Intel® RealSense^{TM} Dimensional Weight Software (DWS)

Based on Intel® RealSense^{TM} LiDAR Camera L515

Revision 1.1

Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at intel.com.

Intel technologies may require enabled hardware, specific software, or services activation. Check with your system manufacturer or retailer.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or visit www.intel.com/design/literature.htm.

Intel and the Intel logo, Intel® Core™, Intel® Atom™, trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.

© 2020 Intel Corporation. All rights reserved.

Contents

Figures

Tables

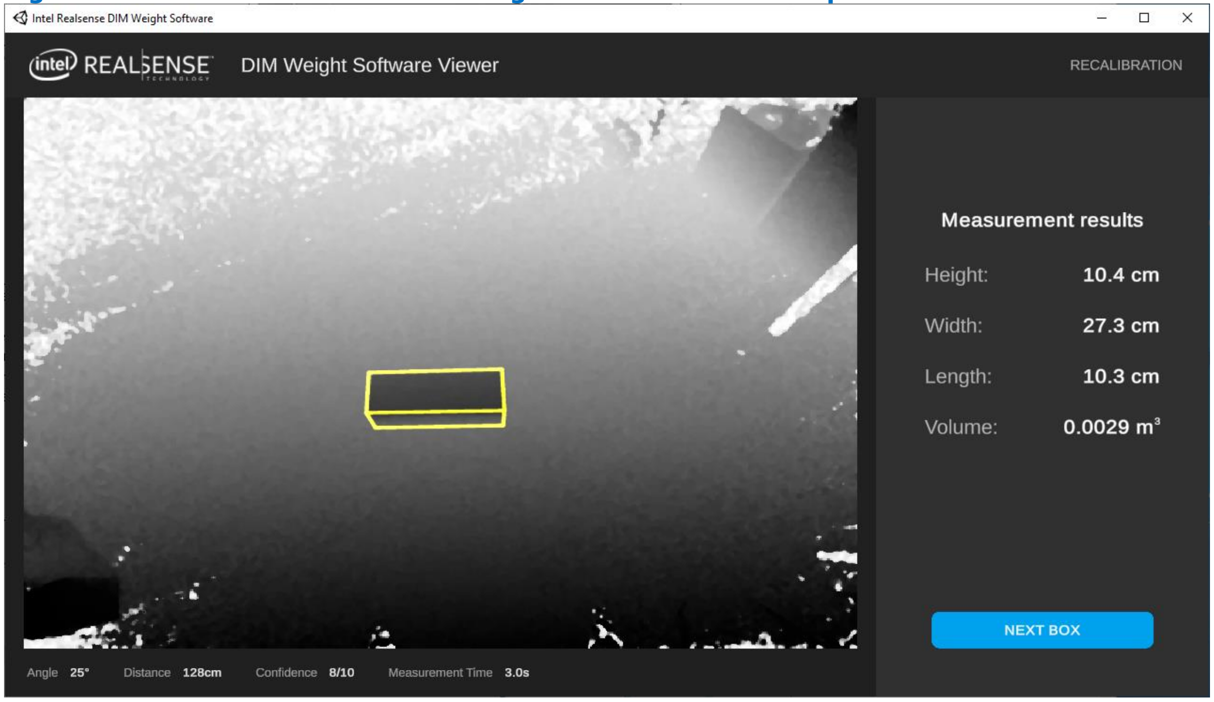
1. Overview

Intel® RealSense™ Dimensional Weight Software (DWS) is an easy to use, high-speed, precise volumetric measurement software solution.

By combining the power of our proprietary volumetric measurement SDK with the award-winning Intel® RealSense™ LiDAR Camera L515, we've built a legal for trade ready, highly accurate measurement platform. From small boxes to large pallets, our solution enables logistics companies to better manage their inventory and shipments. Intel® RealSense™ DWS enhances billing accuracy, aids in warehouse space management, and increases workflow efficiencies through fast and accurate to-the-millimeter measurements. Our platform seamlessly integrates with existing enterprise systems or can be activated right out of the box.

Intel® RealSense™ DWS – Leading the dimensional measurement revolution by empowering our partners with best-in-class Intel® RealSense™ technologies.

Figure 1. Intel® RealSense™ DIM Weight Software Demo snapshot



Intel Realsense DIM Weight Software

DIM Weight Software Viewer

RECALIBRATION

Measurement results

Height:	10.4 cm
Width:	27.3 cm
Length:	10.3 cm
Volume:	0.0029 m³

NEXT BOX

Angle **25°** Distance **128cm** Confidence **8/10** Measurement Time **3.0s**

2. APIs

2.1 Available configurations

Feature		Description
Operational mode	Static	Box and camera stationary
	Dynamic	Future feature
DWS callback function		DWS_measureCallback returns 2D and 3D data in order to enable user view of the box in the scene as part of DWS_Box
DWS frame callback		Returns Dpeth, IR, & RGB frames from the camera streams, alongside an internal frame number identifier.
DWS status callback		Is called when a noteworthy event has occurred (e.g. calibration completion, errors, etc.)
Calibration mode		Calibration enhances the dimension accuracy. Performed once during camera installation.

```
DWS_Status dws_create(DWS_Mode mode, bool multiple_boxes, dws_frameCallback
usercallback, dws_statusCallback statuscallback, DWS_Box * box_buffer, int&
outHandler);
```

Creates the SDK engine, starts the camera at the preconfigured settings, and returns stream via a callback.

Accepts:

1. Mode: Static
2. Multiple boxes: Future use
3. Frame callback: Pointer to functioned used to get streaming frames from the SDK
4. Status callback: Pointer to function used to update on SDK events and errors
5. Box buffer: A structure to hold returned boxes.
6. Out handler: A variable which will be assigned the ID of the created engine. This ID is used in later calls to identify which engine instance to perform the current API call.

```
DWS_Status dws_calibrate(int handler, int x, int y):
```

This is a future feature, and currently is a stub.

Initializes the plane which coordinates x & y belong to as the base plane, on which the boxes will appear

Accepts:

1. XY coordinates
2. Handler ID: see create for details.

`DWS_Status dws_autoCalibrate(int handler, bool use_calibration_target):`

Initializes a base plane at the engine's discretion.
Will raise statusCallback (see create) to indicate progress/completion.
Accepts:

1. Handler ID: see create for details.
2. Use_calibration_target: signals the SDK if calibration attempt will make use of a calibration pattern

`DWS_Status dws_start(int handler, dws_measureCallback usercallback):` Starts the measuring engine, returning measured objects via callback

Accepts:

1. Handler ID: see create for details
2. User callback: the callback via which results will be returned, contained in the buffer supplied at create()

`DWS_Status dws_pause(int handler):`

Pause the measurement engine while still receiving camera stream.
To resume measuring, call start().
Pausing retains the calibration data attained by autoInit.
Accepts:

1. Handler ID: see create for details.

`DWS_Status dws_stop(int handler):`

Stop both streaming and measurements.
To restart, call Create() again.
Frees all resources.
Accepts:

1. Handler ID: see create for details.

`DWS_API_CALL const char * dws_version(int handler):`

returns the SDK version
Accepts:

1. Handler ID: see create for details.

DWS Dimension Callback signature:

```
void: Foo (int frame_id, int box_count, rsBox* boxes);
```

P 1: frame identifier. This matches to an identifier received previously in dws_frameCallback()

P 2: number of boxes detected

P 3: pointer to the detected box buffer (this is the buffer supplied by the user in dws_create())

DWS_Box:

- **Dimensions**:3 float fields, **height, length, width**
- **Corners_2D**: 1x8 2D Point array
- **Corners_3D**: 1x8 3D Point array
- **Confidence**: value in range [0 – 10] (int)
- **ID**: int unique box ID (unique only within current session)
- **resolutionAtAxis**: 1x3 float array, representing point cloud density on the dimension [H,L,W]
- **planeDistance**: 1x3 float array, representing distance to the plane denoting the dimension [H,L,W]
- **planeAngles**: 1x3 float array, representing averaged angle to that plane
- **boxOrigin**: 1x3 float array, representing 3d location of the intersection of planes used in the measuring process

```
void * dws_frameCallback(const void*, int, int, int, const void*, int, int, int,
const void*, int, int, int, int);
```

P 1: pointer to depth frame buffer

P 2: depth frame width

P 3: depth frame height

P 4: pointer to IR frame buffer

P 5: IR frame width

P 6: IR frame height

P 7: Pointer to RGB frame

P 8: RGB frame width

P 9: RGB frame height

P 10: Identifier for the current frames as a group (loops around at MAX INT)

DWS Status Callback:

```
void* dws_statusCallback(DWS_Severity severity, DWS_MessageCode code, const char *)
```

P 1: The notification’s severity (Error/Warning/Info)

P 2: numeric message code identifying the status

P 3: String with additional data

3. Recommended setup

Stationary setup guidelines:

In order to achieve best performance and accuracy Top set-up position is recommended

- Distance to floor or table-top should be larger than 0.8m.
- Distance to box top surface should be larger than 0.7m.
- Distance to floor or table-top should be less than 4m.
- Angle between line of sight from camera to box top surface normal should be between 0 and 45deg.
- Entire box must be included within the central 80% of frame (rim of 10% on each of the 4 sides).
- Box should be flat on a surface (not tilted).
- The scene should not be exposed to sunlight.
- For best results surface under box should have high reflectivity and very low specularity (matte and not shiny)

The setup is shown in Figures 2 and 3.

Figure 2. Intel® RealSense™ DWS – Camera angle

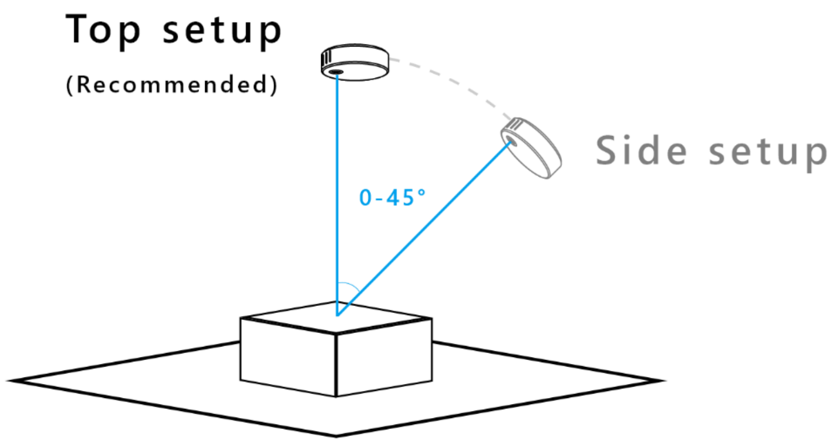


Figure 3. Intel® RealSense™ Camera distance to furthest visible point of box

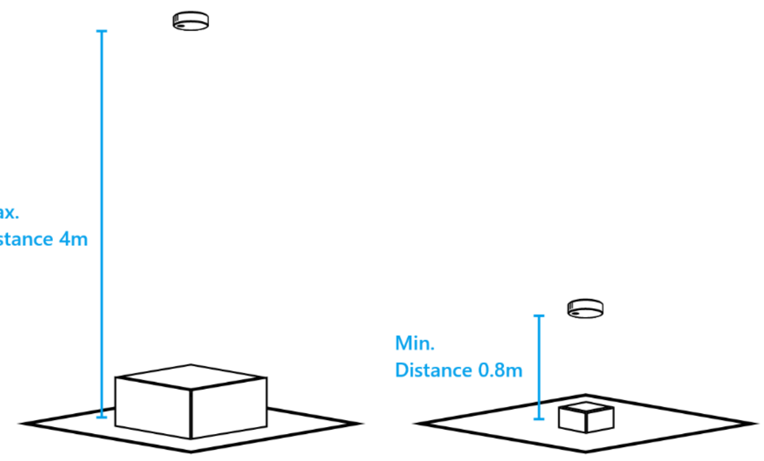
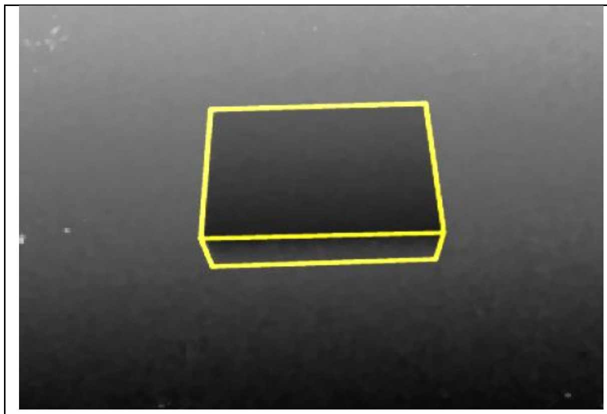
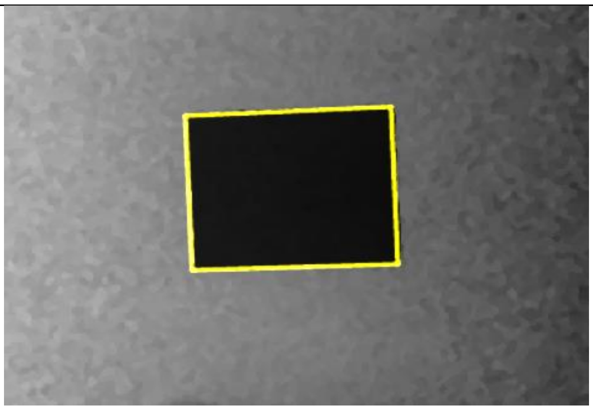


Figure 4. Example of an acceptable side view of the box (left) and an acceptable top view of the box (right)

 Example of orientation showing 2 facets with angle to normal less than 45°	 Example of orientation showing only 1 facet

4. Initialization Phase

DWS uses the plane and position of the floor or table-top in box measurement calculations. During initial setup a calibration to the floor or table-top is required.

4.1 Floor calibration without a target

Dws_autoCalibrate() first looks for a calibration target. If no target is found, the floor (table-top) is used for calibration. Requirements for floor or table-top calibration

- The surface (floor, table-top) must be flat. (For uneven surfaces see target calibration method below.)
- The surface should be reflective. A wood floor is an example of a good reflective material. Dark carpet is an example of a poorly reflective material.
- The surface should not be specular. A highly polished and shiny floor has high specular reflection and is not advised.
- The camera must remain in a fixed position after calibration. If the camera or surface is moved, a new calibration is required.

4.2 Targeted calibration

In the case that the targeted area for box placement is different than the floor sensed by the camera, a guided calibration is available. An example of this use case can be a camera that has a table and a floor in the scene where the table is used to measure the box size. Another example is a camera that has in the field of view both a scale and floor where the box is placed on the scale which is higher than the floor level.

The target can be downloaded at this URL https://www.intelrealsense.com/wp-content/uploads/2020/07/DWS_Calibration_Target.png

dws_autoCalibrate() will automatically look for a calibration target. If the target is found, it will be used for the calibration. If the target is not found, the floor (table-top) is used for calibration, and a CALIBRATION_SUCCEEDED_NO_MARKERS status will be raised, to signal that calibration was completed, but failed to detect the target.

If the target is printed on material that has some thickness, this height can be automatically removed by the application during measurement. Within the SDK folder open the file /Configuration/static_mode_config.json. Replace the value in “floor_bias” with the appropriate thickness of the target in mm. For example, if the target material is 1mm thick, the line would be

```
“floor_bias” : -0.001,
```

Target size required based on camera distance to floor or other surface.

Distance from camera to floor is less than 2 meters.	Print target on ISO A4 or ANSI A (8.5”x11”)
Distance from camera to floor is greater than 2 meters and less than 4 meters.	Print target on ISO A2 or ANSI C (16”x23”)

Requirements for floor or table-top calibration with target:

- The printed target must be flat. Folds and creases should be avoided. If the edges of the paper are curling, gluing or taping the target to flat cardboard is recommended.
- Print on non-glossy, white copy paper.
- Scale to fit the page size.
- Place the target in the center of the camera frame for proper calibration.

5. Box Measurement specifications

The basic performance metrics of Box Measurement application are listed in Table 1.

5.1 DWS Specifications

Table 1. Box Measurement spec table

Parameter	Value
Measurement granularity	1mm
Lighting	Indoor lighting with no sunlight. Other light sources that emit light in the 860nm wavelength should also be minimized.
App latency	<2 sec from calling measure command to receiving X,Y,Z measurements with high confidence. Benchmark performed on Intel Core i5 Gen6, 8GB RAM with an SSD HD, running Windows 10.
Box materials	Wood, cardboard
Surface materials	Lambertian (matte) surfaces * shiny surfaces such as reflective tape or plastic wrap could cause measurement inaccuracy

5.2 Measurement Accuracy

Measurement accuracy is defined in terms of scale division “d”, which represents the max rounded error. This d value is dependent on the box size and box distance from the camera.

Table 2. Box Measurement accuracy table

d (cm)	Min box size LxWxH	Max box size LxWxH	Max distance camera to floor
0.5	5cmx5cmx5cm	90cmx80cmx50cm	1.5m
1	10cmx10cmx10cm	100cmx120cmx100cm	2.5m
5	30cmx30cmx30cm	150cmx200Lx200cm	4m

The above table is valid for perfect cuboids. Tests are performed using wooden boxes on a wooden floor. Irregular shapes and differences in reflectivity of the boxes and floor could affect measurement results.

For real life boxes, where the box might be dented, have rounded corners, etc., accurate measurement is defined using the following formulas:

BOX_SIZEx = one of the tape measured for a specific size

BOX_SIZE_AVG = Box size average will be calculated as the average of three BOX_SIZE measurements plus the max error between measurements

BOX_SIZE_STD = max(|BOX_SIZE_AVG – BOX_SIZE1|; |BOX_SIZE_AVG – BOX_SIZE2|;|BOX_SIZE_AVG – BOX_SIZE3|)

BOX_SIZE_STD must not exceed 5% of BOX_SIZE_AVG

BOX_H = avg (2 opposites sides of the BOX_SIZE_AVG)

BOX_H_STD = max (2 opposites sides of the BOX_SIZE_STD)

BOX_W = avg (2 opposites sides of the BOX_SIZE_AVG)

BOX_W_STD = max (2 opposites sides of the BOX_SIZE_STD)

BOX_D = avg (2 opposites sides of the BOX_SIZE_AVG)

BOX_D_STD = max (2 opposites sides of the BOX_SIZE_STD)

Accurate measurement for box height will be equal to BOX_H + BOX_H_STD

Accurate measurement for box width will be equal to BOX_W + BOX_W_STD

Accurate measurement for box length will be equal to BOX_L + BOX_L_STD

Product Limitations

The DWS measurement system is designed for and tested with wooden and cardboard boxes. Although other materials may work well, avoid using the system with materials that are irregular or highly reflective. Avoid materials such as black plastic wrap and black cardboard which are poor reflectors of IR light.

6. Installation instructions

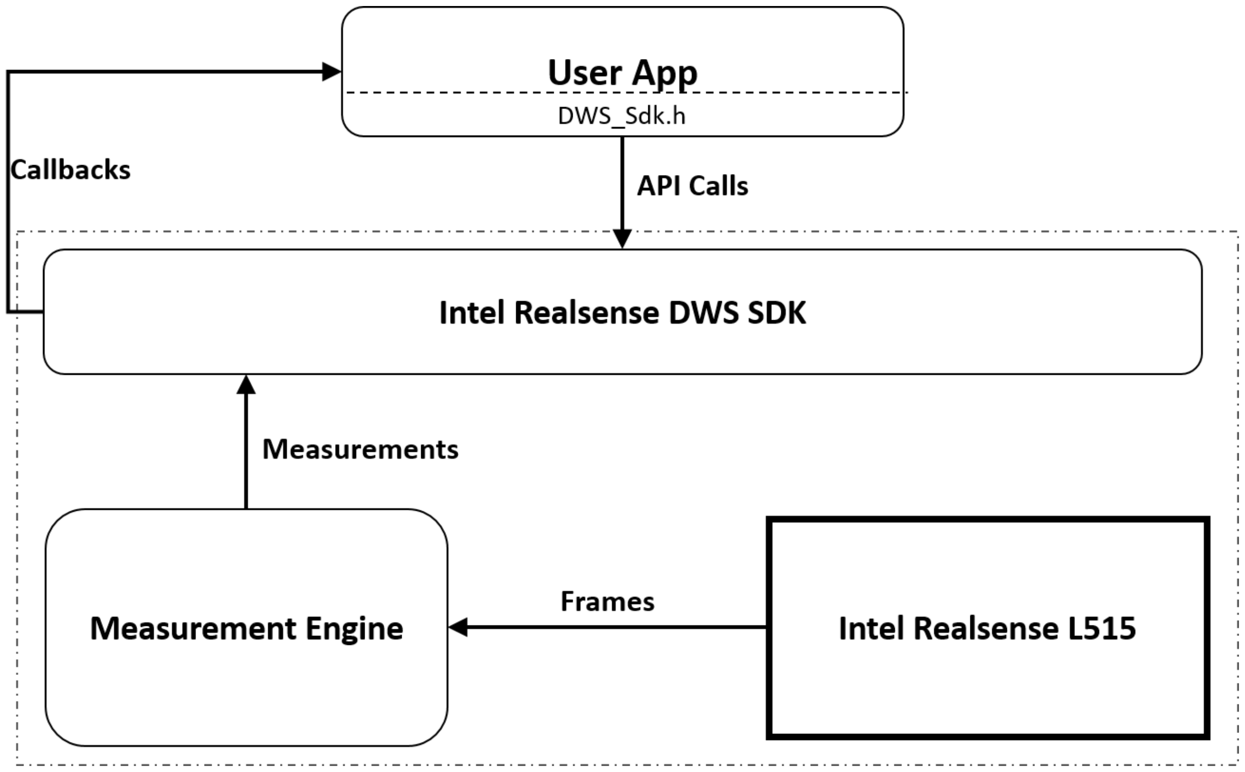
6.1 Download instructions & licensing

DWS along with the necessary license can be downloaded from https://www.intelrealsense.com/dimensional-weight-software/

Refer to Intel® RealSense™ Software Activation Tool Guide for details on how to install trial and purchased licenses

6.2 Integration

Figure 5 Software block diagram



6.3 Interaction with the SDK

The SDK supplies API calls to allow functioning. All API calls are one-directional, returning only completion status.

All video frames and measurement data are sent to the 3rd party app via callbacks.

6.3.1 Required files

- Binaries:
  - All dlls in the download folder
  - Plugins.xml
- Header Files:
  - DWS_Sdk.h
  - DWS_Types.h
  - DEBUG_DWS_sdk.h
- Additional files under /Configuration/
  - BoxMeasure <DIR>
  - Static_mode_config.json

We strongly discourage tampering with the configuration files, as any change in them could deteriorate accuracy and performance.

6.4 Integration

Integration of the SDK was designed to be as simple as possible.

1. Link the DWS_Sdk.dll to your project.
2. Add the header files to an include directory in the project.
3. Make sure the rest of the *.dll files supplied are copied to the build directory (same level as DWS_Sdk.dll)
4. Do the same with plugins.xml.
5. Copy the “Configuration” directory to an appropriate location:
   The binaries will look for it in your app’s runtime “working directory” (usually where the executable file is found).

Note that the streams and their resolutions and formats are hardcoded in the SDK as follows:

- Depth: 640x480 Z16 (16 bits per pixel (bpp))
- IR: 640x480 Y8 (8 bpp)
- Color 1280x720 RGB (24 bpp)

To display the frames returned from the SDK, copy the images to a local buffer. Make sure that the buffers allocated are of sufficient size.

7. System requirements

7.1 Operating system

At launch, the following operating systems will be supported:

- Windows 10 (build 15063 or later)

7.2 Host (PC) configuration

The recommended configuration to optimally run DWS is as follows:

- Intel Core i5 (6th Gen minimum) processor, 8GB memory
    - This assumes a desktop/laptop with sufficient cooling arrangements to make use of the CPU.
    - Ultra-portable laptops that do have Intel Core i5 CPUs, but utilize fanless designs may suffer poor DWS performance

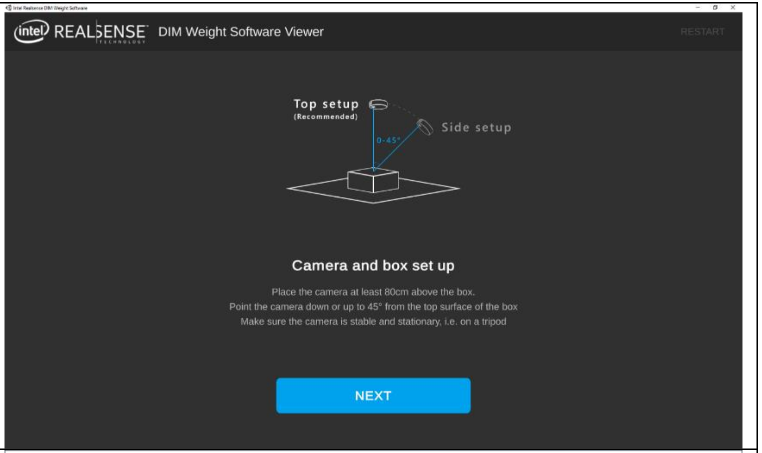
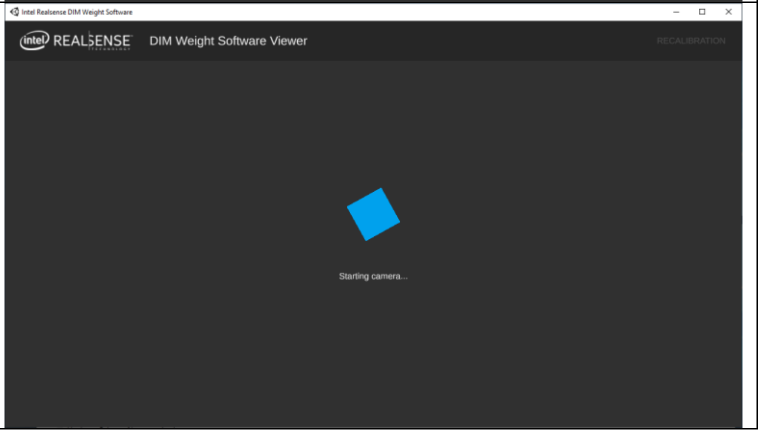
8. Example code

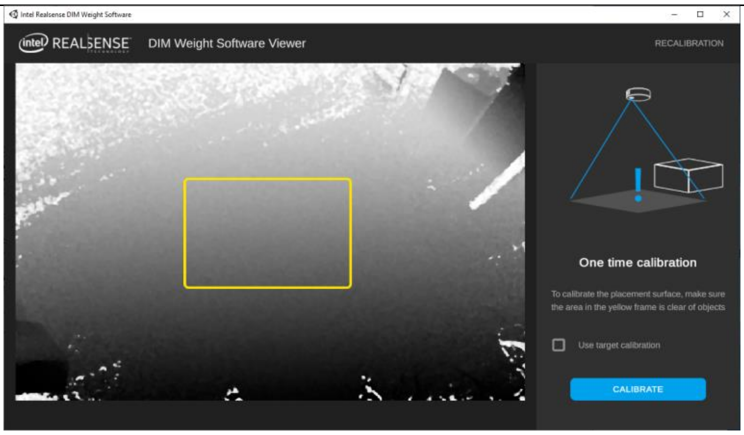
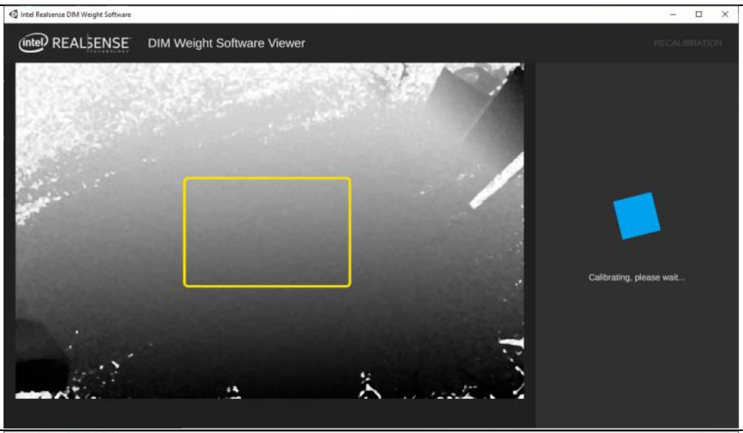
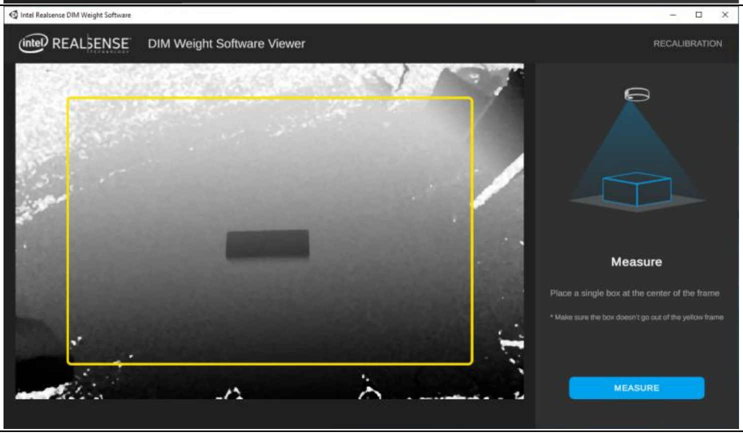
```
/****************************************************************************

INTEL CORPORATION PROPRIETARY INFORMATION
This software is supplied under the terms of a license agreement or nondisclosure
agreement with Intel Corporation and may not be copied or disclosed except in
accordance with the terms of that agreement
Copyright(c) 2020 Intel Corporation. All Rights Reserved.

****************************************************************************/

#include "DWS_sdk.h"
#include <iostream>
#include <mutex>

void * FrameBuffers[3]; // 0 - depth, 1 - IR, 2 - RGB
void * local_depth = new unsigned char[640 * 480 * 2];
void * local_ir = new unsigned char[640 * 480 * 1];
void * local_colour = new unsigned char[1280 * 720 * 3];
bool keep_alive = true;
bool image_ready = false;
std::mutex mutex_display;
DWS_Box boxes[DWS_MAX_NUM_OF_BOXES];
int numOfDetectedBoxes = 0;
long im_id = -1;
int handler = -1;

void * _callback(const void* depth_buffer, int d_w, int d_h, int d_bpp, const void*
ir_buffer, int i_w, int i_h, int i_bpp, const void* colour_buffer, int c_w, int c_h, int
c_bpp, int frameId)
{
	// ################################
	// #### CRITICAL ####
	// COPY FRAMES TO LOCAL BUFFER!!!!
	// ################################
	{
		std::lock_guard<std::mutex> lck(mutex_display);
		memcpy(local_depth, depth_buffer, (d_w *d_h *d_bpp));
		memcpy(local_ir, ir_buffer, (i_w *i_h *i_bpp));
		if (colour_buffer!=nullptr)
			memcpy(local_colour, colour_buffer, (c_w *c_h *c_bpp));
		im_id = frameId;
		image_ready = true;
	}

	return nullptr;
}


void * _dim_callback(int frameId, int box_count, const DWS_Box* box_buffer)
{
	numOfDetectedBoxes = box_count;
	return nullptr;
```

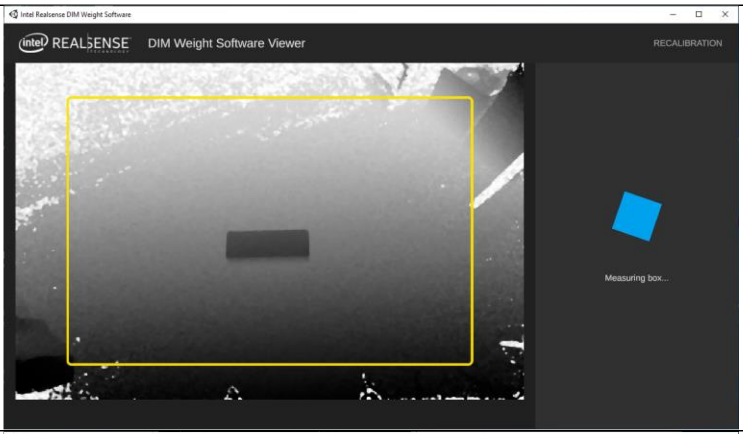
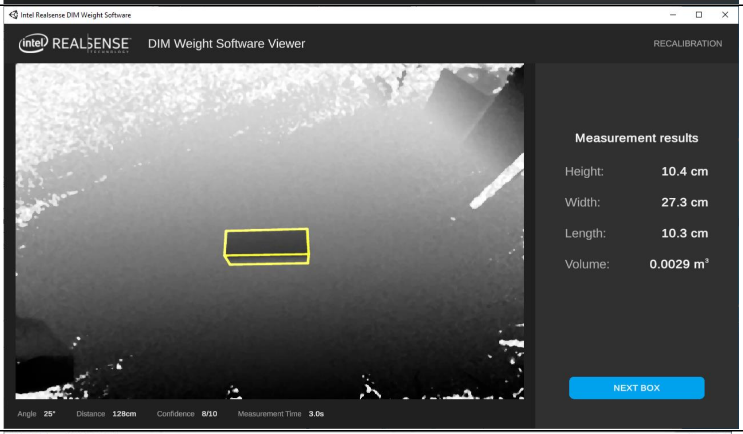
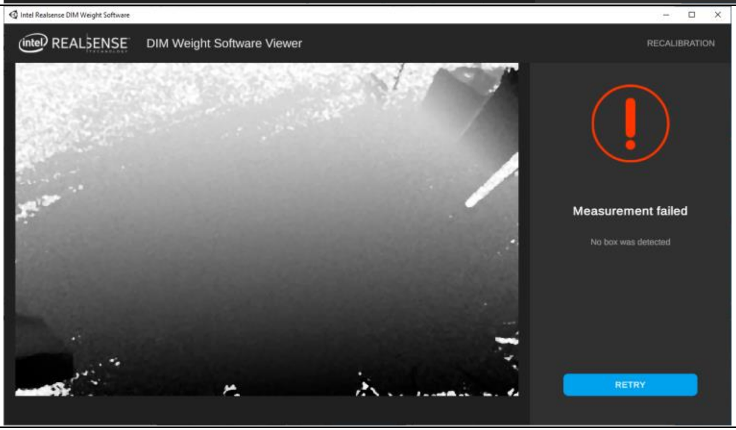
```
}

void* _status_callback(DWS_Severity severity, DWS_MessageCode code, const char* msg)
{
       std::cout << "Severity: " << severity << ", Code: "<< code << ", " << msg <<
std::endl;

       switch (code) {
       case CALIBRATION_FAILED:    // Couldn't complete base surface detection
       case BAD_USER_INPUT:        // Bad input to dws_create
       case CAMERA_DISCONNECTED:   // Detected camera disconnection
       case INVALID_LICENSE:              // License file missing, expired, or doesn't
match camera SN
       case MISSING_CONFIGS:              // Couldn't find the /Configuration/ folder
       case BAD_API_CALL:                 // API handle called during an active earlier
one
       case UNKNOWN_ERROR:                // Internal, exception, msg might contain more
details
              dws_stop(handler);
              keep_alive = false;
       default:
              break;
       }

       if (code == CALIBRATION_SUCCESSFUL)
       {
              if (dws_start(handler, _dim_callback) != DWS_Status::SUCCESS)
                     dws_stop(handler);
       }
       return nullptr;
}

int main(int argc, char* argv[])
{
       int imageNum = 0;
       auto status = dws_create(DWS_Mode::STATIC_MODE, false, _callback,
_status_callback, &boxes[0], handler);
       if (status != DWS_Status::SUCCESS)
       {
              return -1;
       }
       std::string _ver = dws_version(handler);
       dws_autoCalibrate(handler, false);
       while (keep_alive) {
              if (image_ready) {
                     std::lock_guard<std::mutex> lck(mutex_display);
                     std::cout << "new image received. ID = " << im_id << std::endl;
                     image_ready = false;
              }
       }

       return EXIT_SUCCESS;
```

9. }DWS Viewer Application

The SDK includes a DWS viewer application which can be used to test the solution and serve as an example of how the APIs function. The application is not designed for production systems and should only be used as a demonstration application.

The initial application page provides guidelines for best results.	 Intel Realsense DIM Weight Software DIM Weight Software Viewer RESTART Camera and box set up Place the camera at least 80cm above the box. Point the camera down or up to 45° from the top surface of the box Make sure the camera is stable and stationary, i.e. on a tripod NEXT
Clicking Next initializes the camera.	 Intel Realsense DIM Weight Software DIM Weight Software Viewer RECALIBRATION Starting camera...

As part of the initialization process, the software will calibrate to the floor. It is important that the camera have a clear view of the floor for this step. The live view in the application is the depth map of the scene. It is important that the yellow rectangle is free of any objects. Check the box if a target is used for calibration (see Section 4 in this document). If the camera moves after calibration a recalibration is required. This is accomplished by clicking RECALIBRATION at the top right corner of the application.	 Intel Realsense DIM Weight Software DIM Weight Software Viewer RECALIBRATION One time calibration Use target calibration CALIBRATE To calibrate the placement surface, make sure the area in the yellow frame is clear of objects
The camera needs a moment to calibrate to the floor.	 Intel Realsense DIM Weight Software DIM Weight Software Viewer RECALIBRATION Calibrating, please wait...
This screen indicates that the system is ready to measure a box. Place a box on the floor within the yellow rectangle and click the Measure button.	 Intel Realsense DIM Weight Software DIM Weight Software Viewer RECALIBRATION Measure MEASURE Place a single box at the center of the frame * Make sure the box doesn't go out of the yellow frame

The application only takes a moment to measure the box.	 Intel Realsense DIM Weight Software DIM Weight Software Viewer RECALIBRATION Measuring box...
Measurement results are displayed on the screen. Details including the angle to the top surface, confidence, and execution time are also shown. This information is available through the SDK.	 Intel Realsense DIM Weight Software DIM Weight Software Viewer RECALIBRATION Measurement results Height: 10.4 cm Width: 27.3 cm Length: 10.3 cm Volume: 0.0029 m³ Angle 25° Distance 128cm Confidence 8/10 Measurement Time 3.0s
Error messages are displayed when common issues are encountered. Possible “bad box” errors include: • Box is too close/far • Box is too large/small • Box detected out of central part of FOV • Camera to box angle is greater than 45deg.	 Intel Realsense DIM Weight Software DIM Weight Software Viewer RECALIBRATION Measurement failed No box was detected

10. Troubleshooting

10.1 Application debug

The DWS Viewer application has the ability to save log files for debug purposes. It is recommended to only enable debug mode when necessary. To enable debug mode open the file AppConfiguration.json in the SDK directory. Change the value of "`Development`" from `false` to `true`, save the file, and start the application. Log files are saved in C:\temp\Realsense\Logs\.

If the system does not obtain a measurement of high confidence it will continue to try until MeasureTimeOut expires. For debugging purposes it may be useful to extend the time out value from the default of 3 seconds. MeasureTimeOut is located in the AppConfiguration.json file. Simply modify the file using a text editor, save, and run the application with the new time out value.

10.2 SDK Debug

The DWS SDK has the ability to save log files and record images for debug purposes.
Note that using this ability will quickly fill your storage.

To make use of this functionality:

- Include DEBUG_DWS_sdk.h
- Call debug_ debug_dws_startDebugRecordingSession() to start the session
- Call debug_ debug_dws_endDebugRecordingSession() to end the session.

Note: Only start/end debug sessions while no DWS operations are active! i.e before dws_start/calibrate calls have been made, or after dws_pause has been called and returned.

10.3 SDK error messaging

The SDK includes error messages and warning messages returned through `dws_statusCallback`.

CALIBRATION_INVALIDATED (error)	The camera has moved during processing to render the previous calibration invalid. Recalibration is required.
NO_CALIBRATION_FOUND (warning)	This warning is expected if a measurement is attempted before calibration is performed. If this warning is encountered, a calibration is required.
CALIBRATION_DONE_NO_MARKERS (warning)	Indicates that the calibration was successful and that a target was not used for calibration.
CALIBRATION_FAILED (error)	Calibration was attempted and failed.

	If a target was used for calibration make sure the target has no bends and creases. If no target is being used confirm in live depth preview that the floor or table-top are providing a good depth map. If depth map is poor, consider moving away from sunlit areas. Confirm no objects other than the target are inside the preview yellow box during calibration.
CAMERA_DISCONNECTED (error)	Confirm that the L515 camera USB cable is properly connected. Confirm that the L515 camera shows in device manager and a valid camera device. The application requires a restart.
INVALID_LICENSE (error)	Attempting to use DWS without a valid license. A license can be obtained from https://www.intelrealsense.com/dimensional-weight-software/.
LICENSE_EXPIRES_SOON (warning)	License will expire within 10 days. A new license will be required to continue using DWS after the current license expires.
MISSING_CONFIGS (error)	The SDK cannot find the necessary configuration files, or they are corrupt/syntactically invalid. Place the files in the correct location and confirm files are valid.
UNEXPECTED_BOXES (warning)	SDK was started in single box mode, but multiple boxes were detected in the frame.
BAD_BOX (warning)	This warning will be accompanied by one of the following messages giving more information about the issue "Box is too close" "Box is too far" "Box is too large/small" "Box detected out of central part of FOV" "Camera to box angle is greater than 45 deg":

BAD_USER_INPUT	Occurs when invalid input is given to dws_create()
BAD_API_CALL	Indicates that an API call has been made out of order. Either a previous step hasn’t been completed, or a previous API call is still running
UNKNOWN_ERROR (error)	The SDK suffered an exception and is unable to recover. Restart the application and contact support if the issue continues.