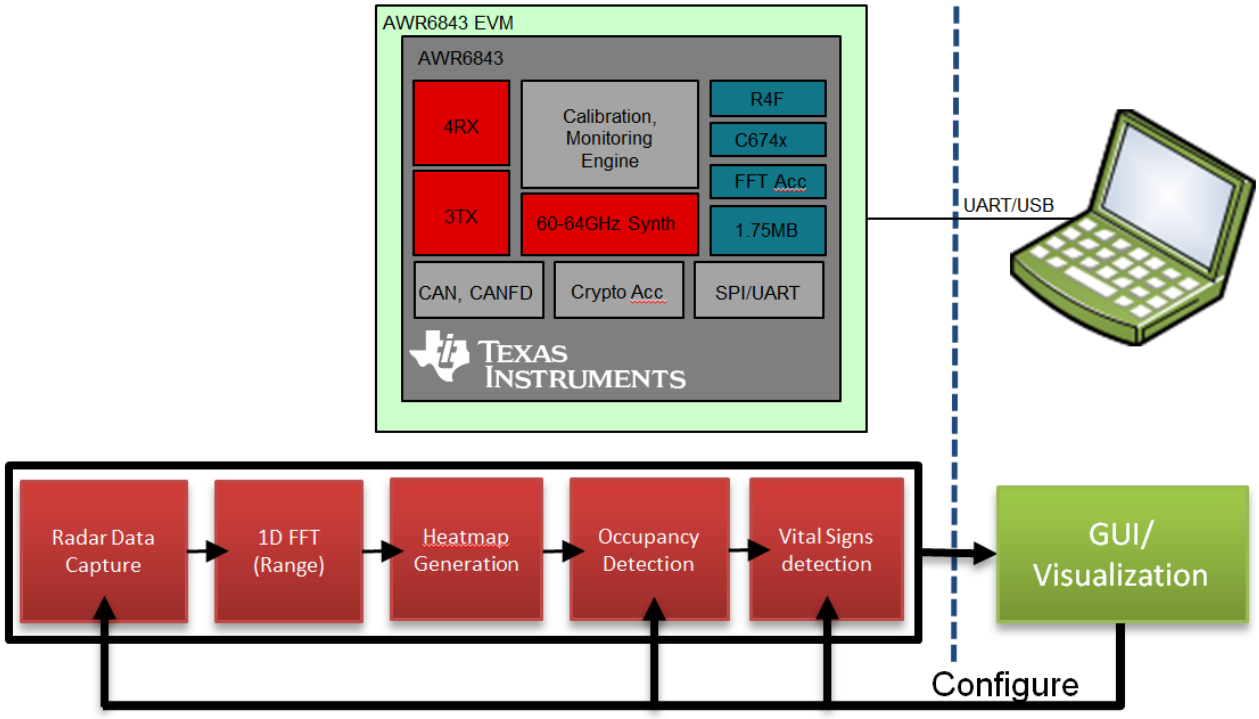
Overview

This lab combines two labs, Vehicle Occupant Detection and Driver Vital Signs, into a single application and GUI. You can find these two individual labs on TI's Resource Explorer in the mmWave Sensors Automotive Toolbox. This release supports two zones for both occupancy detection and vital signs.

This lab demonstrates the use of TI mmWave sensors to detect people in pre-defined zones of coverage and measure their vital signs such as heart rate and breathing rate. Using the AWR6843, algorithms run onboard the single-chip device to create an Range-Azimuth heatmap, then extract features, detections, and vital signs information from the heatmap. Chirp configurations are selected to detect slight movements (like breathing) with a high degree of accuracy.

The Occupancy and Vital Signs Detection Demo generates an angular heatmap that is displayed using a MATLAB GUI. Two zones of interest are defined, and algorithms examine the heatmap to provide frame by frame occupancy decisions. Each zone of interest is identified as occupied or not along with a breathing and heart rate if the zone is occupied by a person.



For the purposes of this lab, this release has been developed to support one pair of zones for both occupancy detection and vital signs.

Quickstart

The quickstart contains:

- Precompiled binaries for flashing the device using Uniflash
- Visualizer as .exe

1. Hardware and Software Requirements

Hardware

Item	Details

Item	Details
Device	AWR6843 EVM, AWR1843 EVM or AWR1642 EVM
Mounting Hardware	The EVM needs to be mounted at chest height. An adjustable clamp style smartphone adapter mount for tripods and a 60-75" tripod can be used to clamp and elevate the EVM. This is only an example solution for mounting; other methods can be used so far as setup specifications are met.
Computer	PC with Windows 7 or 10. If a laptop is used, please use the 'High Performance' power plan in Windows. 2.4Ghz processor, 8GB RAM recommended.
Micro USB Cable	For safety during testing, it is recommended to use an extension cable long enough to route it out of the test vehicle without hindering entry and exit.
Power Supply	5V, 3A 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.
Pulse Oximeter	Used to compare the output from the GUI. This example device attaches to the index finger.

Software

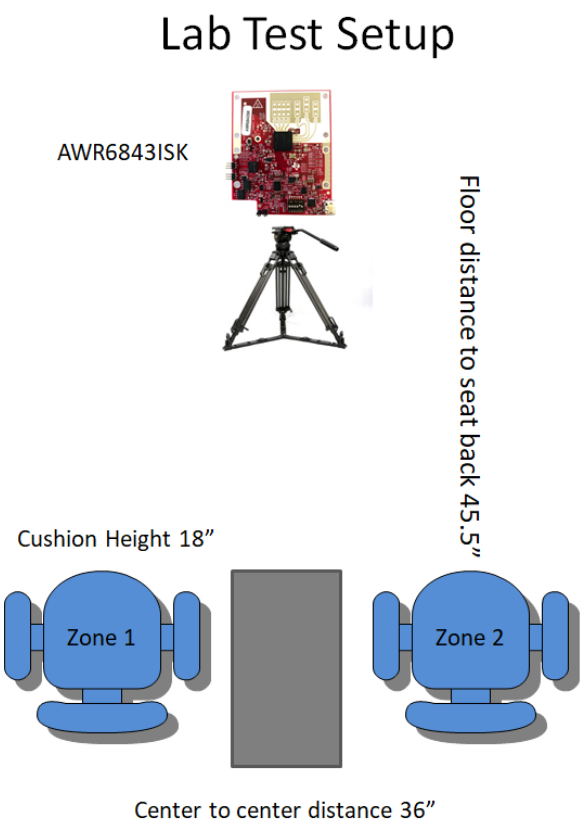
Tool	Version	Required For	Details
mmWave Automotive Toolbox	3.0 or later	ES2.0 silicon	Contains all files (quickstart, visualizer and firmware source files) related to this Lab
MATLAB Runtime	2017a (9.2)	GUI Visualizer	To run the quickstart visualizer the runtime is sufficient.
TI mmWave SDK	3.4.0.3+	Firmware Source Code	The latest TI mmWave SDK and all the related tools are required to be installed as specified in the mmWave SDK release notes
TI Emulators package	7.0.188.0 or later	-	Upgrade to the latest using CCS update process (see SDK user guide for more details)

Reference Material

Item	Details
Vehicle Occupant Detection Lab	TIREX Release for Vehicle Occupant Detection Lab
Driver Vital Signs Lab	TIREX Release for Driver Vital Signs Lab
VOD Design Document	TI Design for Vehicle Occupant Detection

2. Physical Setup

For best results, the EVM should be positioned as shown below, with the sensor placed at chest height when seated. This is because the zones and coefficients in the chirp configuration are tuned to the scene that is being monitored.



Chirp Configuration	Test Scene	Description
vod_vital_signs_10fps.cfg	Lab	Configuration that runs at 10 frames per second

3. Flash the Device

- XDS Emulation software is required for flashing if CCS (Code Composer Studio) is not installed (CCS installs it automatically). XDS Emulation software can be found here.
- Power on the EVM using a 5V/3.0A power supply.
- Flash the following image using **Uniflash**

Image	Location
Meta Images	`C:<install_dir>\occupancy_plus_vital_signs\prebuilt_binaries\vod_vital_signs_68xx.bin`

Expand for help using Uniflash

4. Choose Visualizer Setup Settings

Before running the demo either in Matlab or from DOS, you will need to know the COM port numbers for the EVM's User and Data UART ports. This is discussed in the pulldown section above titled "Expand for help using Uniflash". Once found, the COM port numbers will usually not change from run to run or boot to boot. These are the command line arguments for the Occupancy plus Vital Signs demo gui:

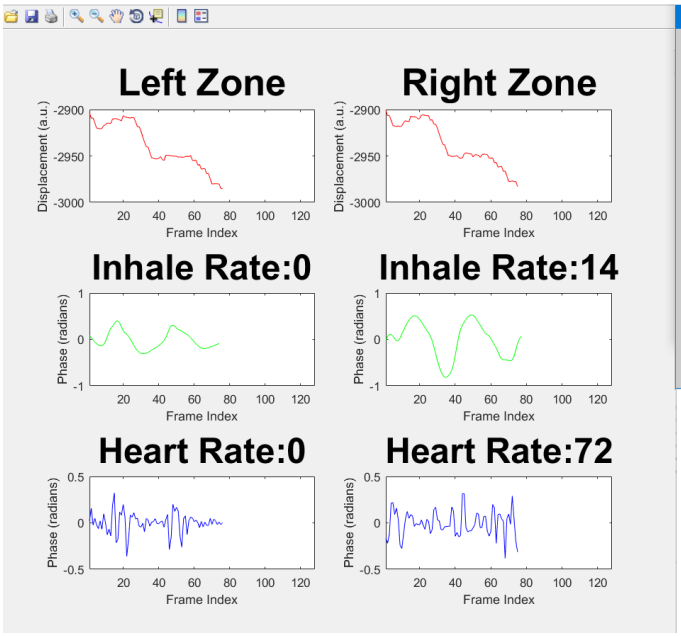
Note, Matlab Runtime is required to be installed if a full license of Matlab is not installed. See the Software section above for the download link.

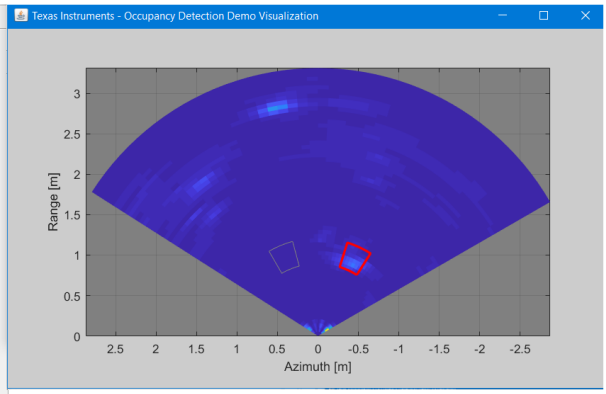
Argument	Example/Purpose
Program Name	vod_vital_signs (it will either be a .exe (for DOS) or .m (for Matlab))

Argument	Example/Purpose
COM port (Data)	3
COM port (User)	4
Chirp Configuration	`C:<install_dir>\occupancy_plus_vital_signs\chirp_configs\vod_vs_68xx_10fps.cfg`
Heatmap Type	1 (0 = none, 1 = Polar, 2 = Rectangular)

5. Run the Lab Visualizer (DOS prompt)

- The following steps assume the AWR6843 EVM is flashed with the Occupancy plus Vital Signs firmware.
- Mount the EVM as shown above in the Test Setups.
- Attach the micro USB cable from the EVM to the host PC.
- Attach the 5V power adapter cable to the EVM.
- Open a DOS Command Window, and cd to `C:<install_dir>\occupancy_plus_vital_signs\gui`.
- At the DOS prompt, enter a command like this:
    - vod_vital_signs.exe 3 4 ..\chirp_configs\vod_vs_68xx_10fps.cfg 1
- After a successful start, the visualizer will look like this (polar plot is selected).





To terminate the demo, perform the following steps:

- Click inside the GUI to get focus.
- Press lowercase q.
- Click the "Exit" icon (X) in the top right of the Visualization window.

Note: You may see "Buffer overflow" or "Packets lost" messages when running the MATLAB GUI. These errors occur when MATLAB takes too long to draw the GUI displays, causing data from the target to be dropped. If you encounter this, try closing other programs or use a faster machine. Resizing a GUI window causes Matlab processing to pause, so it is normal to see errors in this case. If you are collecting row noise data, dropped messages can mean you might miss the rowNoise values. Simply restart the demo in this case.

6. Understanding the Output

The GUI for this lab has two displays: the heatmap display (shown on the right) and the vital signs display (shown on the left).

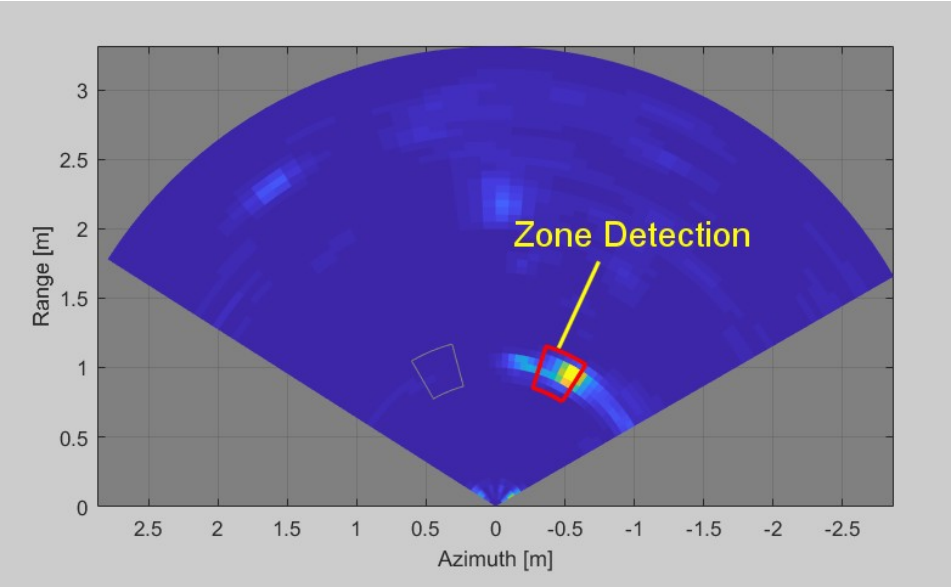
The heatmap display is automatically scaled to the range of values within the heatmap for each frame. Blues are smaller values and Yellows are large. A bright spot indicates a higher energy return that has not been removed by the clutter removal algorithm.

The vital signs display shows three real-time plots for each zone. The waveforms that are plotted include the unwrapped phase values, the breathing waveform, and the heart rate waveform (shown from top to bottom). For each zone, the calculated breathing rate and heart rates values are displayed above their respective waveform plots. More information about these plots can be found in the Driver Vital Signs TIREX lab.

At times when there is no person in the zone a "bright spot" can actually be relatively small values because of the automatic scaling. When a person enters the scene, the scaling will adjust and the person will be shown as a bright spot with everything else fading to smaller value colors.

The provided configurations define two zones, and they are presented from the user's point of view. So when you sit in the zone on the right side of the EVM (as you face it), the bright spot in the heatmap will appear in the right zone.

When no detection is calculated for a zone, the zone will have a gray border. When there is a detection, it will be turn red if the decision is coming from the AWR6843, or purple if generated by Matlab.



Developer's Guide

Building the Firmware from Source Code

1. Prerequisites for Firmware

> The software prerequisites must be met before continuing!

To verify proper installations, navigate to `C:\ti` and ensure that the following tools have been installed in the *EXACT* directory specified.

Tool	Version	Folder Path	Download link & Details
CCS	9.2 or later	`C:\ti\ccsv9`	Download link Note: CCSv6.x cannot be used
TI SYS/BIOS	6.73.01.01	`C:\ti\bios_6_73_01_01`	Included in mmwave sdk installer
TI ARM compiler	16.9.6.LTS	`C:\ti\ti-cgt-arm_16.9.6.LTS`	Included in mmwave sdk installer
TI DSP compiler	8.3.3	`C:\ti\ti-cgt-c6000_8.3.3`	Included in mmwave sdk installer
XDC	3.50.08.24	`C:\ti\xdctools_3_50_08_24_core`	Included in mmwave sdk installer

Tool	Version	Folder Path	Download link & Details
C64x+ DSPLIB	3.4.0.0	`C:\ti\dsplib_c64Px_3_4_0_0`	Included in mmwave sdk installer
C674x DSPLIB	3.4.0.0	`C:\ti\dsplib_c674x_3_4_0_0`	Included in mmwave sdk installer
C674x MATHLIB	3.1.2.1	`C:\ti\mathlib_c674x_3_1_2_1`	Included in mmwave sdk installer
mmwave device support packages	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)

2. Import Lab Project

For the Occupancy plus Vital Signs 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.

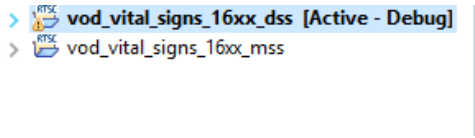
- Start CCS and setup workspace as desired.
- Import the projects below to CCS using either TI Resource Explorer in CCS or CCS Import Projectspecs method:
  - **vod_vital_signs_68xx_dss**
  - **vod_vital_signs_68xx_mss**

Expand for details on importing via TI Resource Explorer in CCS

Expand for details on importing via CCS Import Projectspecs

Successful Import to IDE

After using either method, both project should be visible in **CCS Project Explorer**



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 Automotive Toolbox is not touched.

3. Build the Lab

Build DSS Project

The DSS project must be built before the MSS project.

The DSS project must be built using compiler version 8.3.3. To check the build settings, select **vod_vital_signs_68xx_dss** and right click on the project to select **Show build settings...**. Under the **General** tab, the **Advanced Settings** section has a drop down menu for **Compiler Version**. Ensure that it reads **TI v8.3.3**.

With the **vod_vital_signs_68xx_dss** project selected in **Project Explorer**, right click on the project and select **Rebuild Project**. 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.

> **Successful DSS Project Build**
>
> In the **Project Explorer** panel, navigate to and expand **vod_vital_signs_68xx_dss > Debug** directory. The project has been successfully built if the following files appear in the **Debug** folder:
>
> - vod_vital_signs_68xx_dss.bin
> - vod_vital_signs_68xx_dss.xe674

Build MSS Project

After the DSS project is successfully built, select **vod_vital_signs_68xx_mss** in **Project Explorer**, right click on the project and select **Rebuild Project**.

> **Successful MSS Project Build**
>
> In the **Project Explorer** panel, navigate to and expand **vod_vital_signs_68xx_mss > Debug** directory. The project has been successfully built if the following files appear in the **Debug** folder:
>
> - vod_vital_signs_68xx_mss.bin
> - vod_vital_signs_68xx_mss.xer4f
> - vod_vital_signs_68xx.bin

> **Build Fails with Errors**
>
> If the build fails with errors, please ensure that all the prerequisites are installed as mentioned in the mmWave SDK release notes.

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 **vod_vital_signs_68xx.bin** found at `<PROJECT_WORKSPACE_DIR>\vod_vital_signs_68xx_mss\Debug\vod_vital_signs_68xx.bin`
  - The same procedure for flashing can be use as detailed in the Quickstart Flash the Device section.
- Debug mode: enables connection with CCS while lab is running; useful during development and debugging

Expand for help with Debug mode:

After starting the lab on the AWR6843 using either method, the program will wait until a chirp configuration is sent to it via the "User" COM port.

Chirp Configuration

This section describes the chirp configuration used for the Occupancy and Vital Signs Detection Lab. The chirp was developed to maximize total chirp time for the occupancy detection portion of the processing chain while also providing a high enough frame rate for the vital signs processing chain.

A summary of the chirp configuration is shown below:

Chirp Parameter	Value
Start Frequency (GHz)	77
Idle Time (us)	250

Chirp Parameter	Value
ADC Start Time (us)	10
Ramp End Time (us)	40
Slope (MHz/us)	98
TX Start Time (us)	1
Number of ADC samples	64
ADC Sampling Rate (ksps)	2200

For more information regarding the chirp configuration, please refer to the User Guides for the Driver Vital Signs Demo and the Vehicle Occupancy Detection Demo.

Visualizer Source Files

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

The detection processing chain are implemented both in the firmware and in Matlab, and can be selected independently by configurations in the chirp config and the gui arguments. The visualizer serves to read the UART stream from the device and plots the heatmap, zone decisions, and vital sign information.

Source files are located at `C:<install dir>\occupancy_plus_vital_signs\gui`.

- **vod_vital_signs.m** : the Matlab gui program which reads and parses the UART data for visualization.
- **vod_vital_signs.exe** : the DOS executable version of the gui program.

Chirp Configuration files are located at `C:<install dir>\occupancy_plus_vital_signs\chirp_configs`. In addition to the normal mmWave chirp configuration CLI commands, there are several commands that are specific to the Occupancy plus Vital Signs Detection Demo:

Command	Parameters
guiMonitor	(this command has unique arguments for this demo)
	1 = send **Decision Vector** as a TLV to the UART
	8, 16, 32 = send the **Heatmap** as 8-bit int, 16-bit int, or 32-bit float respectively. 0 = don't send. The GUI automatically adjusts to each format.
	1 = send out **Vital Signs** information
zoneDef	2 = number of zones, followed by the following four values for each zone. Note that you are defining 1 pair of zones.
	range start, specified as the row index into the heatmap
	range length, specified as the number of rows in the heatmap
	azimuth start, specified as the column index in the heatmap
	azimuth length, specified as the number of columns in the heatmap
coeffMatrixRow	zone pair number (0 through 2)
	matrix row number (0 through 3)
	coefficients for the row (currently 6 values)

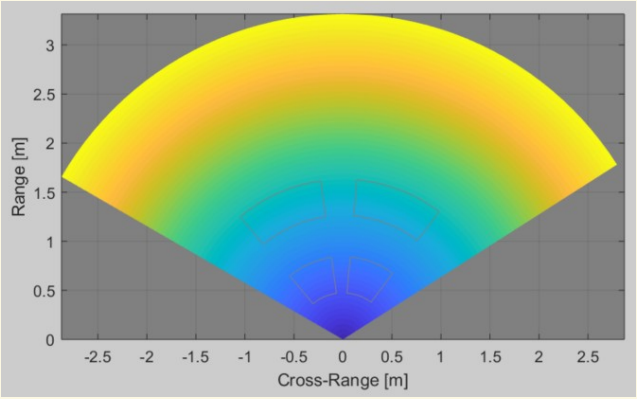
Command	Parameters
meanVector	zone pair number (0 through 2)
	5 values (mean vector used to calculate the feature extract vector)
stdVector	zone pair number (0 through 2)
	5 values (standard deviation vector used to calculate the feature extract vector)
rowNoise*	Generated row noise-floor values for a set of contiguous rows
	first row number (zero based)
	number of rows (number of values to follow)
	noise-floor values
oddemoParms	window length (windowLen = number of frames)
	gamma value: diagonally-loading parameter in MVDR. When the covariance estimate is singular or near-singular, adding a small value along the diagonal avoids numeric stability issues (see VOD Design Doc referenced above).
vitalSignsCfg	Start Range (meters; this parameter is ignored as the range-azimuth pair is taken from the zone definition information)
	End Range (meters; this parameter is ignored as the range-azimuth pair is taken from the zone definition information)
	Breathing Waveform Size (Specifies the number of points within the waveforms)
	Heart rate Waveform Size (Specifies the number of points within the waveforms)
	RX Antenna number to process. Data from a single RX antenna is processed in the current implementation
	Alpha filter value for Breathing waveform energy computation (Alpha filter values for recursive averaging of the waveform energies)
	Alpha filter value for heart beat waveform energy computation (Alpha filter values for recursive averaging of the waveform energies)
	Scale factor for breathing waveform (Scaling factor to convert waveform values in floating points to 32 bit integers required for the FFT)
	Scale factor for heart beat waveform (Scaling factor to convert waveform values in floating points to 32 bit integers required for the FFT)
motionDetection	Enable (1 = enabled the block; 0 = disable the block)
	Data segments Length (L) (Data segment over which the energy is computed)
	Threshold (Energy threshold value. If the energy in the data segment length exceeds this value then the data segment is discarded)
	Gain Control (1 = enable gain control; 0 = disable gain control)

Important Notes on Arc Removal and rowNoise* generation:

A side effect of MVDR is that an "arc" can be generated with some strong targets. This arc is extra energy in all azimuth cells along the affected range row of the heatmap. To combat this, the target code now sports an Arc Removal algorithm. This algorithm depends on knowing what the normal noise-floor value is for each row. The noise-floor is the average energy in the row when no moving targets are present.

The code will automatically generate these noise-floor values for your environment. This is how it works:

1) If rowNoise commands are not read from the config file, the code assumes that they need to be generated. In this case, the GUI will display this static screen for about 10 seconds:



Also, the command window will display the following message:

```
Done
oddemoParms  1  8  0.03  0.50  18
Done
sensorStart
Calculating empty FOV row noise-floor values...
```

When you see these, the code is calculating noise-floor values and will dump the commands to the command window that you can copy paste into your chirp config file, before the sensorStart command.

IT IS CRITICAL that during this time there are **NO** moving objects, people, etc. in the field of view or even nearby.

2) If rowNoise commands are read from the config file, the code will load these values and start normally.

Other notes:

1) Extremely bright (large RCS) targets can cause the 1D FFT to overflow, resulting in an arc that cannot be removed. Targets with this level of RCS will not be found inside a vehicle (an example would be a large corner reflector at close range). These arcs appear as a solid yellow line across the heatmap row.

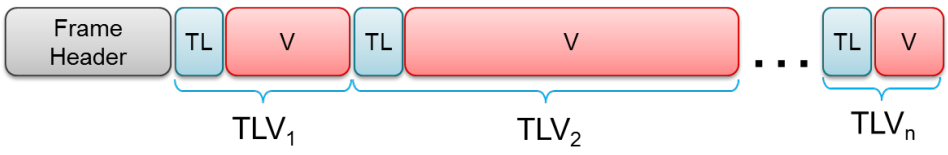
2) noise-floor values will vary slightly from environment to environment, and so should be re-generated as needed. To do this, simply comment out any existing rowNoise commands and restart the demo.

3) If you change Tx Backoff or RxGain values in your profileConfig command, you will need to let the demo regenerate new noise-floor values for you.

Data Packet Format

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 selected via the guiMonitor command. There are 3 possible TLV types for the Occupancy plus Vital Signs demo:

TLV Name	Type	Data Size (bytes)
Azimuth Range Heatmap	8	48 x 64 x 2 or 4 (size of short int or float)
Decision Vector	9	4
Vital Signs Vector	10	5 (elements) x 2 (number of zones) x 4 (size of float)



Frame Header

Size: 32 bytes

Select text

```
frameHeaderStructType = struct(...
    'sync',             {'uint16', 8}, ... % syncPattern in hex is: '02 01 04 03 06 05 08 07'
    'totalPacketLen',   {'uint32', 4}, ... % In bytes, including header and 32 byte padding
    'platform',         {'uint32', 4}, ... % 0xA6843 or 0xA1843
    'frameNumber',      {'uint32', 4}, ... % Starting from 1
    'timeCpuCycles',    {'uint32', 4}, ... % Time in DSP cycles when the message was created
    'numDetectedObj',   {'uint32', 4}, ... % not used in VOD
    'numTLVs' ,         {'uint32', 4}, ... % Number of TLVs in this message
```

Frame header in MATLAB syntax

TLVs

The TLVs can be of type **RANGE_AZIMUT_HEAT_MAP**, **DECISION**, or **VITAL SIGNS VECTOR**. Each TLV consists of a TLV header plus a unique data type.

TLV Header

Size: 8 bytes

Select text

```
% TLV Type: 08 = Heatmap, 10 = Decision Vector, 11 = Vital Signs Vector
tlvHeaderStruct = struct(...
    'type',             {'uint32', 4}, ... % TLV object identifier
    'length',           {'uint32', 4});    % TLV object length, in bytes
```

TLV header in MATLAB syntax

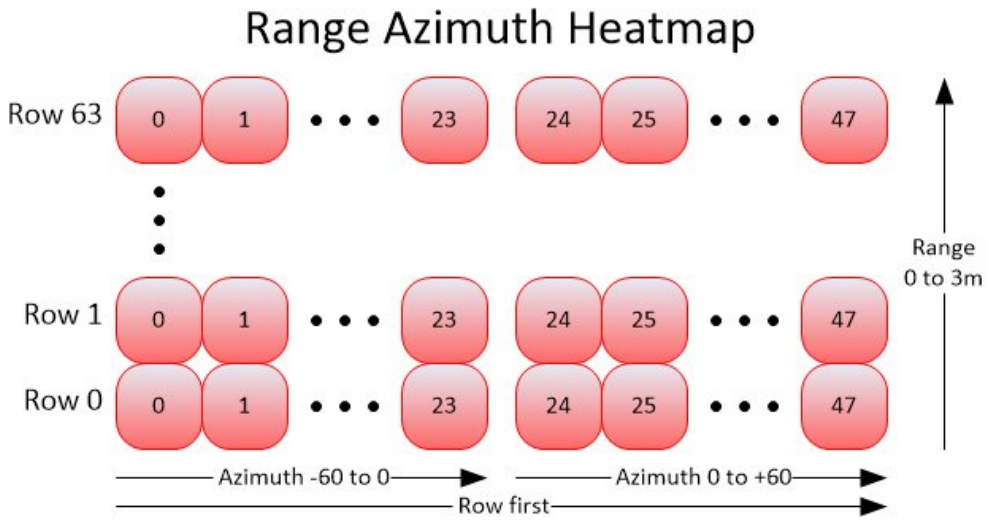
Following the header, is the TLV-type specific payload

Range Azimuth Heatmap TLV

Type: RANGE_AZIMUT_HEAT_MAP

Size: 64 x 48 x sizeof (float or short int, depending on guiMonitor parameter)

> The Occupancy Detection Range Azimuth Heatmap is a 2D array of floats or short ints, currently defined as 64 range rows with 48 azimuth angles per row. The total range is defined at 3 meters, so the range resolution of each row is 3m / 64 = 4.69mm. In terms of azimuth, zero degrees is perpendicular to the antennas, with 60 degrees of view on either side. Negative angles are to the left, with positive angles to the right. With 48 total angles, there are 24 angles per 60 degrees on each side, or 2.55 degrees per angle. The zone definition command (**zoneDef**) is specified as indexes into the heatmap.

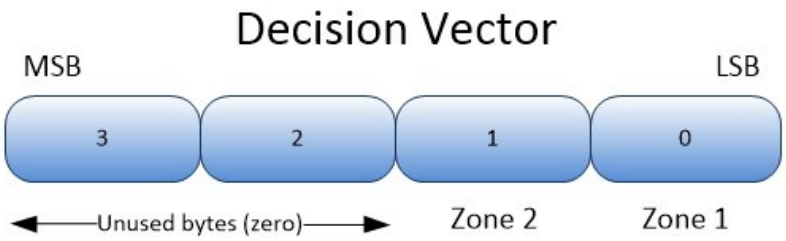


Decision Vector TLV

Type: DECISION

Size: 2, 4 or 6 bytes

The Decision Vector is an array of bytes; one byte per zone. Each byte contains the value 1 if a positive (occupied) decision has been calculated. It is zero otherwise.



Vital Signs Vector TLV

Type: VITAL SIGNS VECTOR

Size: 5 (elements) x 2 (zones) x 4 (size of float)

The Vital Signs ouput vector contains five elements pertaining to each occupancy zone. Three of the values are used to plot waveforms in the MATLAB GUI. The two remaining values are displayed in the MATLAB GUI.

Select text

```
featureVector = struct(...
    'unwrapped_waveform[2]',    {'float', 4}, ... % Unwrapped phase values (plotted)
    'heart_waveform[2]',        {'float', 4}, ... % Heart waveform values (plotted)
    'breathing_waveform[2]',    {'float', 4}, ... % Breathing waveform values (plotted)
    'heart_rate[2]',            {'float', 4}, ... % Heart rate calculated value (displayed)
    'breathing_rate[2]',        {'float', 4}, ... % Breathing rate calculated value (displayed)
```

Vital Signs Structure in MATLAB syntax

Zone and Coefficent Tuning

To maximize the performance of these algorithms, the coefficients need to be tuned for each environment, or scene, in which the demo is used. We provide a tool and the following instructions to tune the demo parameters. The tools have been consolodated in the Vehicle Occupancy Detection (VOD) demo, also found in the Automotive Toolbox. The GUI for this demo no longer supports data capture. Please install and use the VOD demo for documentation and tools for coefficient tuning.

Need More Help?

- Find more details about VOD by referring to the VOD Design document or to the TIREX lab
- Find more details about Vital Signs by referrring to the TIREX lab
- Find answers to common questions on mmWave E2E FAQ
- Search for your issue or post a new question on the mmWave E2E forum