

AN15113

Stereo Vision on the IMX95LPD5EVK-19 Board

Rev. 1.0 — 21 October 2026

Application note

Document information

Information	Content
Keywords	AN15113, Neural Processing Unit (NPU), IMX95LPD5EVK-19, i.MX 95, Stereo vision
Abstract	The objective of this document is to implement the stereo vision system, starting from the hardware level and extending it to the actual image-processing stage.



1 Introduction

The objective of this document is to implement the stereo vision system, starting from the hardware level and extending it to the actual image-processing stage. The platform selected for this project is IMX95LPD5EVK-19, which supports the simultaneous capture of two video streams and accelerated processing through a Neural Processing Unit (NPU). These capabilities make it well suited for experimenting with both traditional computer vision methods and artificial intelligence-based approaches.

This document provides detailed information about a Proof of Concept (PoC) for the IMX95LPD5EVK-19 stereo vision.

2 Hardware and software requirements

[Table 1](#) provides details of the hardware and software used.

Table 1. Hardware and software requirements

Category	Description
Hardware	Development kit: IMX95LPD5EVK-19 Camera kit: 2 x IMX95-OS08A20 Micro SD card: <ul style="list-style-type: none">• SanDisk Ultra 32-GB micro SDHC• Class 10 is used for the current experiment USB: Micro-USB cable for the debug port
Software	Host PC running a recent version of Ubuntu 22.04 Linux 6.12.49 BSP The source code mentioned in this document is available at: <ul style="list-style-type: none">• CPU• NPU

3 Camera positioning

Proper camera positioning is essential for obtaining accurate depth maps in a stereo vision system. The cameras must be aligned in the same direction, with optical axes as parallel as possible, to simplify image rectification and processing.

The distance between the cameras (baseline) directly affects depth estimation accuracy. A small baseline reduces depth precision for distant objects, while a large baseline can make matching nearby objects more difficult. Therefore, the baseline must be selected according to the application requirements.

Both cameras must be rigidly mounted on a common support to prevent relative movement. Also, use identical lenses and camera settings (exposure, focus, gain) to ensure consistent image quality.

Frame synchronization is also important. Images must be captured simultaneously; otherwise, moving objects can appear in different positions, leading to disparity estimation errors.

In summary, careful camera positioning and configuration provide the foundation for reliable stereo vision performance and accurate depth estimation.

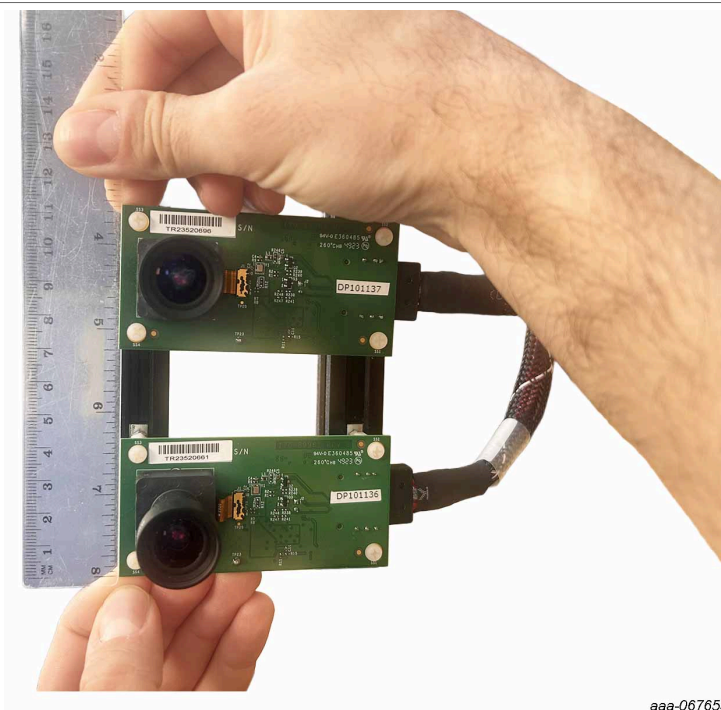


Figure 1. 2 x OS08A20

4 Build the image

For more details on building Yocto, see the *i.MX Yocto Project Users Guide* ([UG10164](#)). The image can also be found on [nxp.com](#).

To download the i.MX Yocto Project community BSP recipe layers, perform the following steps:

Recommended system configuration:

- OS: Ubuntu 22.04 LTS
- RAM: Minimum 16 GB (32 GB recommended)
- Storage: At least 150 GB of free disk space
- CPU: 8 cores (recommended)

Required packages to be installed:

```
$ sudo apt update
$ sudo apt install -y \
gawk wget git diffstat unzip texinfo gcc build-essential chrpath socat \
cpio python3 python3-pip python3-pexpect xz-utils debianutils \
iputils-ping python3-git python3-jinja2 libegl1-mesa libsdl1.2-dev \
pylint xterm rsync curl zst
```

Initialize and synchronize the repo:

```
$ repo init -u https://github.com/nxp-imx/imx-manifest \
-b imx-linux-scarthgap \
-m imx-setup-release.xml

$ repo sync
```

Initialize build:

```
$ DISTRO=fsl-imx-xwayland MACHINE=imx95-evk source imx-setup-release.sh -b  
build-imx95
```

Build image:

```
$ bitbake imx-image-full
```

The image used to demonstrate the concepts presented in this work can be found after the build process is completed at the following location:

```
build-imx95/tmp/deploy/images/imx95-evk/imx-image-full-imx95-evk.rootfs.wic.zst
```

After using any image-writing tool to flash the image onto an SD card, the card must be inserted into the available SD card slot on the EVK.

To access the system command-line interface, the use of a serial terminal application such as PuTTY is recommended. PuTTY communicates with the board through the serial interface, allowing monitoring of the boot process and sending commands to the target system.

Device tree configuration for dual OS08A20 cameras:

Immediately after powering on the board, the boot process is interrupted in U-Boot by pressing any key on the keyboard. To enable the use of two cameras of the same type, the appropriate device tree must be configured using the following commands and settings.

```
u-boot=> edit fdtfile  
edit: imx95-19x19-evk-dual-os08a20-isp-it6263-lvds0.dtb  
u-boot=> saveenv  
Saving Environment to MMC... Writing to MMC(1)... OK
```

5 SGBM running on CPU

1. Capture frames

To perform calibration, synchronized frames must be captured using the `capture_sync_frames.py` script. Capturing more than 120 frames is recommended. However, only the last 100 frames are saved, which automatically discards any unsynchronized frames captured at the beginning of the process.

For this procedure, a physical checkerboard calibration pattern is recommended, consisting of 7×10 squares, with each square having a side length of 2.5 cm. The pattern must be visible to both cameras simultaneously during image capture.

After the capture process is complete, verify the contents of the `captured_saved/camera0` and `captured_saved/camera1` directories to ensure that they contain the same number of images. If any discrepancy is found, for example, if one camera contains an additional frame, remove the extra image to maintain dataset consistency.

The images are acquired from the sensors in 1080p YUV format. To improve overall processing performance, they are converted to 720p RGB using the hardware-accelerated GPU-based GStreamer plugin `imxvideoconvert_oel`.

2. Calibration

Using the images captured with the script described in the previous section, the next calibration step is performed using the `calibration.py` script.

Main functions of the script

a. Calibration pattern detection:

- Automatically detects the checkerboard calibration pattern in images captured by both cameras.

- Precisely identifies the inner corner points of the checkerboard.
- Verifies that the pattern is visible in both camera images simultaneously.
- b. Individual camera calibration:
 - Computes the intrinsic parameters of each camera (focal length and optical center).
 - Determines the lens distortion coefficients.
 - Measures the reprojection error for each camera.
- c. Stereo calibration:
 - Computes the geometric relationship between the two cameras (rotation and translation).
 - Determines the distance between the cameras (baseline).
 - Generates the mapping matrices required for depth estimation from disparity.
- d. Image rectification:
 - Computes the transformations required to align the image pair.
 - Ensures that the corresponding image features lie on the same horizontal lines.
 - Facilitates subsequent depth map computation.
- e. Result storage and validation:
 - Saves all calibration parameters into `.pkl` and `.npz` files.
 - Generates test images for calibration quality verification.
 - Creates visualizations with horizontal alignment lines to validate the rectification results.
- f. Generated outputs:
 - `stereo_calibration.pkl`: complete stereo calibration parameters.
 - `rectified_test.jpg`: test image containing alignment lines.
 - `corners_visualization/`: directory containing checkerboard corner detection results for each image pair.

3. Depth map computation

To generate the depth map, a Python script named `depth_map.py` was developed using the calibration data obtained in the previous step.

Main functionality:

- Loads calibration parameters: reads the stereo camera calibration data from the `stereo_calibration.pkl` file.
- Rectifies images: corrects lens distortion and aligns stereo image pairs to facilitate disparity computation.
- Computes the disparity map: identifies the positional differences of corresponding objects in the two images using the StereoSGBM algorithm.
- Generates the depth map: converts disparity values into real-world distances (depth) using the Q matrix obtained during calibration.
- Processes and filters the results: applies filtering techniques to reduce noise and improve output quality, including median filtering and the WLS (Weighted Least Squares) filter.
- Saves the generated results in multiple formats:
 - Disparity and depth maps (`.npy` format)
 - Colorized depth visualizations (`.png` format)
 - Side-by-side comparison images

The script is capable of automatically processing all stereo image pairs within a directory and generates useful visualizations for analyzing and validating the results.

4. Tune results

The initial results obtained in the previous step can be further improved using the `tune_disparity.py` script in an interactive tuning mode.

Main functionality:

- Loads a stereo image pair (left-right) captured from a calibrated stereo camera setup.
- Displays the original image alongside the generated disparity map (depth map).
- Allows parameter adjustment through interactive sliders (trackbars) with immediate visualization of the results.

Adjustable parameters:

- Num disp: Number of disparities (controls the detectable depth range).
- Block size: Matching block size (larger values produce smoother results but can reduce detail).
- Uniqueness: Uniqueness ratio used to reduce false matches.
- Speckle: Noise filtering parameters (window size/disparity range).
- P1/P2: Smoothness penalty parameters.
- Median filter: Post-processing filter for noise reduction.
- WLS filter: Advanced edge-preserving filtering (requires opencv-contrib-python).

Controls:

- Adjust parameters in real time using the trackbars.
- Press “s” to save the current parameter configuration to a file.
- Press “r” to restore the default settings.
- Press “q” to exit the application.

After the tuning process is completed, the optimized parameters can be saved to the stereo_params.npy file for later use during depth map generation.

6 Results obtained

1. Synchronization

To verify that the two cameras are synchronized and capture frames at the same point in time, place a high-resolution timer or timestamp display within the field of view of both cameras. The timer must update frequently enough to indicate the capture instant clearly.

Capture images simultaneously from both cameras and compare the displayed timestamp in the acquired frames. If both images show the same timestamp (or the same timer value), the cameras are considered synchronized and are capturing frames at the same moment in time. Any difference in the displayed timestamp indicates a synchronization offset between the cameras.

2. Rectification

[Figure 2](#) shows the rectified images from the two cameras. As indicated by the green horizontal lines, corresponding pixels representing the same objects in both camera images are aligned along the same image rows. This alignment confirms that the rectification process has been successfully applied and facilitates accurate disparity and depth estimation.

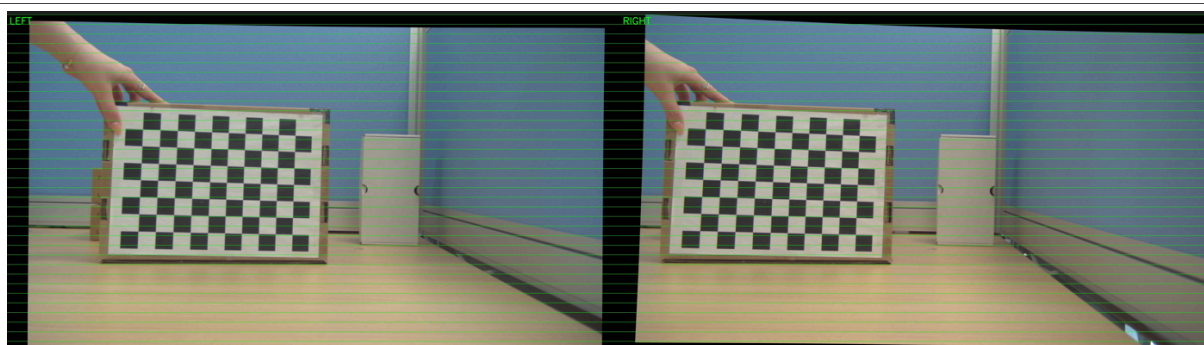


Figure 2. Rectified images

3. Depth map

- a. Color without WLS filter applied.

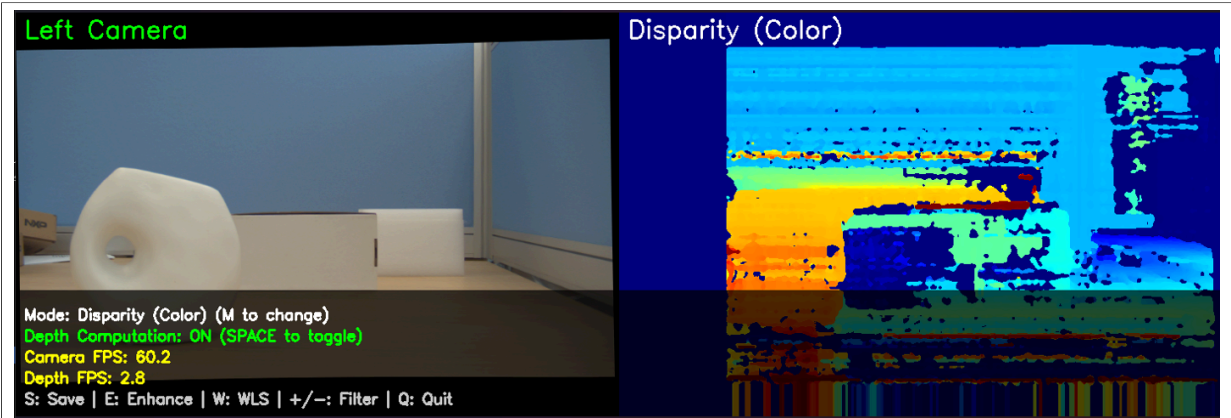


Figure 3. Color without WLS filter

- b. Color with WLS filter applied
- In [Figure 4](#), the results show that objects closer to the camera lenses are represented in red. However, the objects located farther away gradually transition to blue shades, which reflects the increase in distance from the stereo camera system.



Figure 4. Color with WLS filter

- c. Black and white with WLS filter
- In [Figure 5](#), the results show that objects closer to the camera lenses are represented in white. However, the objects located farther away gradually transition to darker shades, approaching black, which indicates an increase in distance from the stereo camera system.



Figure 5. Black and white with WLS filter

d. Conclusions for SGBM on CPU

Visually the most compelling results are those obtained using the WLS filter for post-processing enhancement. The quality of these results can be further improved through professional camera calibration performed with dedicated calibration equipment.

However, the performance of this implementation is relatively low, achieving only 1.6 disparity maps per second. This limitation arises because all processing stages are executed on the CPU. Therefore, to achieve higher performance and better real-time capabilities, the following sections present a hardware-accelerated approach that uses the NPU available on the i.MX 95 platform.

7 Depth map generation using NPU

This section describes the development of a depth map generation solution accelerated by the NPU of the i.MX 95 platform. A lightweight Convolutional Neural Network (CNN) is trained to estimate disparity maps from stereo image pairs captured by synchronized cameras.

The training dataset is generated using disparity maps produced by the Semi-Global Block Matching (SGBM) algorithm presented in [Section 5](#). After training, the model is converted to an optimized TensorFlow Lite (TFLite) format and quantized for NPU execution. Finally, the model is evaluated and deployed for real-time depth estimation on the IMX95LPD5EVK-19 platform.

7.1 Create the data set for training

The dataset used to train the model is generated using the SGBM algorithm presented in the [Section 5](#). The dataset is responsible for the initial generation of the disparity maps. This process is based on images captured from the two synchronized cameras, which are configured on the IMX95LPD5EVK-19 platform.

The training data is organized within the `scenes` directory, where each scene is stored in a separate subdirectory. Each subdirectory contains the images corresponding to a single capture session, as illustrated in [Figure 6](#).

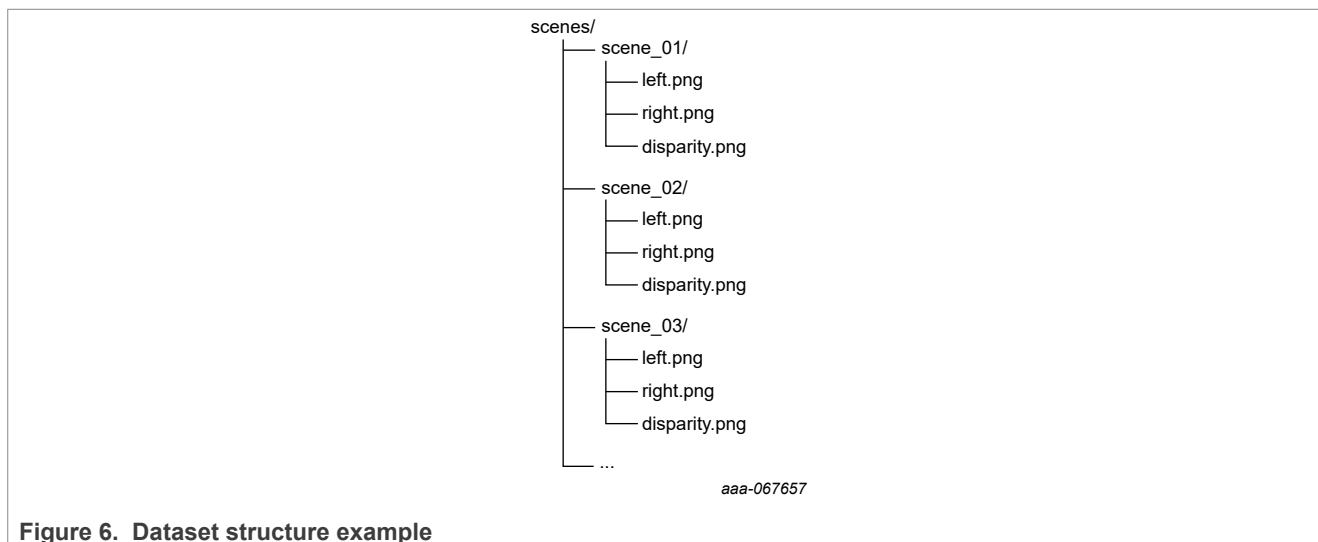


Figure 6. Dataset structure example

For each scene, the following files are available:

- `left.png` - image captured by the left camera.
- `right.png` - image captured by the right camera.
- `disparity.png` - disparity map used as the ground truth, generated using the SGBM algorithm.

This clear and modular organization facilitates efficient data management and ensures the consistency required for the model training process.

7.1.1 Train the model

After preparing the dataset, the next step is to train the stereo matching model using the images and disparity maps generated in the previous stage. Before training can begin, the required dependencies for running the training script must be installed:

```
$ pip install -r requirements.txt
```

The training process can be started by running the dedicated training script:

```
$ python train_model.py
```

The model is a CNN optimized for the i.MX 95NPU, designed to estimate disparity maps from stereo image pairs.

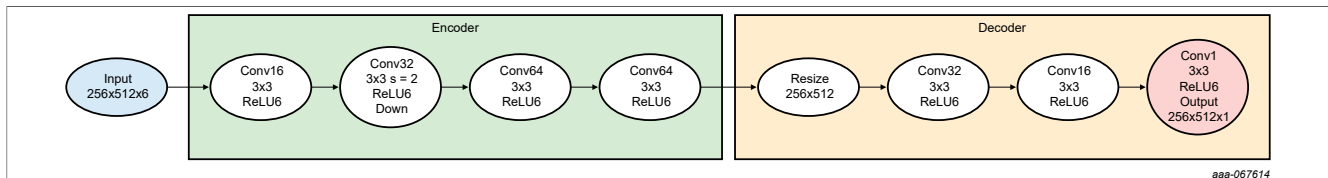


Figure 7. AI model architecture

The AI model architecture is as follows.

- Input:
 - Two images: left_image and right_image ($256 \times 512 \times 3$ RGB)
 - Concatenation along the channel dimension $\rightarrow 256 \times 512 \times 6$
- Encoder (Feature Extraction):
 - Block 1: Conv2D(16, 3×3) $\rightarrow$ ReLU6
 - Block 2: Conv2D(32, 3×3 , stride=2) $\rightarrow$ ReLU6 (downsampling)
 - Block 3: Conv2D(64, 3×3) $\rightarrow$ ReLU6
 - Block 4: Conv2D(64, 3×3) $\rightarrow$ ReLU6
- Decoder (reconstruction):
 - ResizeBilinear $\rightarrow 256 \times 512$ (upsampling)
 - Conv2D(32, 3×3) $\rightarrow$ ReLU6 (refinement)
 - Conv2D(16, 3×3) $\rightarrow$ ReLU6 (refinement)
 - Conv2D(1, 3×3) $\rightarrow$ ReLU6 (output: disparity map)
- Output:
 - Disparity map: $256 \times 512 \times 1$ (normalized values in the range $[0, 6]$)
- Key Features:
 - 85 % NPU compatible - no Lambda layers are used.
 - Full INT8 quantization for hardware acceleration.
 - Lightweight architecture (~a few hundred kB).

Table 2. Files generated by the training script

File	Format	Purpose	Usage	Details
best_model_npu.h5	Keras HDF5	Best checkpoint during training	Further training or fine-tuning	Lowest validation loss

Table 2. Files generated by the training script...continued

File	Format	Purpose	Usage	Details
stereo_model_npu.h5	Keras HDF5	Final model after all epochs	Python inference	Standard Keras format
stereo_model_npu.keras	Native Keras format	Modern model storage format (TensorFlow 2.x)	Model storage	Recommended
stereo_model_npu.tflite	TensorFlow Lite INT8	Embedded deployment model	Deployment on the i.MX 95 NPU	uint8 inputs/outputs
model_params_npu.npz	NumPy archive	Denormalization parameters	Output conversion	max_disparity, output_scale

7.1.2 Convert the model

After the model has been trained in TensorFlow/Keras, it cannot be used directly on embedded hardware such as i.MX 95. Therefore, a multi-stage conversion process is required to make the model compatible with and efficient for execution on the NPU.

1. Conversion to TFLite

Once training is completed, the model is automatically converted to the TFLite format. This format is designed for:

- Embedded devices (for example, development boards and edge systems)
- Reduced memory consumption
- Fast inference execution

During this stage:

- The model is structurally optimized wherever possible.
- INT8 quantization is applied, meaning that numerical values are converted from 32-bit floating-point format to 8-bit integers.
- Quantization reduces the model size and improves execution performance on hardware accelerators.

The result is a `.tflite` file that can run on the CPU while also being prepared for hardware acceleration.

2. Conversion for the NPU (Neutron NPU)

Although the TFLite model is optimized, it is not yet fully compatible with the NPU available on the i.MX 95. To enable execution on the hardware accelerator, an additional conversion step is required using the NXP eIQ converter.

During this process:

- The model is analyzed layer by layer.
- Each operation (operator) is verified for NPU compatibility.

The result of this conversion is a model optimized for hybrid execution:

- Approximately 85 % of the operations are executed on the NPU, providing significant hardware acceleration.
- The remaining ~15 % are executed on the CPU, as the NPU does not support certain operations directly.

```
Overall statistics for graph "main":
Operators:
  Number of operators after import      = 12
  Number of operators after optimize    = 20
  Number of operators after extract     = 4
    Number of Neutron graphs           = 1
  Number of operators total             = 20
  Number of operators converted         = 17
  Number of operators NOT converted     = 3
  Operator conversion ratio             = 17 / 20 = 0.85
  Operators converted                   = 3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,
Memory:
  Total data      = 12,910,592 (bytes) (Inputs + Outputs + Intermediate Variable Tensors)
  Total weights   = 314,992 (bytes) (Weights)
  Total size      = 13,225,584 (bytes) (All)
Latency:
  Cycle estimation = 9,857,380 (cycles) (NPU only)
  Latency estimation = 9.857 (ms) (@ 1000.000 MHz) (NPU only)
```

Figure 8. Model conversion results

Model execution is performed in a hybrid manner that uses both the NPU and the CPU. Computationally intensive operations, such as convolutions and activation functions, are executed on the NPU to take advantage of hardware acceleration. However, the CPU handles the simpler operations or the operations not supported by the NPU.

This approach provides an optimal balance between performance and flexibility, using the efficiency of dedicated hardware acceleration while maintaining the ability to execute all required operations. As a result, significant performance improvements can be achieved without sacrificing compatibility with unsupported operators.

7.1.3 Test the model

When the conversion process is completed, the model can be tested using one or more image pairs to verify the correctness of the generated results.

To perform the test, run the following dedicated script:

```
$ python test_model.py
```

This script provides the following functionality:

- Runs inference on test images (stereo pairs).
- Computes relevant performance metrics, such as Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and Bad3.
- Visualizes the results by comparing the model predictions with the corresponding ground truth disparity maps.

This enables evaluation of the accuracy and behavior of both models under real-world operating conditions.

7.1.4 Real-time usage

To enable real-time deployment of the model on the IMX95LPD5EVK-19 platform, a dedicated script is provided. This script performs continuous stereo image processing using the NPU-optimized model.

The application can be launched with the following command:

```
$ python NPU_Stereo_Vision.py
```

This script performs real-time inference, that is, the model processes incoming image streams (for example, from stereo cameras) when they become available, with minimal latency.

This approach enables the integration of the model into practical applications, such as robotics vision systems, automotive applications, and embedded systems, where continuous and high-speed processing of visual data is required.

7.1.5 Results

Figure 9 shows the results obtained from the model trained using only 13 training scenarios.

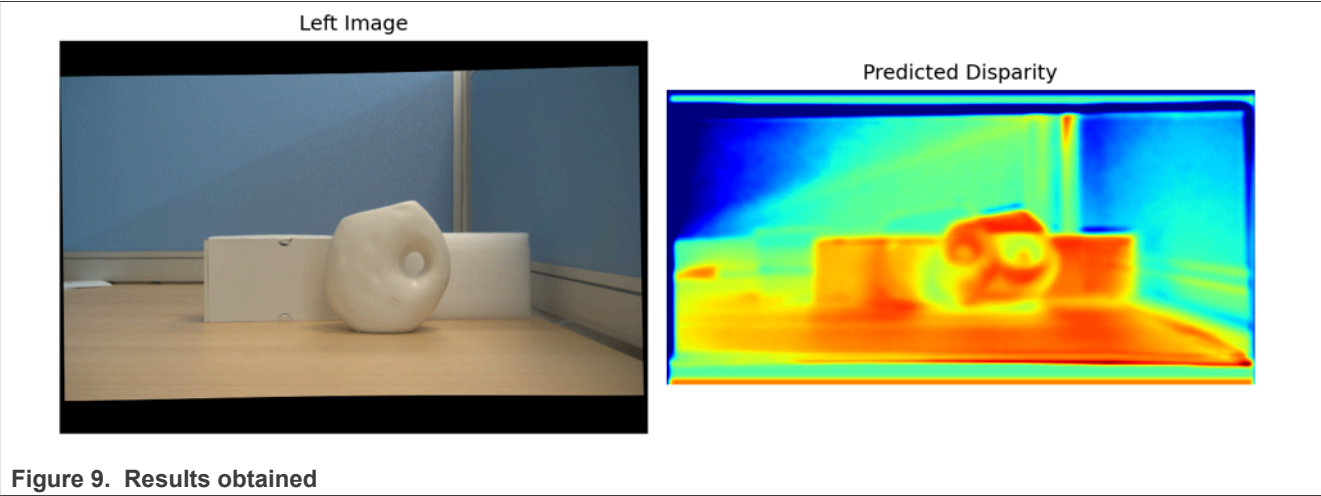
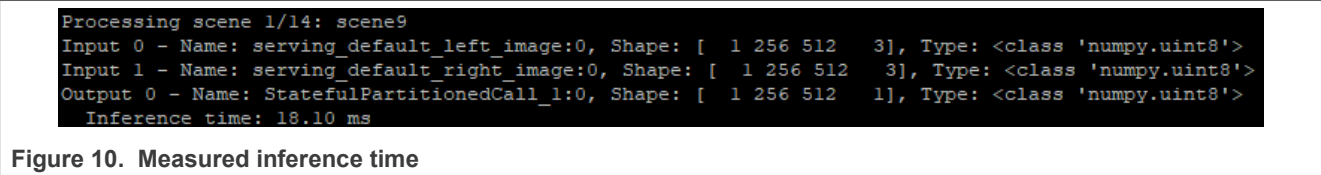


Figure 10 shows the inference time measured on the test dataset.



8 Conclusion

This application note presents a PoC for implementing a stereo vision system on the IMX95LPD5EVK-19 platform. Two depth estimation approaches are evaluated: a classical SGBM-based method running on the CPU and an AI-based model accelerated by the NPU.

The SGBM approach delivers good visual results but is limited by CPU performance. The AI-based solution demonstrates the benefits of hardware acceleration, enables faster inference, and provides a path toward real-time applications. As the model is trained on a relatively small data set, further improvements are expected with larger and more diverse training data.

Overall, the PoC demonstrates that the IMX95LPD5EVK-19 board can efficiently support both traditional computer vision techniques and AI-based stereo vision solutions. This capability makes it a suitable platform for future embedded depth-sensing applications.

9 Acronyms

Table 3 lists the acronyms used in this document.

Table 3. Acronyms

Acronym	Description
BSP	Board Support Package
CNN	Convolutional Neural Network
CPU	Central Processing Unit
EVK	Evaluation Kit
GPU	Graphics Processing Unit
MAE	Mean Absolute Error
NPU	Neural Processing Unit
PoC	Proof of Concept
RMSE	Root Mean Square Error
SGBM	Semi-Global Block Matching
TFLite	TensorFlow Lite

10 Note About the Source Code in the Document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2026 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials must be provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

11 Revision history

[Table 4](#) summarizes the revisions to this document.

Table 4. Revision history

Document ID	Release date	Description
AN15113 v.1.0	21 October 2026	Initial public release

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at <https://www.nxp.com/profile/terms>, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Suitability for use in automotive applications — This NXP product has been qualified for use in automotive applications. If this product is used by customer in the development of, or for incorporation into, products or services (a) used in safety critical applications or (b) in which failure could lead to death, personal injury, or severe physical or environmental damage (such products and services hereinafter referred to as "Critical Applications"), then customer makes the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, safety, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. As such, customer assumes all risk related to use of any products in Critical Applications and NXP and its suppliers shall not be liable for any such use by customer. Accordingly, customer will indemnify and hold NXP harmless from any claims, liabilities, damages and associated costs and expenses (including attorneys' fees) that NXP may incur related to customer's incorporation of any product in a Critical Application.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

HTML publications — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

eIQ — is a trademark of NXP B.V.

TensorFlow, the TensorFlow logo and any related marks — are trademarks of Google Inc.

Contents

1 Introduction2

2 Hardware and software requirements2

3 Camera positioning2

4 Build the image3

5 SGBM running on CPU4

6 Results obtained6

7 Depth map generation using NPU8

7.1 Create the data set for training8

7.1.1 Train the model9

7.1.2 Convert the model10

7.1.3 Test the model11

7.1.4 Real-time usage11

7.1.5 Results12

8 Conclusion12

9 Acronyms12

10 Note About the Source Code in the Document13

11 Revision history13

Legal information14

Please be aware that important notices concerning this document and the product(s) described herein, have been included in section 'Legal information'.