our evaluation was performed by a BDTI engineer experienced in computer vision application development, but with no prior experience with depth cameras.

6. Step 1: Installation

For our hardware platform, we chose to use one of our existing PCs with 32 GB RAM and a USB 3.0 port. While we did not evaluate other hardware platforms, it is notable that Intel offers unsupported versions of the SDK on smaller platforms such as the Raspberry PI 3 Model B and Android. To enable extensibility, full source code for the SDK is available for developers needing to support other platforms.

Intel states that the RealSense SDK 2.0 is fully supported on Microsoft Windows and Linux platforms; in addition, there is limited support for macOS. We decided to test the implementation under Linux. Unfortunately, we immediately ran into problems. We discovered that our Linux kernel was newer than what was then supported by the SDK. While there is SDK documentation to inform the developer of the supported kernel versions, we believe that Linux kernel compatibility issues will likely be a recurring problem for developers. According to Intel, the RealSense SDK 2.0 requires that Intel make modifications to the Linux kernel drivers. Since it takes time for the Intel team to add the modifications whenever the newest Linux kernel is released, the version of kernel supported in the SDK will tend to lag the version of the kernel in the latest Linux distribution. Intel reports that a more universal solution for this type of Linux installation problem is planned.

After installing an older Linux kernel, we completed configuring our system by plugging the D415 depth camera into our USB 3.0 port. We then launched the Viewer application (included with the SDK), which enabled us to visualize video streams with depth information. We repeated the process using the D435 depth camera. Both worked fine. Based on our experience, developers should have no trouble installing the system and visualizing real-time video and depth frames in less than a day.

7. Step 2: Exploration

With a working 3D camera and basic software functionality demonstrated, we began exploring the SDK's tools and sample applications.

Included in the SDK are two application tools (RealSense Viewer and Depth Quality Tool) and five debug tools. We started our evaluation with the Viewer tool, which allows users to stream and visualize RGBD streams [i.e., RGB images plus depth ("D") information for each pixel] from the camera easily and quickly. We found this tool to be easy to use, and using it, we were able to quickly get depth and color images displayed on the screen.

Intel has organized the provided code samples into three categories: basic, intermediate and advanced. We ran samples from each of these three categories.

- rs-capture (Basic): This sample code demonstrates how to configure the camera for streaming and rendering depth and RGB data to the screen. We ran this sample with no problems. For most applications, developers should explicitly align the depth and color streams prior to running this sample code (see rs-align).

- rs-align (Intermediate): This sample code demonstrates the alignment of the depth frames to their corresponding color frames. We ran this sample with no problems. However, this tool could be made more intuitive for new developers by showing the color and depth frames before and after alignment and also to show the alignment of the color frame to the depth frame and vice versa.

- rs-measure (Advanced): This sample code lets developers measure distance between two points in the physical world. Included in this example are a half dozen critical measurement concepts and details of performance techniques such as the use of multiple threads of execution. This sample code ran well with no problems. Entry-level developers would benefit from a simplified version of this sample demonstrating basic 3D measurement concepts.

We were pleasantly surprised to see the "all-in" use of open source for the RealSense SDK 2.0. This is especially attractive for computer vision application developers who are already avid users of open source tools and libraries such as OpenCV. Making all of the RealSense SDK 2.0 code available in source code, using the friendly and well known Apache 2.0 license, removes licensing hurdles and makes it easy for developers to review and experiment with the code. The RealSense GitHub developer portal is well organized with basic documentation, including theoretical (e.g., technical white papers), practical (e.g., troubleshooting guide and API How To) and reference material (e.g., API architecture, frame management and camera-specific topics).

However, with no prior experience with depth cameras, developing a 3D vision application is non-trivial. We experienced a longer-than-desired learning curve on the use of depth cameras. It was a challenge to learn the many configuration (e.g., color and depth frame alignment) and measurement concepts fundamental to depth cameras and the implications for constructing a 3D vision application. Developers would benefit from the addition of a well-organized series of basic tutorials that are designed to rapidly onboard developers to the RealSense platform. These tutorials should include guided learning paths for entry to advanced developers covering topics ranging from prerequisite educational materials for depth cameras to more advanced topics specific to each of the supported environments and platforms (e.g., OpenCV, ROS, LabVIEW and Unity).

We would also have appreciated more advanced tutorials that offered guided learning paths specific to vertical application solutions (e.g., robotics, drones, AR/VR, etc.). These advanced tutorials would accelerate the time-to-first application for developers working in these domains. Intel stated it's planning on expanding their tutorials to meet the needs of both experienced and novice developers.

8. Step 3: Application Design and Development

As mentioned, the application we decided to develop identifies people in an image and measures their height in real time, using depth information. Now, you might wonder, why do we need a depth camera to do this? Can't you just see how tall a person is (in pixels) and go from there? The reason is that, without depth information, you have no way to know how far away a person is from the camera. As a result, a short person standing closer to the camera might appear to be taller than a tall person standing further away. By combining both RGB data (i.e., images from the camera) and depth information, we can accurately determine a person's height.

The core functionality of our simple height detection application includes interfacing with the depth cameras to receive live video and depth streams, identifying people in the image, detecting each person's distance from the camera and then calculating the person's height.

With this application functionality in mind, we started thinking about how we might implement it. The popular OpenCV library seemed like a good starting place as it enables developers to quickly implement and test computer vision algorithms. Fortunately, it turned out that interoperability between the SDK and OpenCV is well designed. For example, converting a depth frame from the camera into OpenCV matrices is well documented in the OpenCV sample code included with the SDK. With the decision that our application would leverage OpenCV, we were able to complete our initial design and construct our high level architecture diagram as shown in Figure 3.

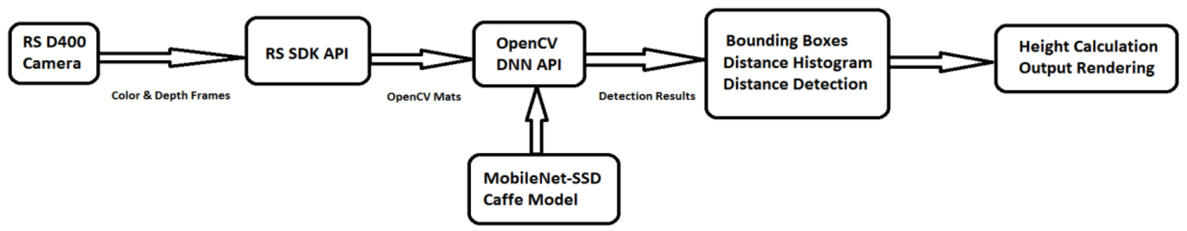

RS D400 Camera
Color & Depth Frames
RS SDK API
OpenCV Mats
OpenCV DNN API
MobileNet-SSD Caffe Model
Detection Results
Bounding Boxes Distance Histogram Distance Detection
Height Calculation Output Rendering

Figure 3: Diagram of the Application Architecture

Now that we had an overall application design, we were ready to start writing the code. The key functions required by the application include interfacing with the depth camera to capture the color and depth frames, detecting people, determining distance, calculating heights and rendering output.

Our first task was to interface with the depth camera. The RealSense depth cameras can capture color and depth frames continuously. The RealSense SDK 2.0, with the librealsense2 library, provides the API functions needed to align the color and depth frames. The alignment is important because the object detection works on color frames and we wanted to measure the distance of the same object on depth frames. Using the API, our application was able to configure the depth camera, capture the color and depth frames and then align them. We then converted the frames to OpenCV matrices.

The next function we needed to code was the object detector. Selecting the appropriate object detector that meets the accuracy, performance and computational constraints of an application can be a very involved and time-consuming process. Fortunately, OpenCV includes a deep learning module (DNN) with support for a number of deep learning frameworks, including Caffe, TensorFlow and others. This DNN API works with pre-trained deep learning models. Sample code included in the SDK demonstrates the interoperability among the depth camera, OpenCV, and OpenCV's DNN API to localize and detect an object in the image, as well as how to detect the object's mean distance from the camera. Our application called the OpenCV DNN API to detect people using a pre-trained Caffe model (MobileNet + SSD). [4]

Now that we had developed our people detector, it was time to code our distance detection. Each pixel in the depth frame includes the distance (in meters) from the camera. Since the color and depth frames were aligned, we had the distance for all the pixels within the bounding box. The challenge was to find which pixels belonged to the person and which pixels were associated with other objects in the image. We found the SDK examples for distance measurement were simplified and results could be obscured by the surrounding objects. The examples used either the distance of the center pixel or the mean distance of a region of interest (ROI) that was deemed the distance of the object. Unfortunately, these techniques left us with an inconsistent distance measurement. To improve the consistency of the distance measurement, we implemented a more robust way to measure the distance of each detected person. Each bounding box was an ROI, and we computed a histogram for that ROI on the depth frame. The distance of the histogram peak was deemed the distance of the detected person. We found this approach to be more resistant to occlusions and background pixels.

The final piece of functionality we needed to code was our height calculation and output rendering stage. This stage included drawing a bounding box around each detected person and calculating the height of the bounding box (in pixels). Applying our formula, based on the distance (in meters) of the person in the bounding box, the height of the bounding box (in pixels), and the camera's focal length (in pixels), we could then compute the height of the person (in meters). This assumes that the height of the person is represented by the height of the bounding box. The focal length (in pixels) is one of the camera's intrinsics, i.e., a parameter determined by the camera optics and components. The RealSense SDK 2.0 provides an API to access such intrinsics. See Figure 4 and the formula below.

$PH = (D * bh)/f$; where:

- PH = Person Height (meters)
- D = Distance between the camera and the object (meters)
- bh = Bounding box height (pixels)
- f = Focal length in pixels (generated by the SDK as shown in the referenced "API How To") [5]

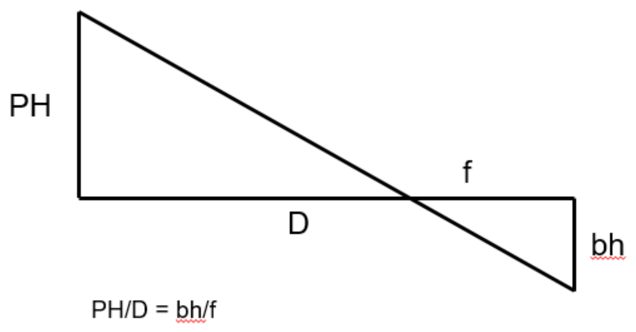


Figure 4: Person height calculation

9. Step 4: Application Testing

As is typical in developing any application, we developed code and tested concurrently. Our testing objective was to validate our code for these four components: receiving and displaying RGBD streams, object localization and detection (people), distance measurement and calculating the height of the bounding box. Our tests were performed indoors with the D415 and D435 depth cameras. We had people standing in front of the camera at certain distances. Our application detected each person and displayed the measured distance on the screen and confirmed with the ground truth distance. To get an accurate height measurement, people had to stand away from the camera so that their whole bodies were seen by the camera. With the D415 depth camera, our application displayed the height of each person, which was then verified with the ground truth. When we repeated the above test using the D435 camera, our application displayed an incorrect height of each person. Despite receiving technical support from Intel, we were ultimately unable to determine the root cause of this problem in the time we had available. For a final implementation using the D435, we would need either to troubleshoot the problem further or "back calculate" the focal distance *f* used in our height calculation formula.

10. Conclusions

Overall, we had a very positive development experience and found the RealSense SDK 2.0 to be an easy-to-use development platform. The SDK supported all the needed building blocks to develop our application, including good support for OpenCV. Interfacing with the depth cameras was straightforward. Based on our experience, developers should have no trouble installing the system and visualizing real-time video and depth frames in less than a day.

Contributing to the ease of use, the entire SDK is available in open source form, enabling developers to review and experiment with the code. Moreover, developers can build the SDK from source code to target non-supported hardware platforms.

That said, there is more work to do to improve the SDK's out-of-box experience, as we encountered problems with installing the SDK on our standard Linux platform. More significantly, we experienced a longer-than-desired learning curve because we had no prior experience with depth cameras. Our developer, who was new to depth cameras, would have greatly benefited from a well-organized series of basic tutorials designed to rapidly onboard developers to the RealSense platform, including prerequisite educational materials for depth cameras. Given that 3D sensing is still relatively new, we expect that this will be the case for many developers.

The other dimension of developer experience that we explored is developer efficiency. For us, efficiency means how quickly a developer can develop a high-quality 3D vision application that meets his business requirements using the contents of the SDK. For our simple 3D application, the SDK aided our developer efficiency by offering a high-level API for interfacing with the depth camera, making our work to interface to the depth camera simple. Another efficiency boost came from the SDK's interoperability with OpenCV, an extensive library of functions that enables developers to easily implement algorithms for vision-based applications. However, we recognize that our simple 3D vision application is not representative of the sophisticated solutions required by the emerging 3D markets. Missing from the SDK are a series of advanced tutorials that offer guided learning paths specific to vertical application solutions (e.g., robotics, drones, AR/VR, etc.). Developer efficiency, and thus time-to-deployed application, would be greatly improved for developers of these types of applications.

11. References

[1] https://realsense.intel.com

[2] https://software.intel.com/en-us/realsense/d400

[3] https://github.com/IntelRealSense/librealsense/wiki/API-How-To

[4] *A Caffe implementation of a MobileNet-SSD detection network:* https://github.com/chuanqi305/MobileNet-SSD

[5] *Get Video Stream Analytics:* https://github.com/IntelRealSense/librealsense/wiki/API-How-To