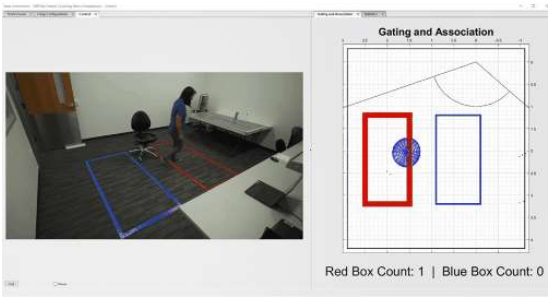
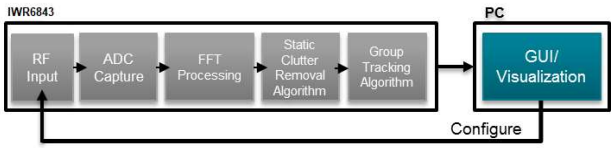
Overview

This lab demonstrates the use of TI mmWave sensors to count and track up to 3 people simultaneously and detect their stance. Detection and tracking algorithms run onboard the IWR6843ODS ES1.0 mmWave sensor and are used to localize people and track their movement with a high degree of accuracy. This demo is confgured to have the IWR6843ODS mounted overhead, with the antennas facing down. mmWave sensors can reduce false detections from challenging environments such as direct sunlight, no-light, fog, or smoke, and are particularly suited for privacy-conscious applications. In this demonstration, localization and tracking is performed upon any moving object in the scene; static objects such as chairs, tables, and walls are ignored. The IWR6843 ES1.0 device outputs a data stream consisting of point cloud information and a list of tracked objects which can be visualized using the software included in this lab.

> **Device Version: ES1.0**
>
> This lab is only compatible with ES1.0 Devices.





Quickstart

1. Hardware and Software Requirements

Hardware

Item	Details
Device	Industrial mmWave Carrier Board (http://www.ti.com/tool/MMWAVEICBOOST) and IWR6843 ES1.0 ODS (http://www.ti.com/tool/IWR6843ISK).
Mounting Hardware	The EVM needs to be mounted at 3 meters with the antennas facing down.
Computer	PC with Windows 7 or 10. If a laptop is used, please use the 'High Performance' power plan in Windows.
Micro USB Cable	Due to the high mounting height of the EVM, an 8ft+ cable or USB extension cable is recommended.
Power Supply	5V, >2.5A with 2.1-mm barrel jack (center positive). The power supply can be wall adapter style or a battery pack with a USB to barrel jack cable.

Software

Tool	Version	Required For	Download Link
mmWave Industrial Toolbox	Latest	Contains all lab material.	mmWave Industrial Toolbox (http://dev.ti.com/tirex/explore/node?node=AJoMGA2ID9pCPWEKPi16wg__VLyFKFf__LATEST)
MATLAB Runtime	2017a (9.2)	Quickstart Visualizer	To run the quickstart visualizer the runtime (https://www.mathworks.com/products/compiler/matlab-runtime.html) is sufficient.
Uniflash	Latest	Quickstart Firmware	Download offline tool (http://www.ti.com/tool/UNIFLASH) or use cloud version (https://dev.ti.com/uniflash/#!/)

2. Physical Setup

1. Follow the instructions for Hardware Setup of ICB for Functional Mode (../../../common/docs/hardware_setup/hw_setup_antenna_module_and_carrier_for_functional.html)

Setup Requirements:

- Elevate EVM: 3m high
- Down tilt: 90 degree (Antennas Parallel with the Ground)

3. Flash the EVM

- Follow the instructions for Hardware Setup of ICB for Flashing Mode (../../../common/docs/hardware_setup/hw_setup_antenna_module_and_carrier_for_flashing.html)
- Follow the instruction to Flash the mmWave Device (../../../common/docs/software_setup/using_uniflash_with_mmwave.html)

Image	Location
Meta Image 1/RadarSS	`C:\ti\<mmwave_industrial_toolbox_install_dir>\labs\people_counting\68xx_overhead_people_counting\Pre-compiled binary\ODS\ods_pplcount_lab_xwr68xx.bin`

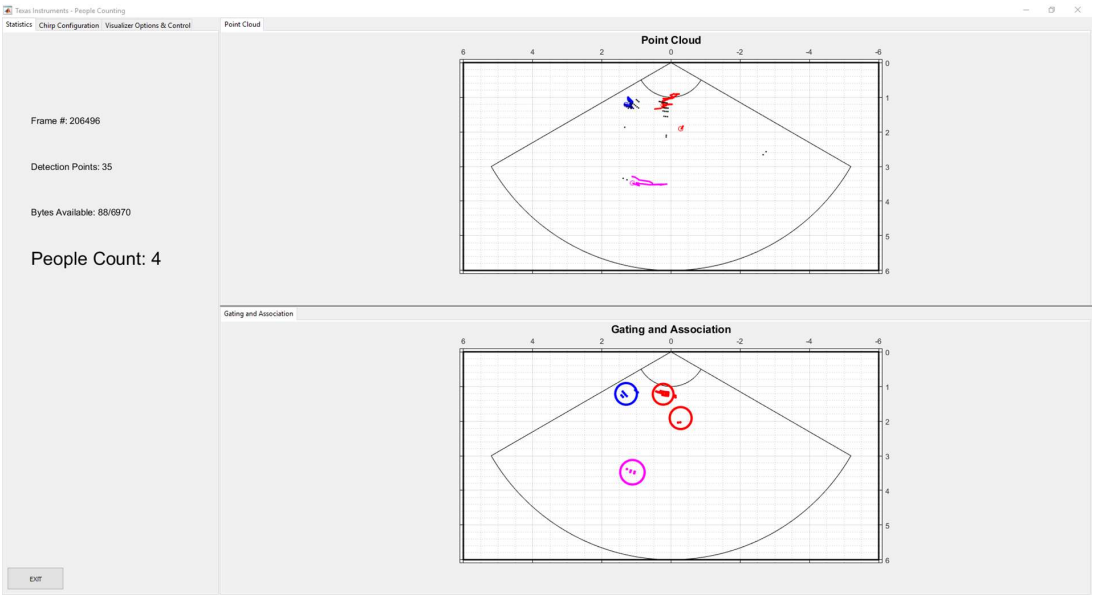
4. Run the Lab

To run the lab, launch and configure the visualizer which displays the detection and tracked object data received via UART.

1. Launch the visualizer:

- Navigate to `C:\ti\<mmwave_industrial_toolbox_install_dir>\labs\people_counting\68xx_overhead_people_counting\GUI_A_base_no_logic\main_pplcount_viz.exe`
- Run `main_pplcount_viz.exe`
- A black console log window will appear.
- After a few seconds, the Visualizer will begin running.

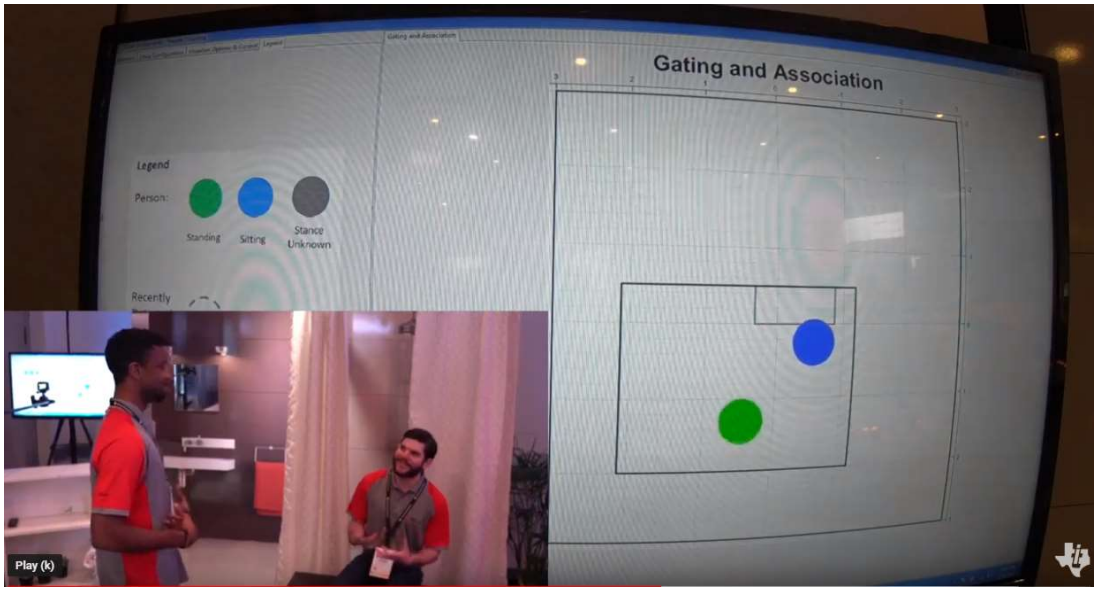
- Understanding the Output GUI A - default GUI



The visualizer consists of:

- A top panel with a **Point Cloud** plot.
  - The black points represent the point cloud returned by the detection layer of the device.
    - Each new tracked object is assigned one of five possible colors (blue, red, green, cyan, and magenta).
    - The small colored ring represents the computed centroid of the point cloud for the tracked object.
    - The "snail trail" trace represent 100 frames of history of the tracked object's centroid.
- A bottom panel with a **Gating and Association** plot.
  - This plot visualizes the result of the tracking algorithm.
    - The colored points represent the detection points (black points in previous frame's Point Cloud plot) which are associated to a specific track. Unassociated points that do not belong to a track are not plotted.

    - The colored circular ring is centered over the centroid of of the tracked object. The diameter of the ring is related to the variance in location of the tracked object's detection points.
- A side panel with three tabs: Statistics, Chirp Configuration, and Visualizer Options
    - TIP: If lag is an issue, check the **Consolidate plotting** option in **Visualizer Options**. This will only display one of the two plots. **Quitting the Visualizer** : To exit the visualizer use the exit button at the bottom left of the window. This will delete the open serial ports and save an output file of the session in fhist.mat.
- Understanding the Output GUI B - GUI with filtering



If using the .exe, the guiCfg.txt file will not be used. Instead, COM Ports must be input in a dialog box. Static area will be limited to 2 meters left of the device, 2 meters right of the device, and 2 meters in front of the device (Top of antenna pattern).

Find this visualizer at: C:\ti\labs\people_counting\68xx_overhead_people_counting\GUI_B_gui_filteringLogic\main_pplcount_viz.exe`

The visualizer consists of:

- A main panel with a **Gating and Association** plot.
    - This plot visualizes the result of the tracking algorithm.
    - Hollow Circles with dotted lines represent newly allocated tracks - this will be colored based on stance.
    - Filled circles represent tracks that the tracker is confident in.
    - Hollow Grey circles with full lines represent static people who are no longer creating radar detection points.
    - Each new tracked object is assigned one of 3 possible colors (blue, green, or grey), based on the Stance of the target.
        - Grey = Stance Unknown
        - Green = Standing
        - Blue = Sitting
- A side panel with four tabs: Statistics, Chirp Configuration, Visualizer Options, and a Legend

Quitting the Visualizer : To exit the visualizer use the exit button at the bottom left of the window. This will delete the open serial ports and save an output file of the session in fhist.mat.

Developer's Guide

Build the Firmware from Source Code

1. Software Requirements

Tool	Version	Download Link
Overhead Stance Detection Lab	1.0.0	Provided
TI mmWave SDK	3.2.0.x	TI mmWave SDK (http://software-dl.ti.com/ra-processors/esd/MMWAVE-SDK/03_02_00_04/index_FDS.html) and all the related tools are required to be installed as specified in the mmWave SDK release notes (http://software-dl.ti.com/ra-processors/esd/MMWAVE-SDK/latest/exports/mmwave_sdk_release_notes.pdf)
Code Composer Studio	8.1.0	Code Composer Studio v8 (http://processors.wiki.ti.com/index.php/Download_CCS#Code_Composer_Studio_Version_8_Downloads)
TI SYS/BIOS	6.52.0.12	Included in mmWave SDK installer
TI ARM Compiler	16.9.1.LTS	Included in mmWave SDK installer
TI CGT Compiler	8.1.3	Included in CCS
XDC	3.50.08.24	Included in mmWave SDK installer
C64x+ DSPLIB	3.4.0.0	Included in mmWave SDK installer
C674x DSPLIB	3.4.0.0	Included in mmWave SDK installer
C674x MATHLIB (little-endian, elf/coff format)	3.1.2.1	Included in mmWave SDK installer
mmWave Radar Device Support Package	1.6.1 or later	Upgrade to the latest using CCS update process (see SDK user guide for more details)
TI Emulators Package	7.0.188.0 or later	Upgrade to the latest using CCS update process (see SDK user guide for more details)
Uniflash	Latest	Uniflash tool is used for flashing TI mmWave Radar devices. Download offline tool (http://www.ti.com/tool/UNIFLASH) or use the Cloud version (https://dev.ti.com/uniflash/#!/)

2. Import Lab Project

For the People Counting lab, there are two projects, the DSS for the C674x DSP core and the MSS project for the R4F core, that need to be imported to CCS and compiled to generate firmware for the xWR6843.

> **Project Workspace**
>
> When importing projects to a workspace, a copy is created in the workspace. All modifications will only be implemented for the workspace copy. The original project downloaded in mmWave Industrial Toolbox is not touched.

- Start CCS and setup workspace as desired.
- Import the project(s) specified below to CCS from the src/ folder. See instructions for importing here (../../../../docs/readme.html#import-ccs-projects-from-the-mmwave-industrial-toolbox-into-code-composer-studio).
  - **ods_pplcount_dss_xwr68xx**
  - **ods_pplcount_mss_xwr68xx**
- Verify that the import occurred without error: in CCS Project Explorer, both **ods_pplcount_dss_xwr68xx** and **ods_pplcount_mss_xwr68xx** should appear.

3. Build the Lab

The DSS project must be built before the MSS project.

1. Select the **ods_pplcount_dss_68xx** so it is highlighted. Right click on the project and select **Rebuild Project**. The DSS project will build.

2. Select the **ods_pplcount_mss_68xx** so it is highlighted. Right click on the project and select **Rebuild Project**. The MSS project will build, the the lab binary will be constructed automatically.
3. On successful build, the following should appear:
    - In ods_pplcount_dss_68xx → Debug, **ods_pplcount_dss_68xx.xe674** (this is the C67x binary used for CCS debug mode)
        - In ods_pplcount_mss_68xx → Debug, **ods_pplcount_mss_68xx.xer4f** (this is the Cortex R4F binary used for CCS debug mode) and **ods_pplcount_lab_68xx.bin** (this is the flashable binary used for deployment mode)

Selecting Rebuild instead of Build ensures that the project is always re-compiled. This is especially important in case the previous build failed with errors.

Build Fails with Errors

If the build fails with errors, please ensure that all the software requirements are installed as listed above and in the mmWave SDK release notes.

Note

As mentioned in the Quickstart section, pre-built binary files, both debug and deployment binaries are provided in the pre-compiled directory of the lab.

4. Execute the Lab

There are two ways to execute the compiled code on the EVM:

- Deployment mode: the EVM boots autonomously from flash and starts running the bin image
    - Using Uniflash, flash the **ods_pplcount_lab_68xx.bin** found at
      `<PROJECT_WORKSPACE_DIR>\ods_pplcount_mss_68xx\Debug\pplcount_lab_68xx.bin`
    - The same procedure for flashing can be use as detailed in the Quickstart Flash the EVM section.
- Debug mode: Follow the instructions for Using CCS Debug for Development (../../../common/docs/software_setup/using_ccs_debug.html)

After executing the lab using either method, the lab can be visualized using the Quick Start GUI or continue to working with the GUI Source Code

Visualizer Source Code

Working with and running the Visualizer source files requires a MATLAB License not just the MATLAB Runtime Engine

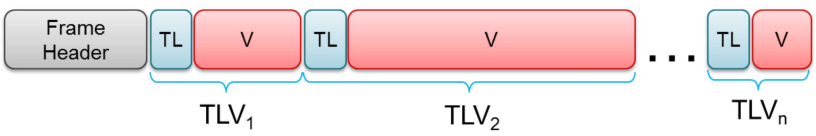
The detection processing chain and group tracking algorithm are implemented in the firmware. The visualizer serves to read the UART stream from the device and then plot the detected points and tracked objects.

Source files are located at
`C:\ti\mmwave_industrial_toolbox_<VER>\labs\people_counting\68xx_overhead_people_counting\GUI_A_base_no_logic .`

- **main_pplcount_viz.m**: the main program which reads and parses the UART data for visualization
- **setup.m**, **setup.fig**: creates the visualizer configuration window used in GUI setup mode where user can input setup parameters
- **mmw_pplcount_demo_default.cfg**: configuration file

Data Formats

A TLV(type-length-value) encoding scheme is used with little endian byte order. For every frame, a packet is sent consisting of a fixed sized **Frame Header** and then a variable number of TLVs depending on what was detected in that scene. The TLVs can be of types representing the 3D point cloud, target list object, and associated points.



Frame Header

Size: 52 bytes

```
frameHeaderStructType = struct(...
    'sync',             {'uint64', 8}, ... % syncPattern in hex is: '02 01 04 03 06 05 08 07'
    'version',          {'uint32', 4}, ... % 0xA6843
    'platform',         {'uint32', 4}, ... % See description below
    'timestamp',        {'uint32', 4}, ... % 600MHz free running clocks
    'packetLength',     {'uint32', 4}, ... % In bytes, including header
    'frameNumber',      {'uint32', 4}, ... % Starting from 1
    'subframeNumber',   {'uint32', 4}, ...
    'chirpMargin',      {'uint32', 4}, ... % Chirp Processing margin, in ms
    'frameMargin',      {'uint32', 4}, ... % Frame Processing margin, in ms
    'uartSentTime' ,    {'uint32', 4}, ... % Time spent to send data, in ms
    'trackProcessTime', {'uint32', 4}, ... % Tracking Processing time, in ms
    'numTLVs' ,         {'uint16', 2}, ... % Number of TLVs in thins frame
    'checksum',         {'uint16', 2});    % Header checksum
```

Frame Header Structure in MATLAB syntax for name, type, length

```
% Input: frameheader is a 52x1 double array, each index represents a byte of the frame header
% Output: CS is checksum indicator. If CS is 0, checksum is valid.

function CS = validateChecksum(frameheader)
    h = typecast(uint8(header),'uint16');
    a = uint32(sum(h));
    b = uint16(sum(typecast(a,'uint16')));
    CS = uint16(bitcmp(b));
end
```

validateChecksum(frameheader) in MATLAB syntax

TLVs

The TLVs can be of type **POINT_CLOUD_3D**, **TARGET_LIST_3D**, or **TARGET_INDEX**.

TLV Header

Size: 8 bytes

```
% TLV Type: 06 = Point cloud, 07 = Target object list, 08 = Target index
tlvHeaderStruct = struct(...
    'type',             {'uint32', 4}, ... % TLV object
    'length',           {'uint32', 4});    % TLV object Length, in bytes, including TLV header
```

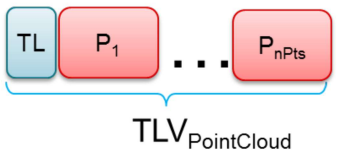
TLV header in MATLAB syntax

Following the header, is the the TLV-type specific payload

Point Cloud TLV

Type: POINT_CLOUD_3D

Size: sizeof (tlvHeaderStruct) + sizeof (pointStruct3D) x numberOfPoints



Each Point Cloud TLV consists of an array of points. Each point is defined in 16 bytes.

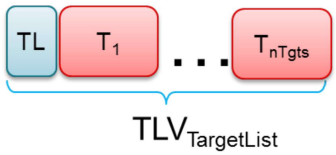
```
pointStruct3D = struct(...
    'range',            {'float', 4}, ... % Range, in m
    'azimuth',          {'float', 4}, ... % Angle, in rad
    'elevation'         {'float', 4}, ... %Elevation, in rad
    'doppler',          {'float', 4}, ... % Doppler, in m/s
    'snr',              {'float', 4});    % SNR, ratio
```

Point Structure in MATLAB syntax

Target Object TLV

Type: TARGET_LIST_3D

Size: sizeof (tlvHeaderStruct) + sizeof (targetStruct3D) x numberOfTargets



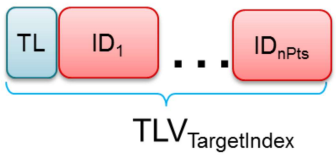
Each Target List TLV consists of an array of targets. Each target is defined in 68 bytes.

```
targetStruct3D = struct(...
    'tid',              {'uint32', 4}, ... % Track ID
    'posX',             {'float', 4}, ... % Target position in X dimension, m
    'posY',             {'float', 4}, ... % Target position in Y dimension, m
    'posZ',             {'float', 4}, ... % Target position in Z dimension, m
    'velX',             {'float', 4}, ... % Target velocity in X dimension, m/s
    'velY',             {'float', 4}, ... % Target velocity in Y dimension, m/s
    'velZ',             {'float', 4}, ... % Target velocity in Z dimension, m/s
    'dimX',             {'float', 4}, ... % Target size in X dimension, m
    'dimY',             {'float', 4}, ... % Target size in Y dimension, m
    'dimZ',             {'float', 4}, ... % Target size in Z dimension, m
```

Target Structure in MATLAB syntax

Target Index TLV

Type: TARGET_INDEX Size: sizeof (tlvHeaderStruct) + numberOfPoints (NOTE: here the number of points are for frame n-1)



Each Target List TLV consists of an array of target IDs. A targetID at index ***i*** is the target to which point ***i*** of the previous frame's point cloud was associated. Valid IDs range from 0-249.

```
targetIndex = struct(...
'targetID',         {'uint8', 1});    % Track ID
```

Target ID Structure in MATLAB syntax

Other Target ID values:

Value	Meaning
253	Point not associated, SNR too weak
254	Point not associated, located outside boundary of interest
255	Point not associated, considered as noise

Need More Help?

- Find answers to common questions on mmWave E2E FAQ (https://e2e.ti.com/support/sensor/mmwave_sensors/w/wiki)
- Search for your issue or post a new question on the mmWave E2E forum (https://e2e.ti.com/support/sensor/mmwave_sensors/f/1023)
- See the SDK for more documentation on various algorithms used in this demo. Start at <MMWAVE_SDK_DIRECTORY>/docs/mmwave_sdk_module_documentation.html>