Stereo Vision System for Autonomous Marine Navigation: Implementation, Integration, and Comparative Analysis

Yong-Sung Masuda

August 13, 2025

Abstract

This report presents both an implementation of a stereo vision depth estimation system integrated with semantic segmentation for autonomous marine navigation and a comparative analysis of two stereo vision depth estimation approaches: traditional OpenCV-based stereo matching and the deep learning-based FoundationStereo method. The marine navigation system utilizes the traditional stereo matching method by combining stereo camera calibration, disparity computation with advanced filtering techniques, and water/land segmentation to provide real-time obstacle detection capabilities for autonomous boats. The implementation includes a complete processing pipeline from raw stereo images to segmented distance measurements, deployed as a containerized FastAPI service supporting both ARM and x86 architectures. The comparative analysis of stereo depth estimation methods was carried out to assess potential performance gains achievable with increased computational resources. The study encompassed 120 experimental conditions, varying camera baselines (6.0 - 12.0 cm), viewing angles (-10° to +10°), and target distances (1.0–10.0 m). Each condition was processed using five distinct checkerboard calibrations. Results revealed notable performance differences among the methods, with FoundationStereo exhibiting both robustness to calibration quality and superior accuracy and consistency across all conditions.

Contents

1 Introduction

Object detection is a common feature required in many applications, particularly in robotics and autonomous vehicles. The two primary components required to implement object detection are sensors and software to interpret the data they acquire. Common types of sensors include infrared (IR), cameras, light detection and ranging (LIDAR), time-of-flight (ToF), and radio detection and ranging (RADAR). These types of sensors can generally be categorized into two types: active and passive. Active sensors—such as LIDAR, RADAR, ToF, and certain types of IR sensors—operate by emitting a signal (e.g., light, radio waves) and measuring how it reflects off surrounding objects. In contrast, passive sensors like optical cameras rely on detecting ambient signals (visible light) emitted or reflected by objects in the environment. While passive sensors typically consume less power and offer the advantage of being less detectable, they are more susceptible to environmental factors such as lighting conditions, fog, or heavy rain, which can degrade their performance. A notable emerging advantage of passive sensors is that the rich visual information they provide is becoming increasingly useful due to advancements in machine-learning techniques. A robust object detection system might combine the usage of both active and passive sensors in order to reap the benefits of both.

Cameras were selected for use in this project due to their low cost, potential for long-range object detection, passive nature, and the rich contextual information they provide for machine-learning applications. A stereo vision technique is employed, using two parallel cameras for accurate distance estimation through triangulation-based depth computation. The resulting depth maps undergo post-processing with Weighted Least Squares (WLS) filtering to improve quality and reduce noise artifacts. An image segmentation model specifically trained to mask water features is integrated with the stereo depth data, enabling reliable distinction between water bodies (classified as safe navigation zones) and land areas (potential obstacles with precise distance measurements). This integrated approach combines geometric depth estimation with semantic understanding to provide comprehensive environmental awareness for autonomous marine navigation. The implementation of a stereo vision system presents a choice between the use of traditional computer vision approaches refined over decades of computer vision research and the use of modern deep learning methods. Traditional stereo matching algorithms, such as those implemented in OpenCV, rely on classical block matching and semi-global matching techniques as well as edge-preserving filtering algorithms. Though these methods are more computationally efficient and interpretable, they have trouble dealing with certain scenarios such as textureless regions, repetitive patterns, occlusions, and varying lighting conditions. While FoundationStereo provides state of the art stereo depth estimation, it is computationally expensive and requires GPU-accelerated hardware. The selection between these two systems requires the careful consideration between cost and performance.

1.1 Scope and Objectives

This report is structured in two main parts:

Part I: System Implementation - Complete technical implementation of a stereo vision system integrated with water segmentation for autonomous marine navigation.

Part II: Comparative Analysis - An evaluation of traditional OpenCV stereo matching versus FoundationStereo deep learning approaches across multiple experimental conditions.

The purpose of this report is to provide both practical guidance for complete system deployment as well as experimental evidence to justify method selection. The implementation portion details the data processing pipeline as well as hardware and software requirements, while the analysis portion compares the performance of the deployed system in a laboratory environment with a more computationally expensive state of the art method. This methodology facilitates informed decision-making regarding system architecture choices, particularly when considering

the balance between real-time processing requirements, hardware limitations, and navigation safety margins.

Part I
System Implementation and Integration

2 System Architecture and Design

2.1 Implementation Overview

This implementation builds upon the pre-existing "AutoSailor Docker" repository, extending the previously developed monocular object detection system with stereo vision capabilities. The updated system uses stereo camera pairs to enable three-dimensional data processing, while integrating with the existing U-Net-based water segmentation model to provide depth-aware environmental perception for autonomous marine navigation.

2.2 Technical Stack

The implementation utilizes the following technologies:

- **Python 3.9**: Primary runtime environment
- **OpenCV 4.x**: Computer vision operations and stereo processing
- **PyTorch**: Deep learning framework for segmentation model inference
- **FastAPI**: Web framework for REST API endpoints
- **Docker**: Multi-architecture containerization
- **NumPy**: Numerical computations and array operations

3 Previous Monocular Approach and Limitations

3.1 Original Distance Estimation Method

The original AutoSailor Docker system implemented a monocular vision approach for distance estimation using a single camera and water segmentation. This method employed the following pipeline:

1. **Water Segmentation**: A U-Net model with ResNet34 encoder classified pixels as water (white, value 255) or land (black, value 0)
2. **Pixel-based Distance Estimation**: Distance was calculated based on the vertical position of the closest land pixel in each angular segment
3. **Empirical Formula**: A complex empirical formula converted pixel coordinates to distance estimates

The distance calculation formula used was:

$$d = \frac{75.95729 + (2047468000 - 75.95729)}{1 + \left(\frac{y_{pixel}}{138.6823}\right)^{23.52245}} \tag{1}$$

where y_{pixel} is the vertical coordinate of the closest land pixel, and d is the estimated distance in inches.

3.2 Limitations of Monocular Approach

The monocular method suffered from several fundamental limitations:

- **Lack of True Depth Information**: Distance estimation relied on geometric assumptions about camera height and viewing angle
- **Empirical Calibration**: The complex formula was derived empirically for specific camera configurations and heights
- **Environmental Sensitivity**: Performance varied with lighting conditions, water surface conditions, and camera tilt
- **Elevated Obstacle Detection**: Unable to estimate distances to objects above the water surface, such as docks, bridges, or overhanging structures, which represent navigation hazards

4 Stereo Vision Implementation

4.1 Complete Processing Pipeline

The stereo vision system implements a comprehensive pipeline that transforms raw stereo images into actionable navigation data. The complete processing flow includes:

1. **Stereo Camera Calibration**: Calculating calibration parameters using checkerboard images
2. **Undistortion and Rectification**: Preprocessing stereo image pairs using precomputed calibration parameters
3. **Disparity Map Computation**: Advanced stereo matching with Semi-Global Block Matching (SGBM) and Weighted Least Squares (WLS) filtering
4. **Depth Map Generation**: Converting pixel disparity to metric depth using camera geometry
5. **Water Segmentation**: Semantic segmentation using a pre-trained U-Net model
6. **Spatial Analysis**: Dividing the field of view into angular segments and computing nearest obstacle distances
7. **API Integration**: RESTful endpoints for real-time processing and result retrieval

4.2 Stereo Camera Calibration

Algorithm 1: Checkerboard Method

Input: Set of stereo image pairs $\{(I_{L,i}, I_{R,i})\}_{i=1}^{n}$ containing checkerboard patterns, Checkerboard dimensions (w, h), Square size s

Output: Camera matrices M_1, M_2, Distortion coefficients D_1, D_2, Rectification transforms R_1, R_2, Projection matrices P_1, P_2, Essential matrix E, Fundamental matrix F

1. Initialize object points: $objpoints \leftarrow \texttt{generateCheckerboardPoints}(w, h, s)$
2. For each stereo pair $(I_{L,i}, I_{R,i})$:
   (a) $corners_{L,i} \leftarrow \texttt{findChessboardCorners}(I_{L,i}, (w, h))$
   (b) $corners_{R,i} \leftarrow \texttt{findChessboardCorners}(I_{R,i}, (w, h))$

(c) Refine corner locations: $corners_{L,i} \leftarrow \texttt{cornerSubPix}(I_{L,i}, corners_{L,i})$

(d) Refine corner locations: $corners_{R,i} \leftarrow \texttt{cornerSubPix}(I_{R,i}, corners_{R,i})$

3. $(M_1, D_1, M_2, D_2, R, T, E, F) \leftarrow \texttt{stereoCalibrate}(objpoints, \{corners_{L,i}\}, \{corners_{R,i}\})$
4. $(R_1, R_2, P_1, P_2) \leftarrow \texttt{stereoRectify}(M_1, D_1, M_2, D_2, R, T)$
5. Return $M_1, M_2, D_1, D_2, R_1, R_2, P_1, P_2, E, F$

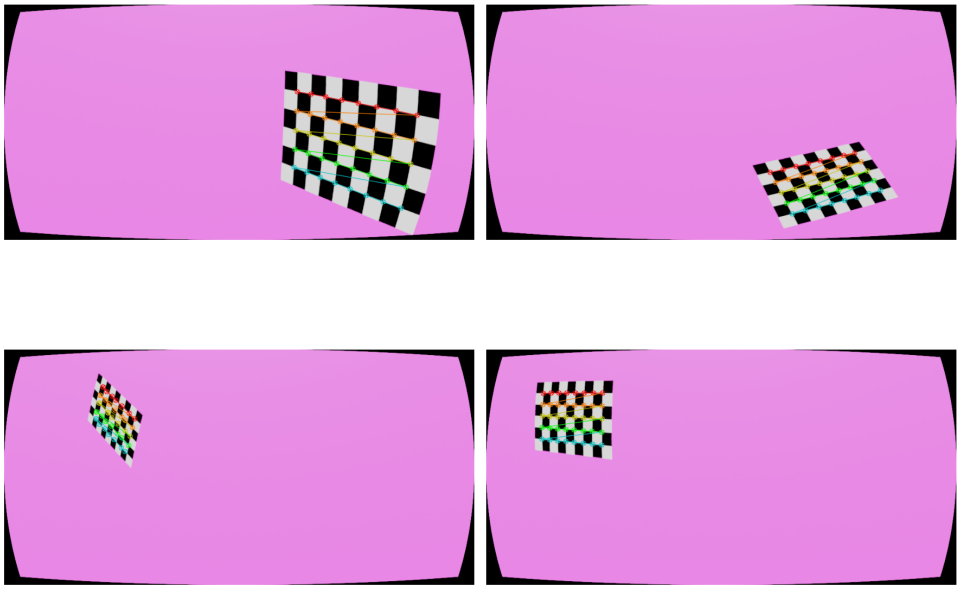

Figure 1: A calibration checkerboard is photographed in varying positions and angles in order to calculate the stereo cameras' intrinsic parameters as well as their rotations and positions relative to each other.

4.3 Distortion Correction and Rectification

The implemented pipeline begins with undistortion and rectification of the input image pair using precomputed calibration parameters. The rectification process ensures that corresponding points lie on the same horizontal lines, simplifying the stereo matching problem.

Algorithm 2: Stereo Image Rectification

Input: Left image I_L, Right image I_R, Calibration matrices M_1, M_2, Distortion coefficients D_1, D_2, Rectification transforms R_1, R_2, Projection matrices P_1, P_2

Output: Rectified images I'_L, I'_R

1. Compute rectification maps: $(map_{1x}, map_{1y}) \leftarrow \texttt{initUndistortRectifyMap}(M_1, D_1, R_1, P_1)$
2. Compute rectification maps: $(map_{2x}, map_{2y}) \leftarrow \texttt{initUndistortRectifyMap}(M_2, D_2, R_2, P_2)$
3. $I'_L \leftarrow \texttt{remap}(I_L, map_{1x}, map_{1y})$
4. $I'_R \leftarrow \texttt{remap}(I_R, map_{2x}, map_{2y})$
5. Return I'_L, I'_R

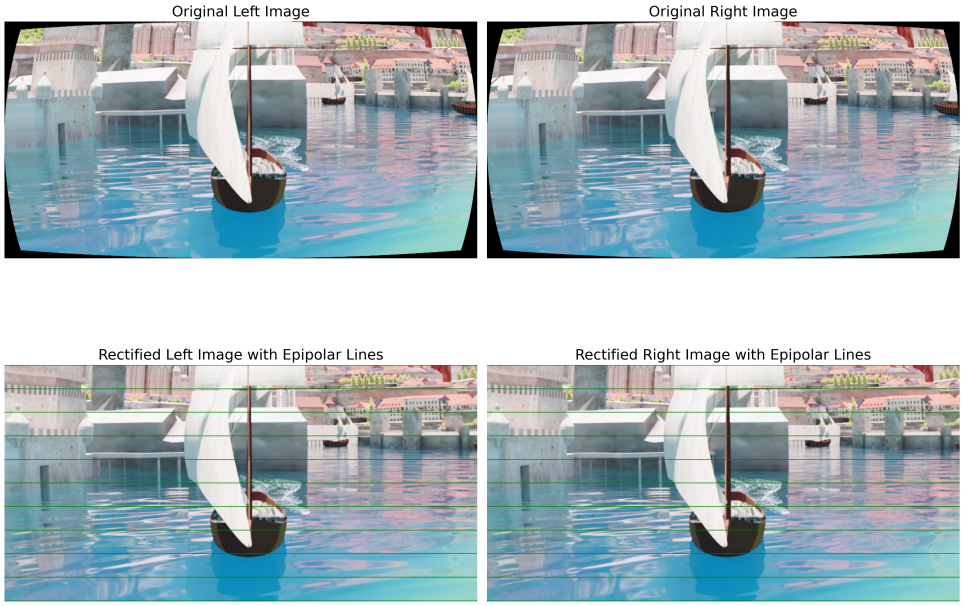

Figure 2: Raw images are undistorted and rectified using calibration parameters previously obtained through checkerboard calibration. The epipolar lines help to verify that the images are aligned vertically, facilitating horizontal stereo matching for pixel disparity calculation.

4.4 Disparity and Depth Computation

The disparity computation employs Semi-Global Block Matching (SGBM) with several enhancements:

- **Border padding**: 70-pixel padding to minimize edge artifacts
- **Parameter optimization**: Dynamically calculated P1 and P2 smoothness parameters
- **WLS post-filtering**: Weighted Least Squares filtering for disparity refinement

Figure 3: Each pixel's value represents the distance it shifted from the left image to the right

4.4.1 Depth Map Calculation

Depth values are computed using the standard stereo vision formula:

$$Z = \frac{f \times B}{d} \tag{2}$$

where:

- Z is the depth in meters
- f is the focal length in pixels
- B is the baseline distance between cameras
- d is the disparity value

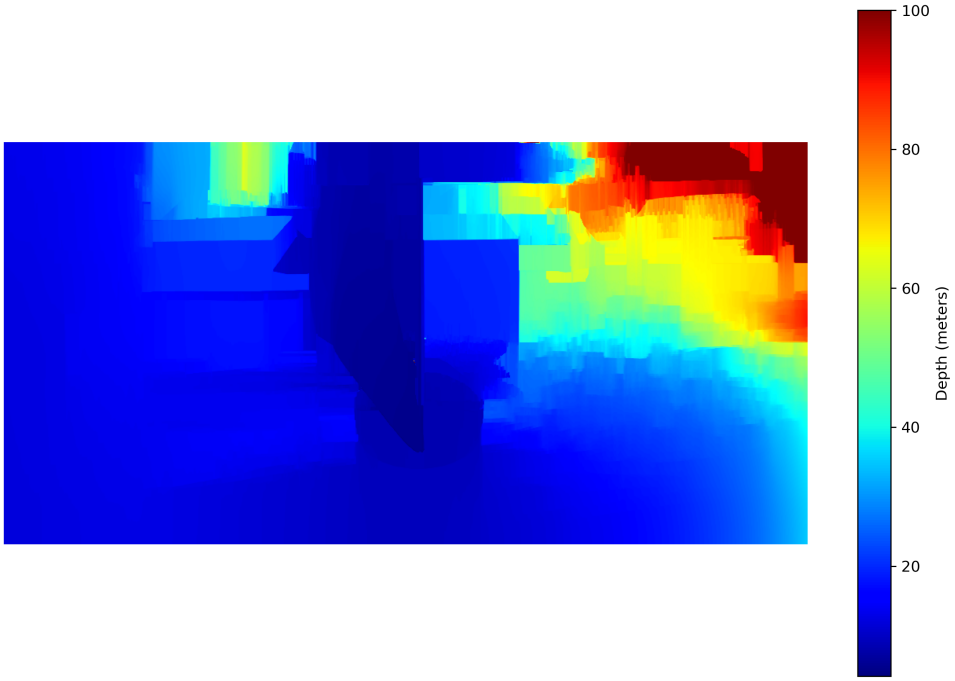

Figure 4: The depth map shows distance estimations across the entire field of view. The color-coded visualization uses a jet colormap where blue indicates closer objects and red indicates more distant objects, providing 3D spatial information at every pixel.

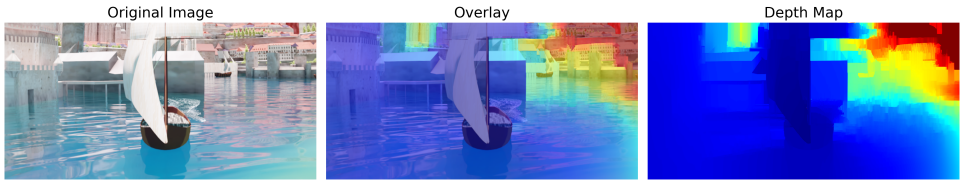

Figure 5: Depth map overlay on the original stereo image. This demonstrates the accuracy of the stereo matching algorithm by showing depth information in context with the original scene.

5 Water Segmentation Integration

The system integrates a pre-trained U-Net model with ResNet34 encoder for water/land segmentation. The segmentation pipeline includes:

1. **Preprocessing**: Resize to 512×1024 pixels, normalization to [-1, 1] range
2. **Inference**: Forward pass through the U-Net model
3. **Post-processing**: Sigmoid activation, thresholding at 0.5, and resize to original dimensions

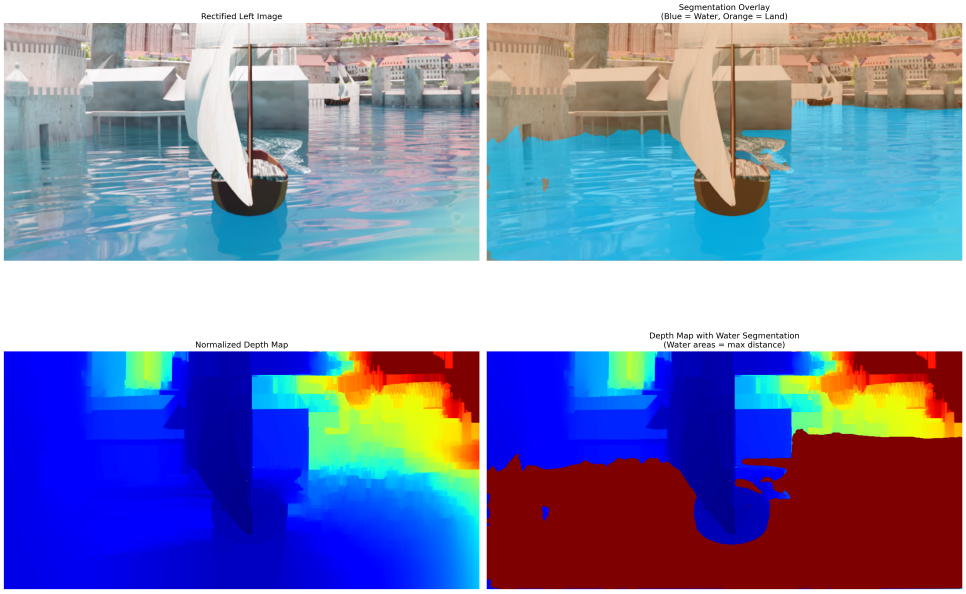

Figure 6: Water segmentation mask overlaid on the depth map, showing the integration of semantic segmentation with stereo depth estimation. Water areas are classified as safe navigation zones, while land areas represent potential obstacles with meaningful distance estimations.

6 Spatial Analysis and Navigation Output

6.1 Obstacle Distance Computation

The final stage combines depth and segmentation information to provide actionable navigation data:

Algorithm: Segment Depth Map and Find Nearest Obstacles

Input: Depth map D, Water segmentation mask W, Occlusion mask O, Number of segments N, Field of view θ_{fov}, Maximum distance $D_{\max}$, Center direction θ_c

Output: Dictionary mapping angles to nearest distances

1. Create masked depth map: $D_m[i,j] = \begin{cases} D[i,j] & \text{if } W[i,j] = 0 \text{ (land)} \\ D_{\max} & \text{if } W[i,j] = 1 \text{ (water)} \end{cases}$
2. Apply occlusion mask: $D_m[i,j] = \begin{cases} D_m[i,j] & \text{if } O[i,j] = 1 \text{ (valid)} \\ D_{\max} & \text{if } O[i,j] = 0 \text{ (occluded)} \end{cases}$
3. Calculate segment parameters:
   (a) $w_{segment} = \lfloor width/N \rfloor$
   (b) $\theta_{per\_segment} = \theta_{fov}/N$
   (c) $\theta_{start} = \theta_c - (\theta_{fov}/2)$
4. Initialize distances dictionary: $distances = \{\}$
5. For $i = 0$ to $N - 1$:
   (a) Calculate pixel boundaries:
      i. $col_{start} = i \times w_{segment}$
      ii. $col_{end} = \min(col_{start} + w_{segment}, width)$
   (b) Extract segment: $S = D_m[:, col_{start} : col_{end}]$

(c) Calculate nearest distance: $d_{nearest} = \min(S)$

(d) Calculate segment center angle:

i. $\theta_{center} = \theta_{start} + (i + 0.5) \times \theta_{per\_segment}$

ii. $\theta_{center} = \theta_{center} \bmod 360$

(e) Store result: $distances[\text{round}(\theta_{center})] = d_{nearest}$

6. Return *distances*

6.2 Final Output Visualization

The complete stereo vision pipeline produces comprehensive distance measurements for autonomous navigation.

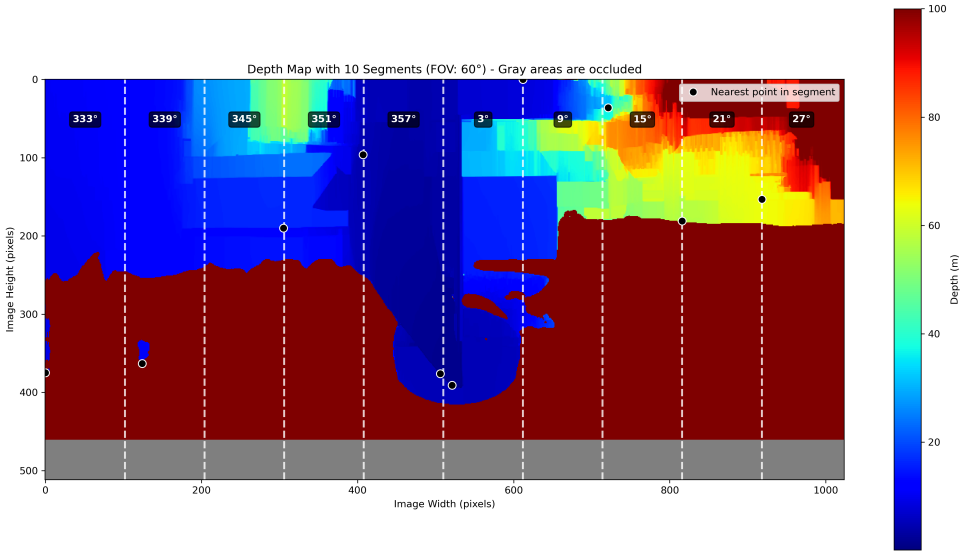

Figure 7: Final output showing angular segments overlaid on the depth map with distance measurements. Each segment represents a navigation direction with the nearest obstacle distance computed from the integrated stereo depth and water segmentation data. White dashed lines indicate segment boundaries, and black dots mark the closest detected obstacles in each sector.

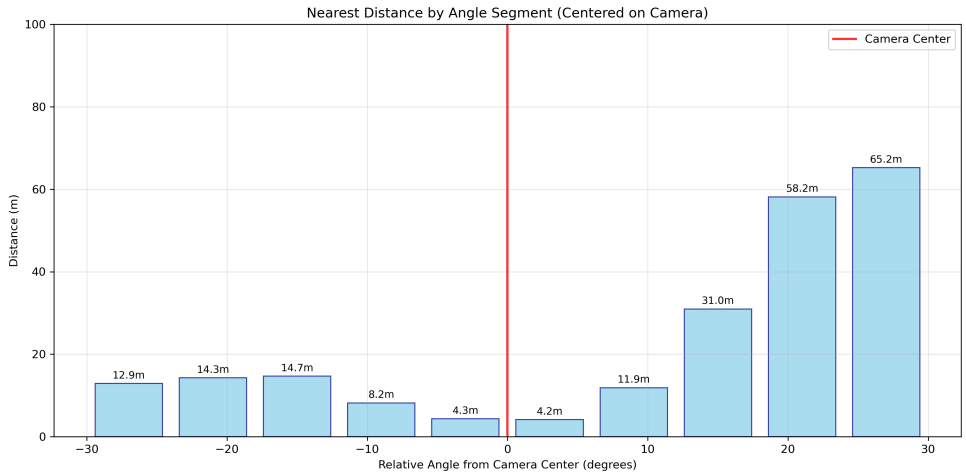

Figure 8: Quantitative distance measurements by angular segment, showing the final navigation output. The bar chart displays the nearest obstacle distance for each angular direction, providing clear actionable information for autonomous navigation algorithms. The red line indicates the camera center direction, and distances are measured in meters.

7 API Design and System Integration

7.1 RESTful Interface

The stereo vision system is exposed through a FastAPI-based REST interface with the following endpoints:

- `POST /set_stereo_parameters`: Upload stereo images and calibration data
- `GET /get_stereo_distances`: Retrieve computed distance measurements
- `GET /stereo_status`: Check processing status

7.2 Asynchronous Processing

The system implements asynchronous processing to prevent blocking during computationally intensive operations:

Listing 1: Asynchronous Processing Implementation

```
class Stereo:
    def __init__(self):
        self.processing_stereo = False
        self.distances = {}

    def compute_stereo_distances(self):
        if self.processing_stereo:
            raise SystemError("Already processing stereo images")

        self.processing_stereo = True
        try:
            # Stereo processing pipeline
            # ... (rectification, disparity, depth, segmentation)
            self.processing_stereo = False
            return self.distances
        except Exception as e:
            self.processing_stereo = False
```

```
18                raise e
```

8 Performance Analysis and Validation

8.1 System Performance Metrics

Figure 9 depicts the performance of 2 different computer systems running this pipeline for varying image scale factors.

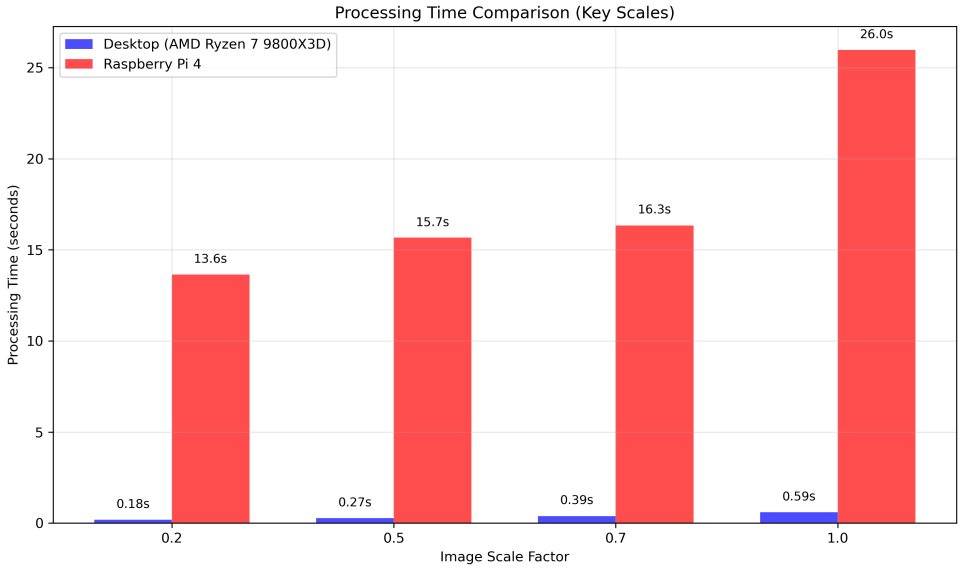

Figure 9: Computation time on a desktop computer vs Raspberry Pi 4

9 Deployment and Multi-Architecture Support

9.1 Containerized Deployment

The system supports deployment across multiple architectures through Docker containerization:

- **x86 Development**: Full PyTorch with CUDA support for development and testing
- **ARM Production**: CPU-optimized PyTorch for Raspberry Pi deployment
- **Multi-stage builds**: Optimized container sizes for production deployment

9.2 Performance Optimization

Several optimizations were implemented to improve processing speed:

- **Efficient memory management**: In-place operations where possible
- **Vectorized computations**: NumPy operations for bulk array processing
- **Selective filtering**: Optional WLS filtering based on quality requirements
- **Configurable parameters**: Adjustable segment count and field of view

Part II
Comparative Analysis of Stereo Vision Methods

10 Methodology and Experimental Design

10.1 Experimental Framework

Both stereo vision methods were evaluated using the same experimental dataset to ensure fair comparison. The dataset encompassed:

- **Camera Baselines:** 6.0, 8.0, 10.0, and 12.0 cm
- **Viewing Angles:** -10°, 0°, and +10°
- **Target Distances:** 1.0 to 10.0 m (1 m increments)
- **Calibrations:** 5 independent calibration sessions per method
- **Total Conditions:** 120 experimental scenarios per method

10.2 Methods Under Evaluation

10.2.1 Traditional OpenCV Approach

The traditional method employs classical stereo matching algorithms implemented in OpenCV, including:

- Block matching for correspondence estimation
- Semi-global block matching (SGBM) for improved accuracy
- Standard post-processing filters
- Geometric triangulation for depth calculation

10.2.2 FoundationStereo Deep Learning Approach

FoundationStereo represents a state-of-the-art deep learning approach featuring:

- Convolutional neural networks trained on large stereo datasets
- End-to-end learning of stereo correspondence
- Advanced feature extraction and matching
- Learned post-processing and refinement

11 Performance Analysis Results

11.1 Overall Accuracy Comparison

Table 1 summarizes the overall performance metrics for both methods across all experimental conditions.

The results demonstrate a dramatic performance advantage for FoundationStereo, with approximately 10× lower mean absolute error and relative error compared to the traditional OpenCV approach.

Table 1: Stereo Vision Method Comparison: OpenCV vs FoundationStereo

Metric	**OpenCV**	**FoundationStereo**	**Improvement (%)**
Total Measurements	600	600	0.00
Absolute Error Metrics (m)			
Mean Absolute Error	1.7660	0.5670	67.89
Median Absolute Error	0.6699	0.1798	73.16
Min Error	0.0013	0.0001	90.98
Max Error	22.9307	36.6143	-59.67
Std Dev Absolute Error	3.0388	1.7472	42.50
Relative Error Metrics (%)			
Mean Relative Error	70.0106	7.7773	88.89
Median Relative Error	10.8813	3.6080	66.84
Std Dev Relative Error	228.1086	18.1158	92.06

12 Distance-Based Performance Analysis

12.1 Accuracy vs. Target Distance

Figure 10 shows how accuracy varies with target distance for both methods.

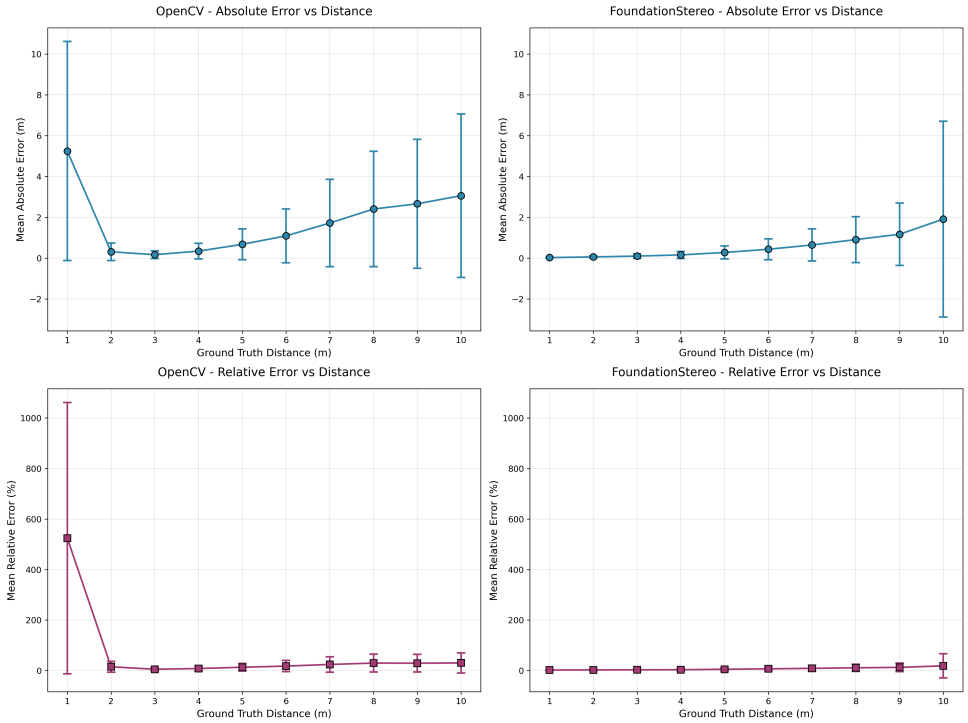

Figure 10: Mean absolute error and mean relative error vs. target distance comparison

12.2 Distance and Baseline-Based Performance Analysis

Figure 11 illustrates the error distribution patterns for both methods across all baselines and distances, grouped by calibrations and viewing angles.

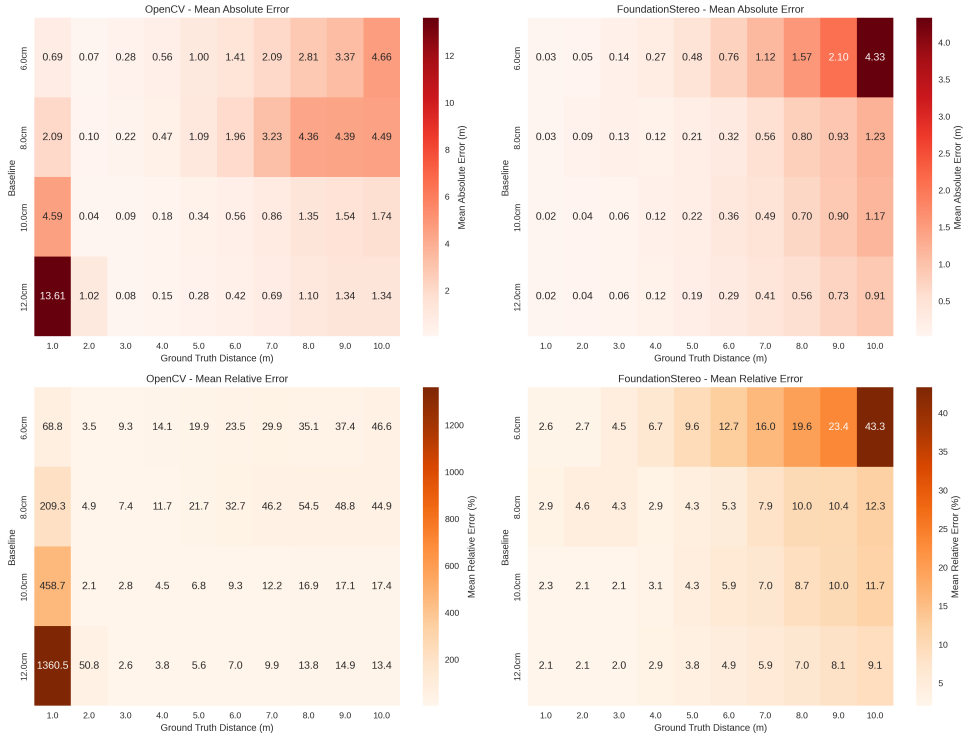

Figure 11: Absolute and relative signed error distribution comparison between methods

These heatmaps reveal that OpenCV's performance degrades at short distances (1-2 meters) as the baseline increases. This degradation is expected because larger baselines create greater distances between cameras, resulting in more occluded areas. In contrast, FoundationStereo does not exhibit this performance decline with increased baseline distance. However, both methods show progressively worse performance at larger distances when using smaller baselines. This phenomenon is especially pronounced in FoundationStereo—while it still outperforms OpenCV overall, the heatmap demonstrates a dramatic reduction in its performance as distance increases with the shortest baseline tested (6 cm).

13 Calibration Consistency Analysis

13.1 Inter-Calibration Variability

Figure 12 depicts box plots comparing the consistency across different calibrations for both OpenCV and FoundationStereo.

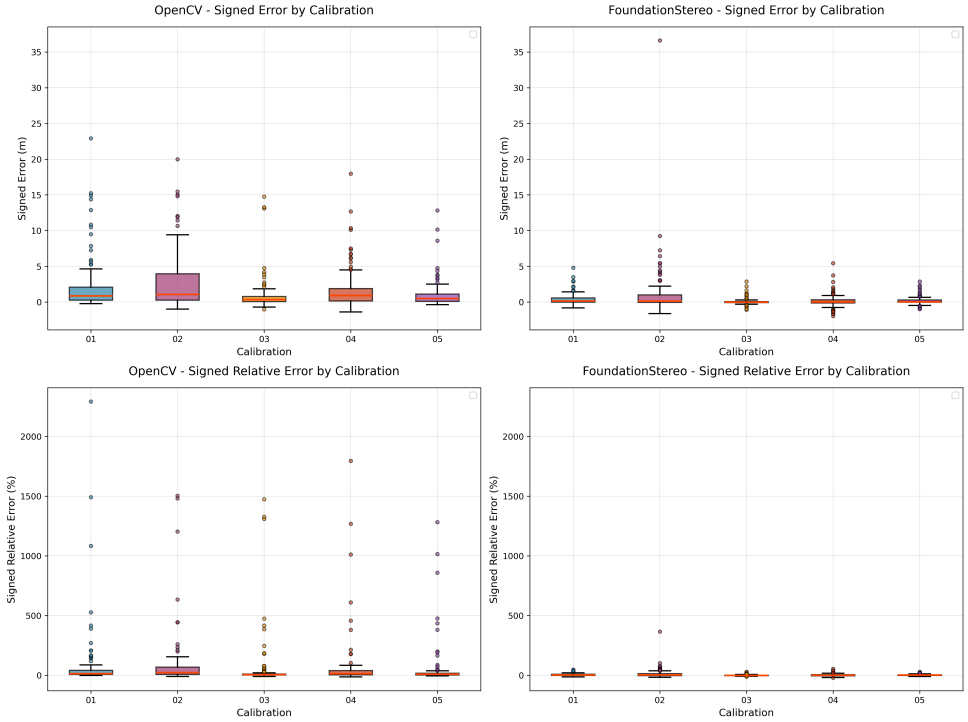

Figure 12: Absolute and mean error distribution comparison between methods

These box plots reveal that OpenCV is more sensitive to calibration quality, with significant outliers extending far beyond the interquartile range. In contrast, FoundationStereo demonstrates much greater consistency across calibrations, with fewer and less extreme outliers. However, FoundationStereo's error distributions still correlate with OpenCV's patterns across different calibrations. This indicates that while FoundationStereo is considerably more resilient to calibration variance, it still benefits from high-quality calibration data.

13.2 Key Findings from Comparative Analysis

The evaluation across 120 experimental scenarios per method reveals several insights for stereo vision system deployment.

1. **Accuracy Superiority:** FoundationStereo achieves approximately 10× improvement in both mean absolute error (1.766m to 0.567m) and mean relative error (70.01% to 7.78%) compared to OpenCV, with error reductions of 67-89% across key metrics.

2. **Distance-Dependent Characteristics:** OpenCV exhibits extremely unreliable estimations for short distances (1-2m) with larger baselines likely due to increased occlusion effects. FoundationStereo maintains consistent performance across baseline variations at short distances but shows pronounced decline at extended distances with minimal baselines (6cm) though still outperforming OpenCV.

3. **Calibration Robustness:** FoundationStereo demonstrates substantially reduced sensitivity to calibration quality variations, with fewer extreme outliers and tighter error distributions across independent calibration sessions, suggesting lower maintenance requirements and improved field reliability.

4. **Deployment Implications:** FoundationStereo excels in applications requiring precision depth estimation but demands significant computational resources. OpenCV, while less accurate, offers substantial computational efficiency and would greatly benefit from data

assimilation techniques such as Extended Kalman Filters or particle filters to smooth temporal inconsistencies. With proper filtering, OpenCV's lightweight computational footprint makes it suitable for real-time object detection and rough distance estimation in resource-constrained environments, while FoundationStereo remains optimal for applications where depth precision is necessary.

14 Conclusions and Future Work

14.1 Key Achievements

1. **Technical Implementation**: Complete production-ready system integrating stereo depth estimation with semantic segmentation
2. **Practical Deployment**: Multi-architecture containerized solution suitable for both development and embedded platforms
3. **Integration Success**: Seamless integration with existing AutoSailor Docker platform
4. **Scientific Validation**: Comparative analysis proving 10× accuracy improvement of deep learning approaches over traditional methods

14.2 Technical Contributions

The key technical contributions include:

- Integrated segmentation framework combining depth and semantic information
- Spatial analysis approach providing navigation-relevant distance measurements
- Robust API design with asynchronous processing capabilities

14.3 Future Research Directions

Several areas for future development have been identified:

1. **Real-time Optimization**: GPU acceleration and algorithm optimization for faster processing
2. **Multi-modal Fusion**: Integration with other sensor modalities (GPS, IMU, compass, radar)
3. **Machine Learning Enhancement**: Learning-based stereo matching for improved accuracy in challenging conditions
4. **Environmental Robustness**: Extended testing in various weather and lighting conditions

Acknowledgements

I would like to express my sincere gratitude to several individuals and organizations who made this research possible. First, I extend my deepest appreciation to my advisor, Professor Peter Sadowski from the Information and Computer Science department at the University of Hawaii, for his guidance, support, and valuable insights throughout this project. I am grateful to Professor Huaijin Chen for his consultation and for allowing me to audit his computer vision course, which provided essential background knowledge that directly contributed to this work. Special

thanks to Thomas Kline from the Naval Postgraduate School, who served as my sponsor for this project. I also acknowledge Jason Gray for his work on the AutoSailor Docker repository, which served as a foundation for my implementation. This work was conducted within the Information and Computer Science department at the University of Hawaii, and I appreciate the resources and academic environment provided by the university. Finally, I acknowledge the open-source software libraries that were instrumental to this project: OpenCV for traditional stereo vision processing, FoundationStereo for deep learning-based depth estimation, and PyTorch for neural network implementation and training.

References

[1] Bradski, G. and Kaehler, A. *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, Sebastopol, CA, 2008.

[2] Tiangolo, S. "FastAPI." Available at: `https://fastapi.tiangolo.com/`, 2024.

[3] Bovcon, B. and Kristan, M. "WaSR—A Water Segmentation and Refinement Maritime Obstacle Detection Network." *IEEE Transactions on Cybernetics*, 52(12):12661–12674, 2021.

[4] Dubois, A.-S. "How Does LiDAR Compare to Cameras and Radars?" *Outsight Insights*, 2023. Available at: `https://insights.outsight.ai/how-does-lidar-compares-to-cameras-and-radars/`.

[5] Hirschmüller, H. "Stereo Processing by Semiglobal Matching and Mutual Information." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2):328–341, 2007.

[6] Nayar, S. K. "Foundations of Computational Vision." Columbia University. Available at: `http://fpcv.cs.columbia.edu`.

[7] OpenCV Team. "OpenCV Documentation." Available at: `https://docs.opencv.org`.

[8] Raspberry Pi Foundation. "Raspberry Pi 4 Model B Specifications." Available at: `https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/`, 2023.

[9] Simon, D. *Optimal State Estimation: Kalman, H_∞, and Nonlinear Approaches*. John Wiley & Sons, Hoboken, NJ, 2006.

[10] Wen, B., Yu, K., Li, X., Huang, K., and Zhu, Z. "FoundationStereo: Zero-Shot Stereo Matching." arXiv preprint arXiv:2501.09898, 2025.

[11] "Stereo Vision with two RPi Cameras." MPR Projects. Available at: `https://mpr-projects.com/index.php/2025/05/08/stereo-vision-with-two-rpi-cameras/`, May 2025.