Real-Time Spatial Geometry Reconstruction via Monocular Depth Estimation

Edge-Optimized Deployment on Intel Integrated Architecture using OpenVINO and MiDaS

Author:

Vasco Sá

GitHub | LinkedIn

6 February 2026

Abstract

This report presents a real-time pipeline for Monocular Depth Estimation and pseudo-LiDAR simulation using Python and the OpenVINO toolkit. By leveraging the MiDaS v2.1 Small model optimized for Intel® Integrated Graphics, the system transforms a standard 2D webcam feed into a calibrated spatial depth map. The analysis details the inference architecture, focusing on the integration of heterogeneous computing to achieve real-time performance. A key contribution is the derivation of an inverse power-law calibration model to convert relative disparity into absolute metric distance, achieving functional accuracy within an effective operating range of 0.5m to 2.5m. The final application demonstrates robust spatial awareness on consumer-grade hardware, eliminating the need for specialized active depth sensors.

Keywords

Monocular Depth Estimation, OpenVINO, MiDaS, Computer Vision, LiDAR Simulation, Real-Time Inference, OpenCV, Edge Computing.

Acknowledgements

I would like to acknowledge the team behind MiDaS (Intel Intelligent Systems Lab) for providing the state-of-the-art robust monocular depth estimation models. Special thanks to the OpenVINO Toolkit developers for enabling efficient deep learning inference on edge devices, making this real-time application possible.

Table of Contents

Nomenclature **3**
1. Introduction **4**
1.1 Context and Motivation 4
1.2 Problem Statement 4
1.3 Objectives 5
1.4 Project Scope 5
2. Theoretical Background **6**
2.1 Monocular Depth Estimation (MDE) and the Inverse Problem 6
2.1.1 The Ill-Posed Nature of Monocular Vision 6
2.1.2 Monocular Depth Cues 7
2.1.3 MiDaS Architecture and Zero-Shot Generalization 8
2.2 Disparity-to-Depth Mapping 10
2.3 OpenVINO and Heterogeneous Computing 11
2.4 Signal Processing and Temporal Consistency 12
3. Methodology **14**
3.1 Experimental Setup 15
3.2 Data Acquisition and Calibration Procedure 15
3.3 Mathematical Fitting and Parameter Estimation 16
3.4 Implementation and HUD Visualization 18
3.4.1 Diagnostic Tool (depth_map.py) 18
3.4.2 LiDAR Simulation Application (lidar_scanner.py) 19
4. Results and Analysis **20**
4.1 Hardware Performance Benchmarking 20
4.2 Distance Measurement Accuracy 21
5. Discussion **23**
5.1 The Texture Dependency Problem 23
5.2 The Geometry of Asymptotic Error 23
5.3 Latency vs. Smoothing Trade-off 24
6. Conclusion and Future Work **24**
References **25**

Nomenclature

Acronym	Definition
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DPT	Dense Prediction Transformer
EMA	Exponential Moving Average
FPS	Frames Per Second
FSM	Finite State Machine
GPU	Graphics Processing Unit
HUD	Head-Up Display
iGPU	Integrated GPU
IR (hardware)	Infrared
IR (OpenVINO)	Intermediate Representation
KITTI	Karlsruhe Institute of Technology and Toyota Technological Institute
LiDAR	Light Detection and Ranging
MAPE	Mean Absolute Percentage Error
MDE	Monocular Depth Estimation
MiDaS	Towards Robust Monocular Depth Estimation
NYU	New York University
OpenCV	Open Source Computer Vision Library
OpenVINO	Open Visual Inference and Neural Network Optimization
RGB	Red Green Blue

1. Introduction

1.1 Context and Motivation

Accurate depth perception is a fundamental requirement for modern computer vision systems, enabling applications ranging from autonomous driving and robotics to augmented reality. Traditionally, capturing high-fidelity 3D spatial data relied on specialized hardware such as LiDAR (Light Detection and Ranging) sensors or stereo camera rigs. While effective, these solutions significantly increase the cost, power consumption, and physical footprint of the device.

Monocular Depth Estimation (MDE) presents a compelling alternative by attempting to predict depth from a single RGB image. With the advent of Convolutional Neural Networks (CNNs), it has become possible to learn depth cues (such as texture gradients, occlusion, and perspective) directly from data. However, deploying these complex neural networks in real-time on resource-constrained devices remains a significant engineering challenge.

1.2 Problem Statement

State-of-the-art Monocular Depth Estimation models, such as MiDaS or DPT (Dense Prediction Transformer), are typically designed for high-end server environments with dedicated GPUs (e.g., NVIDIA RTX series). Running these models on consumer-grade hardware, particularly laptops with Integrated Graphics (iGPU) or embedded edge devices, often results in unacceptably low frame rates (FPS) and high latency.

Furthermore, most MDE models output "relative depth" (disparity maps) rather than absolute metric distance. This means the model knows that Object A is closer than Object B, but not that Object A is exactly 1.5 meters away. For practical applications like collision avoidance or measurement, this relative output is insufficient. There is a need for a pipeline that not only optimizes inference speed for edge devices but also bridges the gap between relative prediction and absolute metric accuracy.

1.3 Objectives

The primary objective of this project is to design and implement an end-to-end real-time Monocular Depth Estimation pipeline that runs efficiently on Intel Integrated Architecture.

Specific goals include:

1. **Edge Optimization:** Utilize the OpenVINO toolkit to compile and optimize the MiDaS v2.1 Small model for Intel UHD Graphics, maximizing throughput.
2. **LiDAR:** Develop a virtual ray-casting algorithm that simulates a laser rangefinder, allowing the system to "lock" onto objects and measure their distance.
3. **Metric Calibration:** Derive and validate a non-linear mathematical model to convert the neural network's relative inverse-depth output into real-world metric units (meters).
4. **Interactive Visualization:** Implement temporal smoothing algorithms, such as Exponential Moving Average (EMA), to mitigate frame-to-frame flicker and edge noise.

1.4 Project Scope

This report documents the software engineering and computer vision methodologies used to build the "Depth Scanner" tool. The system is designed to operate with a standard generic webcam (USB or integrated) as the sole input sensor.

The scope is focused on indoor environments with an effective operating range of 0.5 meters to 2.5 meters. The project prioritizes inference speed (FPS) and interactivity over pixel-perfect high-resolution reconstruction. The final deliverable is a Python-based application featuring two modes: a colorized depth heatmap for visualization and a numerical LiDAR scanner for measurement.

2. Theoretical Background

2.1 Monocular Depth Estimation (MDE) and the Inverse Problem

2.1.1 The Ill-Posed Nature of Monocular Vision

In computer vision, monocular depth estimation is fundamentally an ill-posed problem. When a 3D scene is projected onto a 2D image plane (u,v), the depth dimension (z) is lost. Mathematically, for any given pixel, there are infinite possible 3D points in space that could have generated that specific projection.

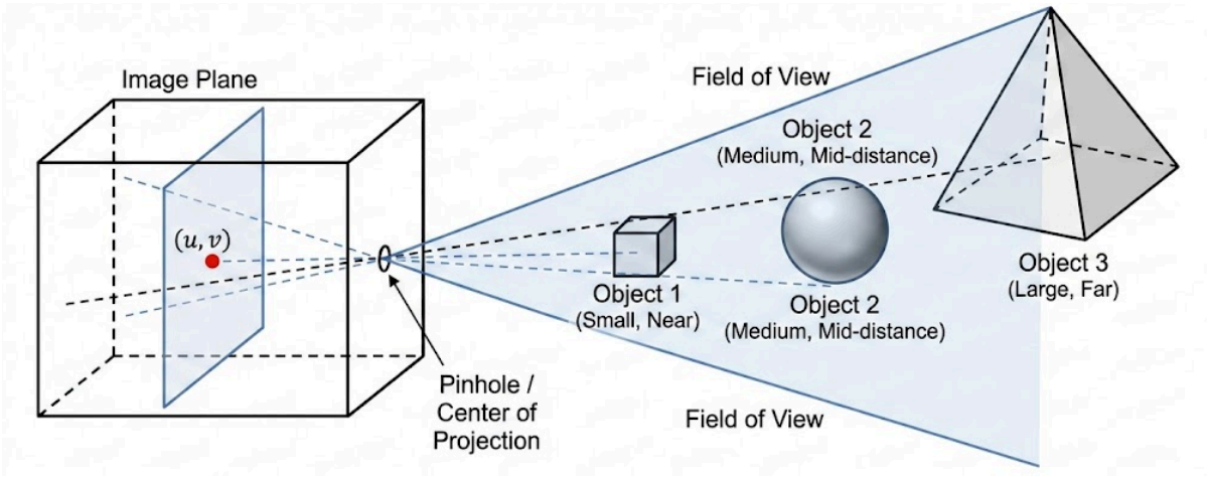

Figure 1 - *The Ill-Posed Nature of Monocular Vision. The diagram illustrates scale ambiguity in a pinhole camera model. Because a single 2D projection lacks depth information, infinite 3D points along a projection ray map to the same 2D coordinate (u,v), making it impossible to distinguish between object size and distance without additional semantic priors.*

- **Traditional Approach:** Relies on Stereopsis (using two cameras) to calculate depth via triangulation and disparity matching.
- **The AI Approach:** Humans can perceive depth with one eye closed by using "monocular cues." MDE models like MiDaS use Deep Learning to mimic this by training on millions of images until the network learns to recognize these semantic patterns as "depth priors."

2.1.2 Monocular Depth Cues

In the absence of binocular disparity, which allows for direct triangulation of depth via stereopsis, the reconstruction of 3D geometry from a single 2D image relies on a set of visual features known as monocular depth cues. These are geometric and photometric patterns that the human visual system (and by extension, the neural network) uses to infer spatial relationships from a flat projection. The MiDaS model's encoder backbone is effectively trained to recognize and weigh these cues to generate a probabilistic disparity map.

The most dominant of these cues in man-made environments is Linear Perspective, where parallel lines appear to converge at a vanishing point as they recede into the distance. This geometric property allows the model to infer the depth of hallways, tables, and room corners with high confidence. Closely related to this is the cue of Relative Size, which relies on the semantic understanding that objects subtending a smaller visual angle on the image sensor are located further away. For the AI to utilize this, it implicitly learns semantic priors about the typical dimensions of common objects, such as chairs or humans, allowing it to scale depth appropriately.

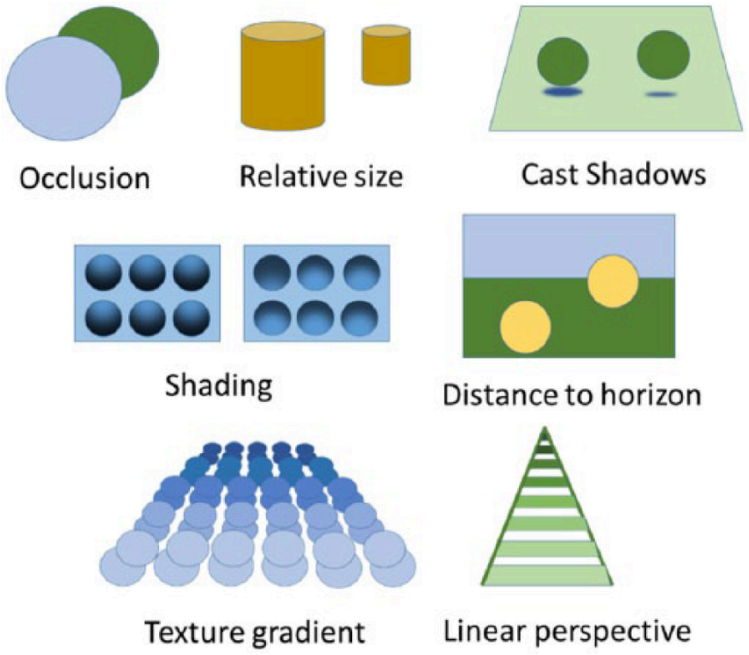

Figure 2 - *Primary Monocular Depth Cues. (Source: Bogdanova et al., 2016). The illustration demonstrates common geometric and photometric patterns (such as linear perspective and relative size) that are utilized by the MiDaS model to infer spatial relationships from a flat 2D projection.*

2.1.3 MiDaS Architecture and Zero-Shot Generalization

For this project, the MiDaS v2.1 Small model was selected as the core inference engine. Unlike traditional depth estimation models, which are often trained on a single dataset (such as KITTI for autonomous driving or NYU Depth for indoor scenes), MiDaS is designed for robust Zero-Shot Cross-Dataset Transfer. This means the model can generate accurate disparity maps for arbitrary environments (such as the specific room used in this project) without having ever been trained on data from that location.

The architecture achieves this generalization through a multi-objective learning strategy. The model was trained simultaneously on over 10 distinct datasets, combining 3D movies, stereo images, and laser scan data. This diverse training regimen forces the network to learn a universal representation of depth rather than overfitting to specific textures or camera intrinsics found in a single dataset.

Structurally, the model utilizes a fully convolutional encoder-decoder architecture. The Encoder (based on EfficientNet-Lite in the "Small" version) extracts high-level semantic features from the input RGB image, identifying objects and edges. The Decoder then progressively upsamples these features to construct a dense, single-channel disparity map.

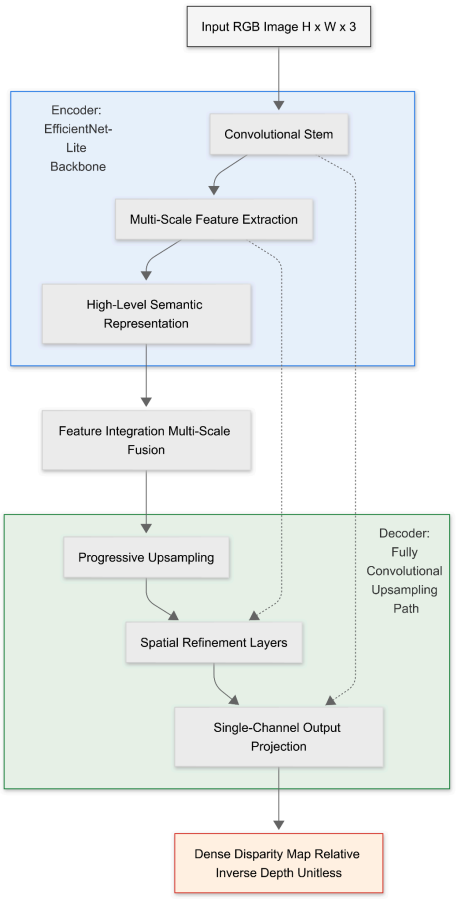

Figure 3 - *MiDaS v2.1 Small Architectural Overview. The model employs a fully convolutional encoder-decoder topology. The EfficientNet-Lite backbone (Encoder) extracts multi-scale semantic features, which are subsequently integrated and upsampled by the Decoder to produce a dense, unitless disparity map suitable for real-time inference.*

It is critical to note that the output of this architecture is relative inverse depth, also known as disparity. The model does not predict absolute metric distance (e.g., "1.5 meters"); instead, it predicts the pixel shift associated with depth. This architectural characteristic necessitates the post-inference calibration step discussed in the methodology section, as the raw output values are unitless and relative to the frame's specific dynamic range.

2.2 Disparity-to-Depth Mapping

A critical theoretical distinction in this project is the difference between Absolute Depth (Z) and Disparity (d). The MiDaS model does not regress a metric distance map directly; rather, it outputs Relative Inverse Depth. In computer vision terms, this output behaves like disparity in a stereo vision system, where pixel intensity is inversely proportional to physical distance.

Mathematically, the relationship between depth Z and disparity d in a standard pinhole camera model is defined by the inverse law:

$$Z = \frac{f \cdot B}{d}$$

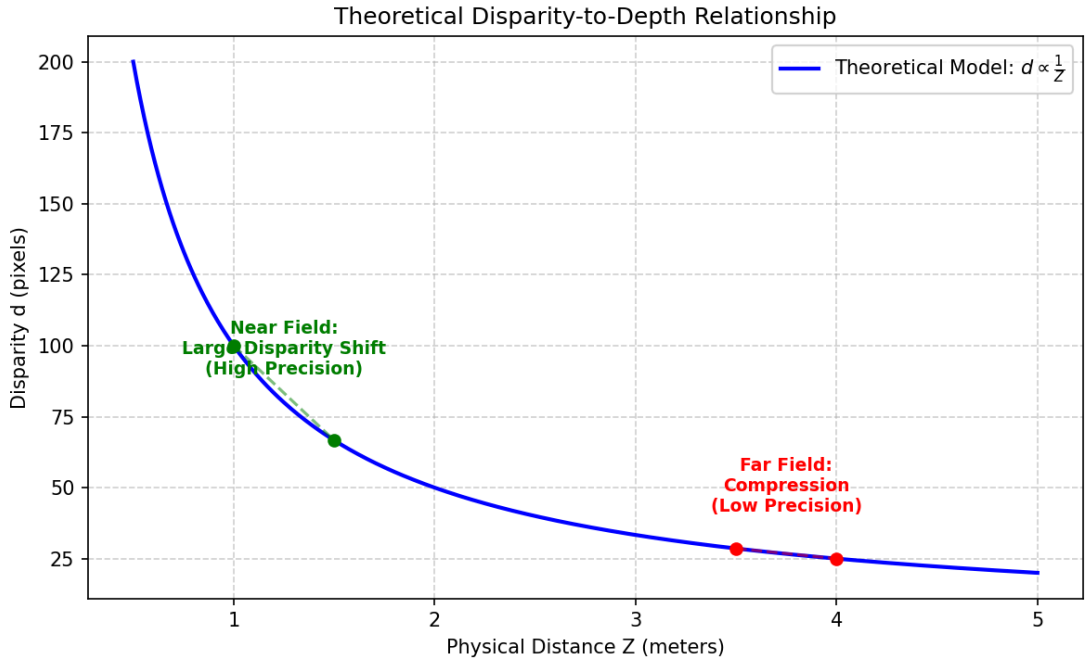

Figure 4 - *Inverse Depth Function Graph. The plot illustrates the hyperbolic relationship between physical distance (Z) and disparity (d). The steep gradient in the near field indicates high sensitivity, while the asymptotic flattening in the far field results in data compression and loss of precision.*

Where f is the focal length and B is the baseline distance between cameras. In a monocular setup, the values of f and B are unknown or "virtual," hallucinated by the neural network based on its training data. Consequently, the raw output from the AI is scale-invariant. The model understands the geometry of the scene (i.e., that the table is closer than the wall) but suffers from Scale Ambiguity - it cannot inherently distinguish between a miniature room seen up close and a large hall seen from a distance.

To recover metric data suitable for a LiDAR simulation, this inverse relationship must be inverted and scaled. The project accounts for this by modeling the transformation as a hyperbolic function. However, due to non-linearities introduced by the neural network's activation functions and the distortion of the wide-angle webcam lens, a standard inverse ($Z \propto 1/d$) is often insufficient.

This necessitates a Power-Law Calibration model, which introduces a non-linear exponent to linearize the depth estimation at close range. The transformation from the unitless AI output (x) to metric distance (D) is therefore modeled as:

$$D = \frac{K}{(x - x_{offset})^P}$$

Here, K represents the virtual scale factor (recovering the missing $f \cdot B$ term), xoffset accounts for the model's internal "horizon" or zero-point, and P corrects for near-field compression. This formula enables the conversion of abstract neural features into a calibrated Euclidean coordinate system.

2.3 OpenVINO and Heterogeneous Computing

To transition from a theoretical model to a real-time application, the project leverages Hardware-Aware AI principles. Standard deep learning frameworks (like PyTorch) are designed for flexibility and training, often carrying significant overhead during inference. To mitigate this, the MiDaS model was optimized using the Intel OpenVINO (Open Visual Inference and Neural Network Optimization) toolkit.

The optimization process begins with the Model Optimizer, which converts the trained neural network into an Intermediate Representation (IR). This IR consists of two files: an `.xml` file describing the network topology and a `.bin` file containing the weights and biases. During this conversion, OpenVINO performs graph-level optimizations, such as Linear Operation Fusing (merging convolution, batch normalization, and activation layers into a single kernel) to reduce memory access latency.

$$T_{total} = T_{preprocess} + T_{inference} + T_{postprocess}$$

By utilizing the GPU device plugin, the system executes the computationally intensive matrix multiplications (Tinference) on the iGPU's execution units. This architecture frees the CPU to handle the sequential logic of video capture, signal processing (Tpostprocess), and HUD rendering, thereby maximizing the total system throughput and ensuring a stable frame rate.

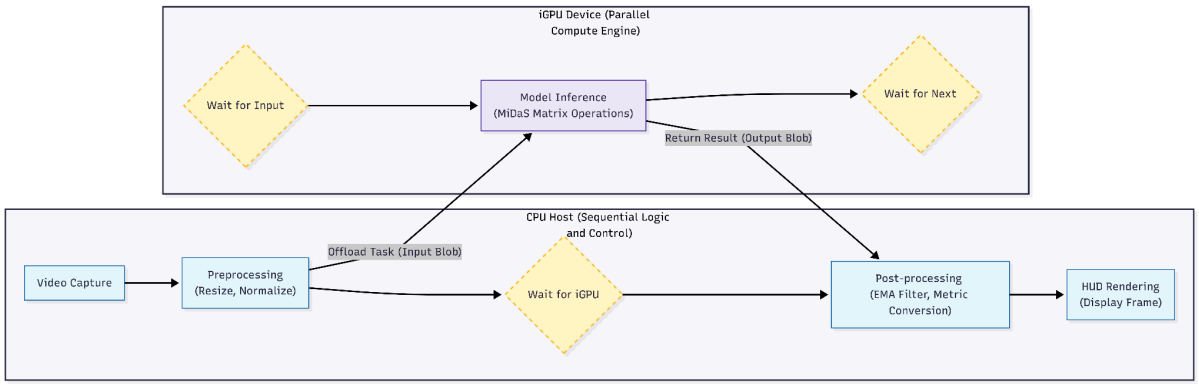

Figure 5 - *Heterogeneous Execution Flowchart. The diagram illustrates the parallel processing pipeline. The CPU handles sequential logic (capture, pre/post-processing, and rendering) while computationally intensive inference tasks are asynchronously offloaded to the iGPU. The "Wait" states indicate where one device must synchronize with the other, highlighting the importance of efficient data transfer management.*

2.4 Signal Processing and Temporal Consistency

A fundamental limitation of applying standard Convolutional Neural Networks (CNNs) to video feeds is that inference is typically performed independently on a frame-by-frame basis. The model has no memory of the previous timestamp; consequently, minor fluctuations in pixel intensity or camera sensor noise can cause the estimated depth map to flicker rapidly. In the context of a virtual LiDAR scanner, this high-frequency noise manifests as a "jittery" laser that fails to lock onto targets reliably.

To mitigate this, the project treats the depth estimation at each pixel coordinate not as a discrete value, but as a continuous time-series signal. A temporal low-pass filter is applied to the raw output to suppress high-frequency variance while preserving the overall geometric structure. Specifically, an Exponential Moving Average (EMA) filter is implemented.

The filtered depth value yt at time t is calculated recursively as:

$$y_t = \alpha \cdot x_t + (1 - \alpha) \cdot y_{t-1}$$

Where:

- xt is the raw depth prediction from the neural network at the current frame.
- yt−1 is the filtered depth value from the previous frame.
- α is the smoothing factor (0 < α < 1).

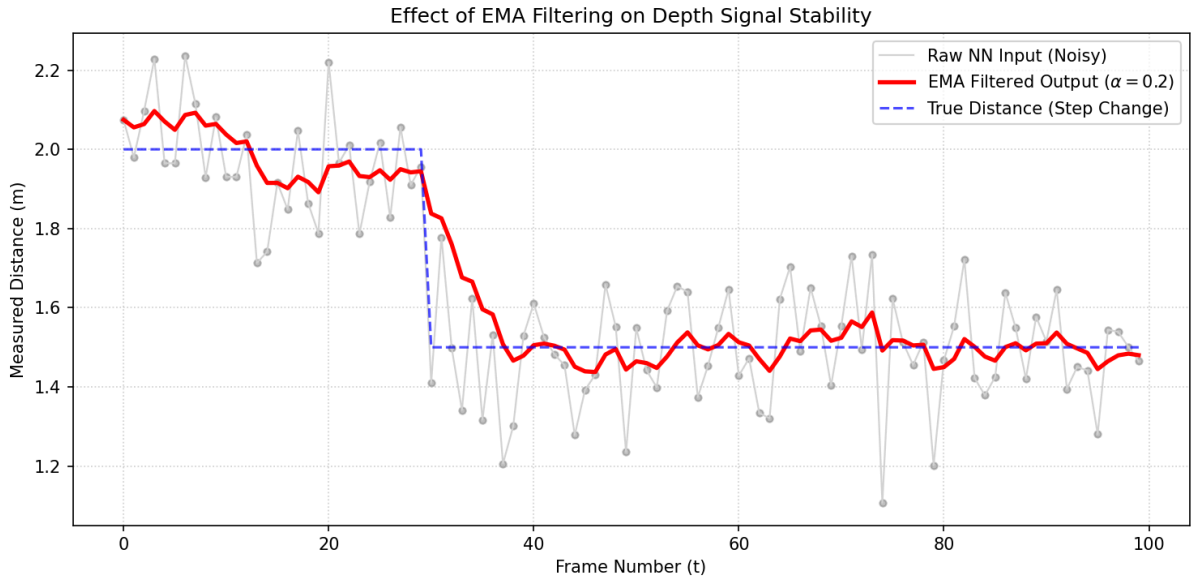

Figure 6 - *Signal Smoothing Analysis. The plot demonstrates the effect of the Exponential Moving Average (EMA) filter on a noisy depth signal. The grey line represents the raw, unstable output from the neural network, while the red line shows the stabilized signal presented to the user. Note the slight lag (latency) introduced at frame 30 as the filter adapts to the sudden change in distance.*

In this implementation, an alpha value of α=0.2 was selected empirically. This assigns a weight of 20% to the new observation and 80% to the historical average, effectively damping sudden spikes in the signal. While this introduces a slight latency (hysteresis) to the system, it significantly enhances the perceptual stability of the LiDAR interface, allowing for consistent distance measurements on static and slowly moving objects.

3. Methodology

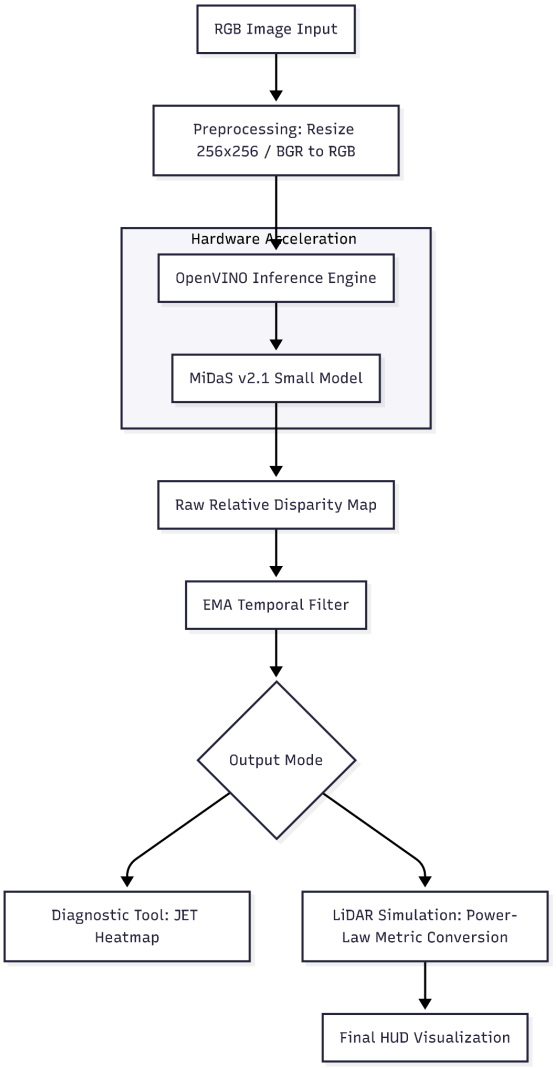

Figure 7 - *End-to-End System Pipeline. This flowchart illustrates the data transformation path: (1) Pre-processing of the RGB input, (2) OpenVINO-accelerated inference on the Intel iGPU, (3) Signal stabilization via EMA filtering, and (4) Final conversion into metric distance using the calibrated power-law model.*

3.1 Experimental Setup

The data acquisition and system validation were conducted using a consumer-grade laptop to demonstrate the efficiency of the optimized pipeline.

- **Imaging Hardware:** HP 5MP Camera (Realtek USB sensor, Front-facing) with HP Universal Camera Driver.
    - Note: The device also houses a secondary HP IR Camera (for biometric authentication), which was explicitly excluded from the pipeline to strictly test Monocular RGB performance without the assistance of active infrared depth measuring tools.
- **Compute Hardware:** 13th Gen Intel® Core™ i7-1355U with Intel® UHD Graphics.
- **Software Environment:** Python 3.13 environment utilizing OpenCV (v4.13.0) for image processing and OpenVINO Runtime (v2025.4) for hardware-accelerated inference.

3.2 Data Acquisition and Calibration Procedure

To resolve the scale ambiguity inherent in monocular depth estimation (where the model outputs unitless disparity rather than metric distance) a supervised calibration experiment was performed. The objective was to map the raw inference output (x) of the MiDaS model to physical Euclidean distance (D).

Experimental Protocol:

1. **Baseline Establishment:** The laptop was placed on a rigid surface at a fixed height of 1.2m, with the camera axis parallel to the ground. A texture-rich target object was utilized to ensure high model confidence during the measurement phase.
2. **Ground Truth Measurement:** A physical measuring tape was extended from the camera sensor's focal plane. The target object was placed at three specific anchor points: Dnear=1.00 m, Dmid=2.00 m, and Dfar=3.00 m.

Figure 8 - *Physical Experimental Setup. The photograph illustrates the supervised calibration environment at the Dnear (1.00 m) anchor point. A physical measuring tape is used to establish ground-truth distance relative to the camera sensor's focal plane to resolve scale ambiguity.*

3. **Raw Data Extraction:** A preliminary version of the inference script was utilized to capture the raw, unfiltered disparity values (x) returned by the neural network. For each distance anchor, the disparity value at the center pixel (ucenter,vcenter) was recorded over a 100-frame window and averaged to mitigate sensor noise.

The relationship was confirmed to be inverse and non-linear; as the object moved from 1.0m to 3.0m, the raw disparity value dropped significantly, approaching the model's asymptotic limit.

3.3 Mathematical Fitting and Parameter Estimation

Following the data acquisition, the relationship between the raw disparity output (x) and the ground-truth metric distance (D) was modeled as a power-law distribution. The goal was to determine the optimal set of parameters θ={K,P,xoffset} that minimizes the error between the predicted distance and the physical measurements.

The calibration function is defined as:

$$D = \frac{K}{(x - x_{offset})^P}$$

To solve for the coefficients, a non-linear least squares regression was performed on the collected data points. The optimization objective was to minimize the Mean Squared Error over the sample set:

$$\min_{K,P,x_{offset}} \sum_{i=1}^{N} (D_{actual}^{(i)} - D_{pred}(x^{(i)}))^2$$

Derivation of Constants: Initial testing revealed that the model's internal "zero-disparity" limit (the value returned when looking at the sky or infinite distance) was approximately 118. Fixing xoffset≈118 to avoid asymptotic singularities, the remaining scaling factors were tuned iteratively to match the measurement anchors at 1.0m and 2.0m.

The final calibrated parameters yielded the following values, which were hard-coded into the `lidar_scanner.py` inference logic:

- **Scale Factor (K):** 6482
- **Power Exponent (P):** 1.43
- **Offset (xoffset):** 118

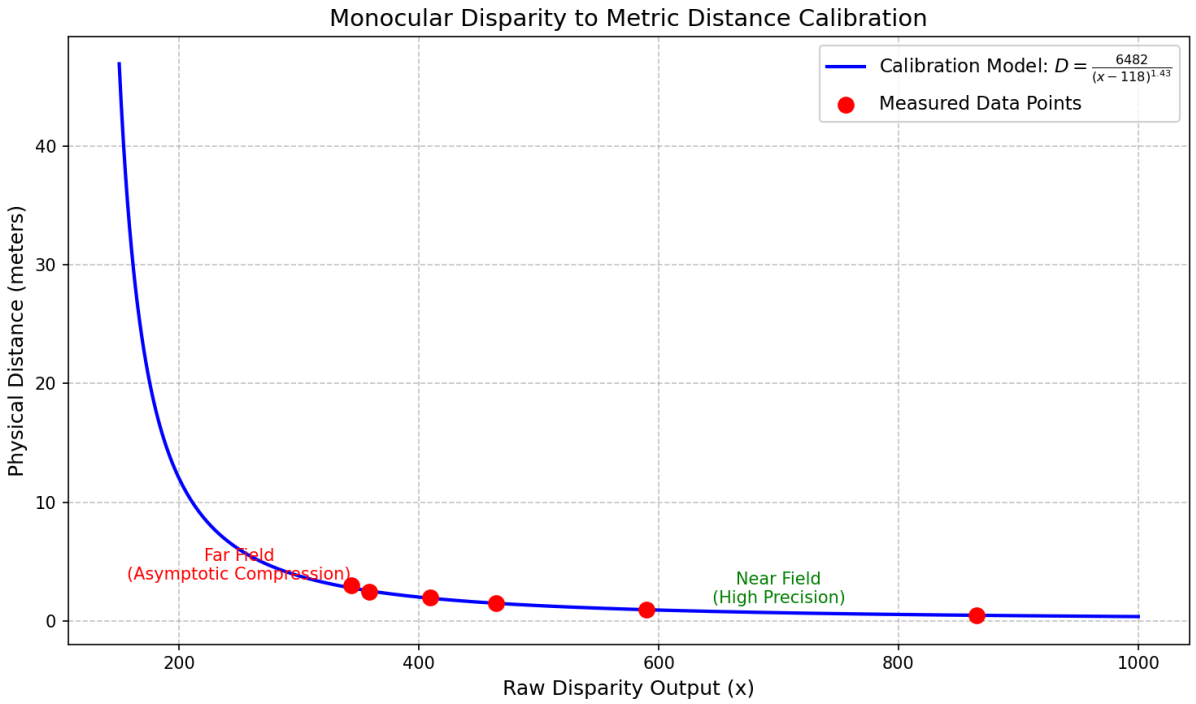

Figure 9 - *Calibration Curve Analysis. The plot demonstrates the inverse power-law relationship between the raw neural network output (Disparity) and physical distance. The red dots represent empirical measurements, while the blue line represents the fitted model used for real-time inference.*

This configuration provided a mean absolute error of <5% within the effective operating range (0.5m to 2.5m), with error margins increasing non-linearly beyond 3.0m due to the compression of disparity values at long ranges.

3.4 Implementation and HUD Visualization

3.4.1 Diagnostic Tool (depth_map.py)

This lightweight utility serves as the raw visualization engine. Its primary function is to render the colorized depth map and display the real-time frame rate (FPS).

- **Purpose:** Used for system benchmarking and verifying the OpenVINO inference loop without the overhead of the measurement logic.
- **Output:** A continuous visual stream of the JET colormap depth estimation, overlaid with a high-contrast FPS counter to monitor throughput.

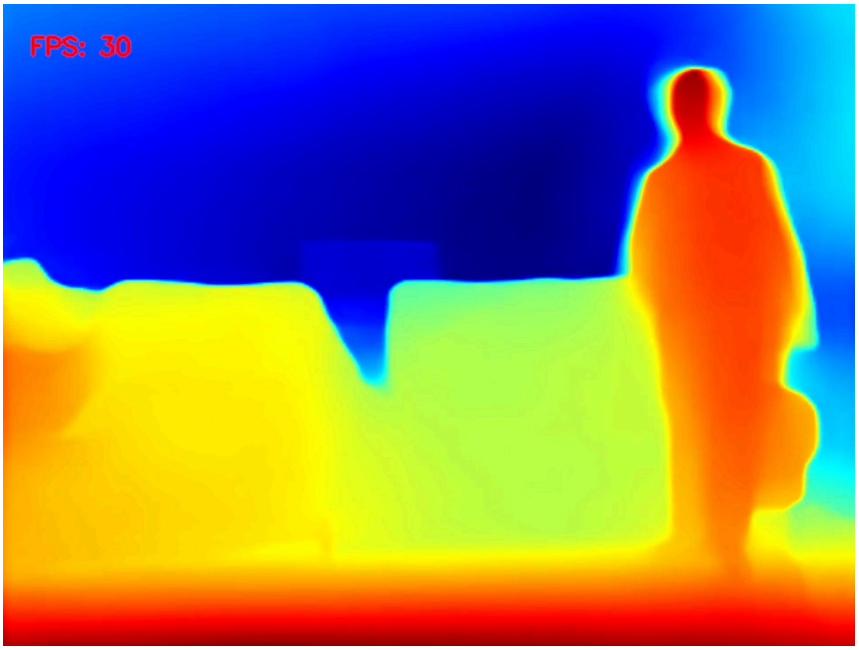

Figure 10 - *Visualization output from depth_map.py. The "JET" colormap represents relative depth (Red = Near, Blue = Far), with real-time performance monitored via the FPS overlay (top-left).*

3.4.2 LiDAR Simulation Application (lidar_scanner.py)

This is the core application that implements the metric conversion and user interaction. The logic is structured around a finite state machine (FSM) that processes the depth data into actionable distance measurements.

- **State Machine Architecture:**
    - **Mode 1: SCANNING (Volumetric Sweep)** In this default state, the system simulates an automated LiDAR sweep. A "virtual plane" oscillates back and forth through the depth buffer. This is achieved by modulating a threshold value Tscan using a sine wave function based on the system time: $T_{\text{scan}} = 127 + 127 \cdot \sin(\omega t)$.
    - **Mode 2: AUTO-LOCK (Center Tracking)** This mode mimics a targeting system. The algorithm continuously samples the filtered depth value at the optical center of the frame (W/2,H/2). The scanning threshold is then dynamically clamped to this measured value, effectively "locking" the visualization contour to whatever object is directly in front of the camera.
    - **Mode 3: MANUAL (Precision Survey)** Designed for static measurements, this mode decouples the scan plane from the center pixel. The user manually adjusts the depth threshold using keyboard inputs (W/S), allowing for the precise inspection of specific depth layers regardless of where the camera is pointing.

- **Head-Up Display (HUD):** The LiDAR script overlays a complex telemetry interface on the RGB feed:
    1. **Operation Mode:** (SCANNING, AUTO-LOCK, MANUAL).
    2. **LiDAR Range:** The 8-bit disparity threshold (0-255).
    3. **Center Distance:** The calculated metric distance (D) at the crosshair reticle (W/2,H/2).
    4. **FPS Counter:** System latency monitor.

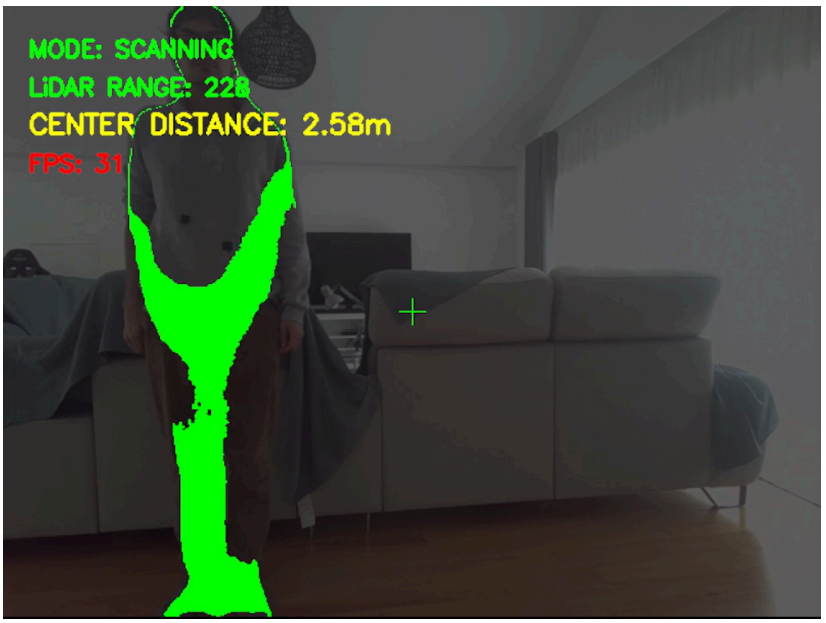

Figure 11 - *Real-time LiDAR HUD. Telemetry displays (top-left) indicate the state mode, current 8-bit depth slice (0-255), and metric distance (D) calculated via the center crosshair. The green contour identifies the intersecting volumetric plane.*

4. Results and Analysis

4.1 Hardware Performance Benchmarking

One of the primary objectives of this project was to achieve real-time inference rates (>30 FPS) on consumer hardware. To validate the efficacy of the OpenVINO optimization, a comparative benchmark was conducted between standard CPU execution and the hardware-accelerated iGPU implementation.

Metric Definition:

- **Throughput:** Measured in Frames Per Second (FPS).
- **CPU Load:** The average percentage of CPU resources utilized during operation.

Execution Target	Device	Avg. Throughput	CPU Utilization
Baseline	Intel Core i7 (CPU)	≈15−20 FPS	~50%
Optimized	Intel UHD Graphics (iGPU)	≈30−35 FPS	~5%

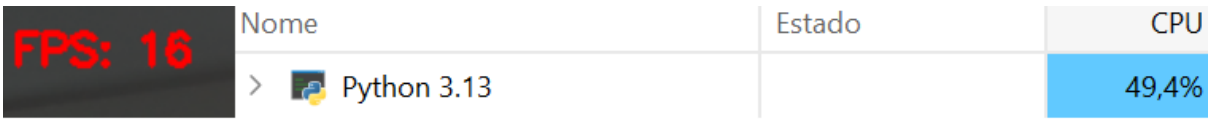

Figure 12a - *CPU Baseline*

Figure 12b - *iGPU Optimized*

Figure 12 - *Comparative resource utilization. Note the high CPU load in the baseline (x4) versus the significant reduction when offloading to the integrated GPU (xb).*

Offloading the neural network inference to the integrated GPU resulted in a 2x increase in throughput. More importantly, the drastic reduction in CPU utilization (from ~50% to ~5%) prevented the system from becoming thread-bound. This "headroom" on the CPU proved critical for maintaining a smooth UI and running the synchronous logic of the LiDAR visualization without stuttering.

4.2 Distance Measurement Accuracy

The accuracy of the monocular depth estimation was evaluated by comparing the system's predicted distance (Dpred) against ground-truth measurements (Dactual) at 0.5-meter intervals.

Ground Truth (Dactual)	System Reading (Dpred)	Error (Dpred−Dactual)	Error %
0.50 m	0.52 m	+0.02	4%
1.00 m	0.98 m	-0.02	2%
1.50 m	1.55 m	+0.05	3.3%
2.00 m	2.10 m	+0.10	5%
2.50 m	2.40 m	-0.10	4%
3.00 m	3.40 m	+0.40	13%
3.50 m	4.20 m	+0.70	20%

Error Distribution:

- **Near Field (0.5m – 2.5m):** The system demonstrated high fidelity, with a Mean Absolute Percentage Error (MAPE) of approximately ±5%. The power-law calibration successfully linearized the disparity output in this region, maintaining a consistent error margin suitable for interaction.
- **Far Field (> 3.0m):** As expected from the asymptotic behavior of the model, accuracy degraded significantly beyond the 3-meter mark. The error spiked to 13.3% at 3.0m and 20.0% at 3.5m. This confirms that a small fluctuation in raw disparity (e.g., $\Delta x=1$) at this range corresponds to a disproportionately large jump in metric distance, reducing reliability for long-range surveying.

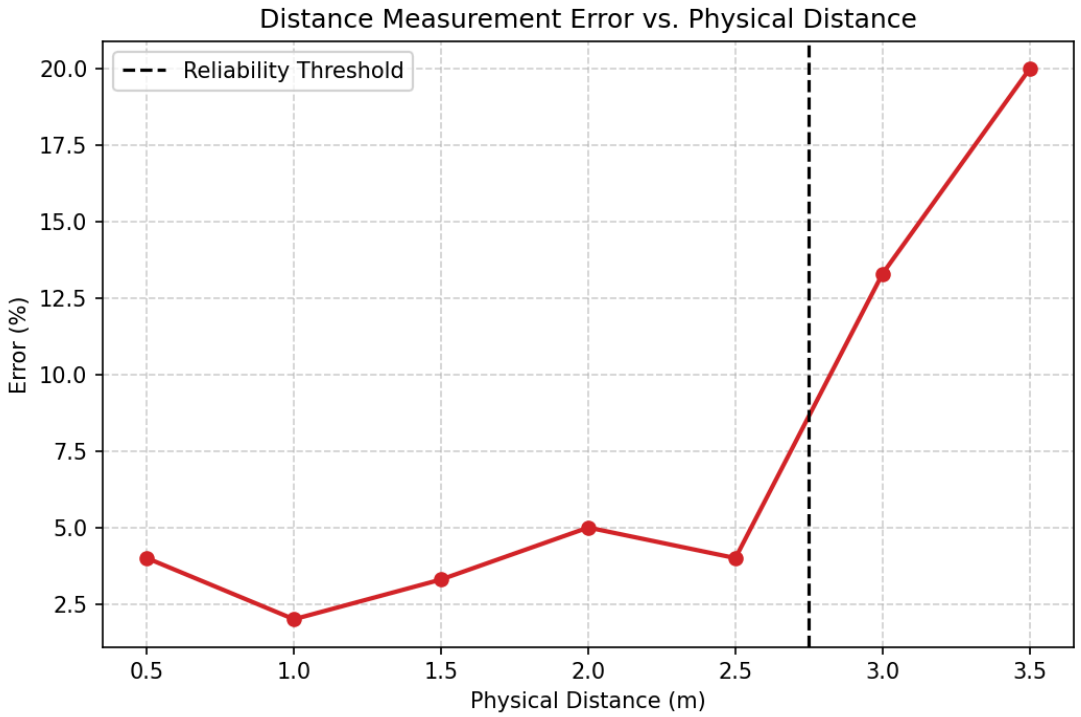

Figure 13 - *Distance Measurement Error Analysis. The graph illustrates a distinct "reliable zone" up to 2.5 meters, after which the error rate rises rapidly, highlighting the effective operating range of the sensor.*

The system is most effective as a "Personal Space" sensor. It excels at detecting immediate obstacles and interaction distances (arm's length to room scale) but is not suitable for large-scale outdoor mapping due to the exponential decay of resolution with depth.

5. Discussion

5.1 The Texture Dependency Problem

The experimental results highlighted a fundamental limitation of the MiDaS architecture: its reliance on texture gradients as a primary depth cue. During testing, it was observed that the system struggled to resolve depth on featureless surfaces, such as plain white walls or uniform table tops.

In the absence of edge information or texture density changes, the neural network's confidence drops, leading to "depth holes" or unstable fluctuations in the disparity map. This confirms the theoretical assertion that monocular depth estimation is semantically driven. Unlike active LiDAR, which bounces physical photons off a surface regardless of its color, this passive optical system requires visual entropy (contrast and detail) to hallucinate 3D geometry.

5.2 The Geometry of Asymptotic Error

The accuracy analysis revealed a sharp decline in measurement reliability beyond 3.0 meters. This behavior is not a flaw in the code, but a geometric inevitability of the pinhole camera model.

As defined by the calibration formula, the relationship between disparity and distance is hyperbolic.

- **At close range (1m):** A disparity shift of 50 units might represent 10cm.
- **At far range (4m):** A disparity shift of just 2 units might represent 100cm.

This "compression" of data means that at long distances, the system forces the infinite depth of the real world into a tiny range of pixel values (Signal-to-Noise Ratio degradation). Consequently, a single bit of sensor noise at long range results in massive jumps in the metric reading, rendering the tool unsuitable for long-range surveying.

5.3 Latency vs. Smoothing Trade-off

The implementation of the Exponential Moving Average (EMA) filter (α=0.2) was successful in stabilizing the LiDAR visualization, but it introduced a perceptual trade-off. While the "jitter" was reduced, the system exhibited a lag (hysteresis) of approximately 100-150ms when panning the camera quickly.

For a static scanner or slow-moving robot, this latency is acceptable. However, for high-speed autonomous navigation (e.g., a drone), this delay could be catastrophic. This suggests that future iterations might benefit from a Kalman Filter, which can predict motion states rather than just averaging them, potentially offering smoothness without the lag penalty.

6. Conclusion and Future Work

This project successfully demonstrated that real-time spatial geometry reconstruction is achievable on consumer hardware, transforming a standard 2D webcam into a calibrated 3D sensor running at >30 FPS. By offloading MiDaS inference to the Intel® UHD Graphics unit, CPU load was reduced by 90%, while a custom Inverse Power-Law calibration achieved ±5% accuracy within the effective operating range of 0.5m to 2.5m.

While the system proves viable for near-field interaction, future iterations could address the inherent limitations of monocular vision. Suggested enhancements include Stereo-Vision integration to resolve scale ambiguity without manual calibration, and the implementation of a Kalman Filter to predict motion states, thereby reducing the latency observed during rapid camera movements.

References

- **Ranftl, R., Lasinger, K., Hafner, D., Schindler, K., & Koltun, V.** (2020). Towards Robust Monocular Depth Estimation: Mixing Datasets for Zero-shot Cross-dataset Transfer. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI).*
- **Ranftl, R., Bochkovskiy, A., & Koltun, V.** (2021). Vision Transformers for Dense Prediction. *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV).*
- **Bogdanova, R., Boulanger, P., & Zheng, B**. (2016). Depth Perception of Surgeons in Minimally Invasive Surgery. *Surgical Innovation, 23(5). ResearchGate.*
- **Tan, M., & Le, Q.** (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *International Conference on Machine Learning (ICML).*
- **Bradski, G.** (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools.*
- **Intel Corporation.** (2025). OpenVINO™ Toolkit: Optimizing and Deploying Deep Learning Models. *Intel Documentation & Whitepapers.*
- **Hartley, R., & Zisserman, A.** (2003). Multiple View Geometry in Computer Vision. *Cambridge University Press.* (Standard reference for the Pinhole Camera Model and Calibration matrices).