An Efficient and Reliable Real-Time Obstacle Detection System Using YOLO11 and SGBM

Tongle Yao
Northeastern University
yao.to@northeastern.edu

Xuedong Pan
Northeastern University
pan.xue@northeastern.edu

Yufei Huang
Northeastern University
huang.yufe@northeastern.edu

Siyi Gao
Northeastern University
gao.siyi1@northeastern.edu

***Abstract*—This project introduces a real-time obstacle detection and navigation assistance system designed to empower visually impaired individuals by providing an efficient, reliable, and low-cost solution for spatial awareness. The system leverages both traditional computer vision techniques and cutting-edge AI methodologies, integrating three core components: object detection, distance estimation, and audible feedback. Utilizing a customized dataset derived from Objects365 and Cityscapes, we fine-tuned YOLO11-nano and YOLO11-s models to achieve precise detection of 42 obstacle classes relevant to visually impaired users, including everyday items and potential hazards. For distance estimation, we employed a purely visual stereo camera approach using the Semi-Global Block Matching (SGBM) algorithm, enhanced with a dynamic Region of Interest (ROI) mask, which improved processing efficiency fivefold. Detected obstacle classes and distances are audibly conveyed to users via a text-to-speech (TTS) module powered by pyttsx3, enabling real-time situational awareness.**

With extensive optimization, the final system achieved a mean precision (mP) of 56% (mP@50) and 41% (mP@50-95), along with a real-time processing speed of 20+ FPS, ensuring both high accuracy and responsiveness. Unlike existing commercial solutions, which often rely on costly hardware and computational resources, our system is lightweight, portable, and accessible for deployment on mobile devices. Key contributions include the creation of a novel obstacle dataset tailored to this application and the integration of optimized components into a unified system that effectively addresses both performance and cost considerations for visually impaired navigation.

***Index Terms*—Obstacle Detection, SGBM, Stereo Vision, TTS System, YOLO11**

I. Introduction

Navigational assistance for visually impaired individuals has become an important focus in the fields of computer vision and assistive technology. Existing commercial solutions, such as AI smart glasses or integrated features like Apple's door detection in the Magnifier app, have made significant advancements [1]. However, these systems often rely on expensive hardware, such as LiDAR or infrared sensors, or require continuous cloud connectivity for computation. These limitations significantly increase the cost, complexity, and energy consumption of the devices, making them less accessible for widespread use. The need for a low-cost, lightweight, and efficient solution that can operate independently on edge devices remains a significant challenge in this domain.

This project introduces a real-time obstacle detection and navigation assistance system that addresses these limitations by leveraging purely vision-based techniques. The system integrates object detection and distance estimation to provide visually impaired users with enhanced spatial awareness. Built upon the latest YOLO11 object detection model and the Semi-Global Block Matching (SGBM) algorithm for stereo vision, the system achieves high accuracy and reliability while remaining computationally efficient. Unlike existing solutions that rely on multiple sensors, this system simplifies hardware requirements, reduces cost, and can be deployed on edge devices with minimal computational resources.

The key innovation in this project lies in the dynamic Region of Interest (ROI) optimization, which significantly improves the efficiency of SGBM. While SGBM is recognized for its reliability in depth estimation compared to other purely visual algorithms, its computational cost remains a bottleneck. By constraining the ROI to areas containing detected objects, as identified by YOLO11, the system minimizes the input size for SGBM processing, achieving a fivefold increase in processing efficiency without compromising accuracy. Additionally, the modular design of the system allows for easy integration of alternative detection models or depth estimation methods, providing flexibility for future enhancements.

The primary contributions of this project are as follows:

1) The development of a low-cost, edge-compatible real-time obstacle detection system for visually impaired users.
2) Integration of YOLO11 and SGBM with dynamic ROI optimization to improve efficiency and accuracy.
3) Creation of a custom obstacle dataset tailored for assistive navigation, leveraging Objects365 and Cityscapes.
4) A modular system architecture that allows for scalability and adaptability to different hardware and software configurations.

The remainder of this paper is organized as follows: Section II details the methodology, including dataset selection, model training, camera calibration, and the integration of YOLO11 and SGBM. Section III presents experimental results, evaluations, and performance. Section IV concludes the paper with potential directions for future works.

II. Methodology

This section outlines the methodologies employed in the selection and pre-procesisng of datasets, object detecion model selection, YOLO11 model training, camera set up and calibra-

tion, integration of Semi-Global Block Matching (SGBM) with YOLO11, and text-to-speech module.

A. Dataset Selection

To address the needs of obstacle detection in both indoor and outdoor environments, we selected Objects365 as the primary data source. Objects365 is a large-scale, high-quality object detection dataset comprising 365 categories, 638,000 images, and over 10,101,000 annotated bounding boxes [2]. Compared to other mainstream datasets, such as COCO, ImageNet, and Pascal VOC, Objects365 offers the following significant advantages:

1) *Rich Category and Scene Coverage*: Objects365 includes a wide range of everyday object categories, especially those commonly found in indoor scenarios, such as furniture, appliances, and household items. This coverage is particularly critical for our task, as many indoor obstacles are often overlooked in other datasets.

2) *High-Density Annotations*: Each image in Objects365 contains an average of 15.8 annotated bounding boxes, more than double that of COCO [2]. This high-density annotation makes the dataset well-suited for multi-object scenarios.

3) *Diverse Lighting and Image Quality*: Objects365 includes images captured under various lighting conditions (e.g., daytime, nighttime, overcast) and different levels of image clarity, ensuring model robustness in complex environments.

Based on our definition of obstacles—"fixed, non-evading objects that appear in pedestrian pathways"—besides person, we selected 42 categories from the 365 available in Objects365, which is shown in Table I. These categories encompass common indoor and outdoor obstacles, such as chairs, tables, and sofas, as well as trash bins, bicycles, and vehicles. This selection ensures that the model focuses on obstacles relevant to visually impaired users.

However, we observed an imbalance in the number of samples across the selected categories. Fig. 1 shows the sample distribution for each category in the filtered dataset. To address this, we applied techniques such as balanced subset selection, data augmentation, and a custom data loader during training to equalize sample distribution across classes. These details are elaborated in Section II-C.

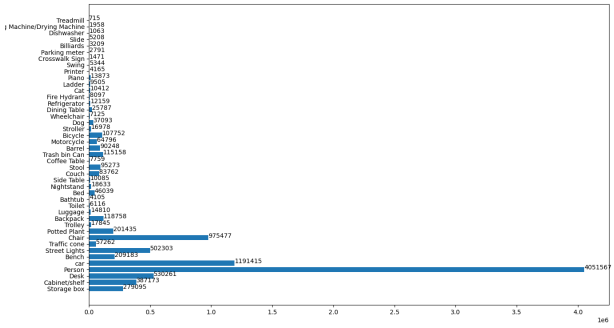

Fig. 1. Original Class Density Plot on the Selected 42 Classes

TABLE I
42 OBSTACLE CATEGORIES

Index	Label Name
0	Person
1	Trash bin Can
2	Chair
3	Wheelchair
4	Desk
5	Car
6	Storage box
7	Barrel
8	Street Lights
9	Stroller
10	Stool
11	Cabinet/shelf
12	Potted Plant
13	Trolley
14	Motorcycle
15	Ladder
16	Bicycle
17	Bench
18	Dog
19	Bathtub
20	Toilet
21	Luggage
22	Backpack
23	Bed
24	Dining Table
25	Nightstand
26	Couch
27	Refrigerator
28	Traffic cone
29	Piano
30	Side Table
31	Coffee Table
32	Parking meter
33	Cat
34	Fire Hydrant
35	Crosswalk Sign
36	Printer
37	Billiards
38	Swing
39	Dishwasher
40	Washing Machine/Drying Machine
41	Slide
42	Treadmill

B. Dataset Cleaning

To ensure the dataset meets the requirements for obstacle detection and YOLO-compatible training, we performed a comprehensive cleaning and format conversion of the Objects365 dataset. The cleaning process involved dataset splitting, category filtering, annotation format conversion, and boundary handling.

The original annotation files in Objects365 use the COCO format, with individual JSON files reaching sizes up to 3GB, making direct processing inefficient. To improve efficiency, we first split the annotation files, storing annotations for each image in a separate file named after the corresponding image. This approach not only expedited subsequent filtering and conversion but also simplified data management.

For category filtering, we retained annotations for the 42 obstacle categories defined in Section II-A. If an image's annotations did not include any of the target categories after filtering, both the image and its annotation file were removed.

This step significantly reduced the number of files to process while maintaining the dataset's precision and relevance.

The filtered annotation files remained in COCO format, which, though suitable for various detection tasks, contains unnecessary information for YOLO training. To simplify the data structure and facilitate visual inspection of annotations, we converted the files to the LabelMe format. The LabelMe format is more intuitive and can be directly viewed using the LabelMe software, enabling quick verification of the filtered annotations.

After ensuring the correctness of the LabelMe-format data, we further converted it into the text format required by YOLO. During this process, bounding box coordinates were normalized to the range [0, 1]. If any coordinates exceeded the boundaries (less than 0 or greater than 1), they were corrected to ensure data integrity. The final annotation files contained the target category IDs and their normalized bounding box coordinates.

C. Data Balancing

In the actual dataset, we observed a significant class imbalance. As shown in the Fig. 1, certain classes (e.g., person and car) have a much larger number of samples compared to other classes, while some classes (e.g., treadmill and washing machine) have very few samples. This imbalance can cause the model to be biased towards the larger classes during training, severely impacting the performance of detecting small class objects. Therefore, balancing the dataset is a crucial step in training our object detection model.

To address the issue of data imbalance, we implemented the following strategies:

We first filtered the original dataset by class, prioritizing the classes with fewer samples. All samples from these smaller classes were included in the sub-dataset until their number reached a predefined limit (e.g., 10,000 or 30,000 labeled boxes). For classes with a larger number of samples, only a subset was retained to meet the predefined limit, and any excess samples were discarded. Fig. 2 illustrates the distribution of the balanced dataset. As can be seen, while the number of samples for classes like person and car remains large, the gap between these and the other classes has been significantly reduced.

To address the slight imbalance that still exists within the dataset, we developed a custom 'torch.utils.data.Dataset'. During initialization, the dataset counts the number of samples for each class and calculates the weight w_i based on the class distribution, where:

$$w_i = \frac{\sum_{j=1}^{C} c_j}{c_i}$$

Here, c_i is the number of instances for the i-th class, and $\sum_{j=1}^{C} c_j$ is the total number of instances across all classes. The weight w_i is then used to compute the probability P_k for each class:

$$P_k = \frac{w_k}{\sum_{l=1}^{n} w_l}$$

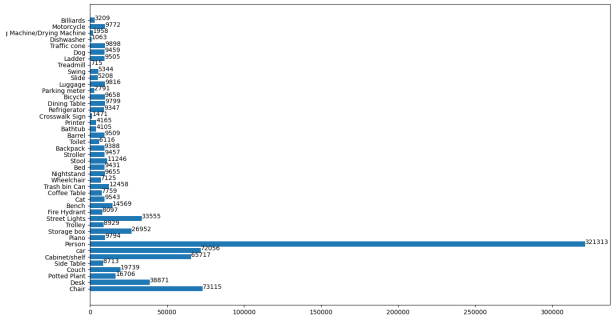

Fig. 2. Balanced Class Density Plot on the Selected 42 Classes

During each sampling, the dataset adjusts the sampling probability according to the class weights, ensuring that each batch has a balanced number of samples from all classes.

During training, we leveraged the automatic data augmentation features from the YOLO SDK, such as flipping, scaling, and cropping, to further balance the underrepresented classes and enhance their performance in the model.

D. Object Recognition Model Selection

In this project, we chose the YOLO series of models as the core framework for object detection. This decision was based on the widespread application of YOLO models, their technical advantages, and the continuous improvements made in object detection tasks.

YOLO is one of the most popular object detection models today, offering a rich set of development tools and SDKs that enable efficient model integration and deployment. Its single-stage detector architecture significantly enhances detection speed while maintaining high accuracy, making it particularly suitable for real-time scenarios.

Among the many YOLO versions, we selected the latest YOLOv11 model. Compared to YOLOv8, YOLOv11 improves accuracy by approximately 3% while reducing computational requirements by 20%, thus optimizing both precision and efficiency [3]. Additionally, YOLOv11 is one of the mainstream object detection models, and many hardware devices (such as TPUs and GPUs) provide robust optimization support for it. This greatly facilitates the subsequent deployment of the model, especially on edge computing devices.

We ultimately chose the YOLOv11-nano and YOLOv11-s models for training. Based on the hardware platform used in this project—Raspberry Pi with a Hailo-8L TPU, offering 13 TOPS of computational power—these two models are the most suitable choice to meet the real-time detection requirement (processing 30 frames per second). Moreover, by comparing the processing speed and accuracy of the two versions, we can provide data support for future model optimization and deployment.

YOLOv11 continues the overall framework of the YOLO series, comprising three main modules: Backbone, Neck, and

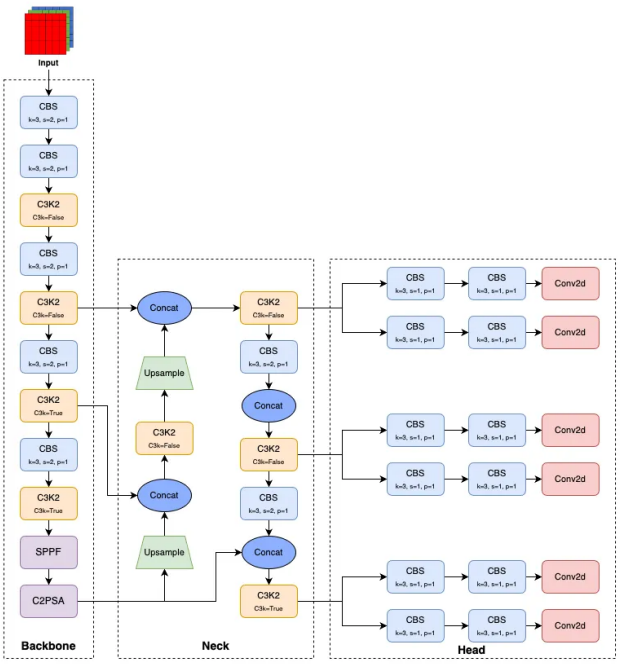

Fig. 3. YOLO11 Architecture [3]

Head, shown in Fig 3, which handle feature extraction, feature fusion, and object detection tasks, respectively:

- **Backbone:** YOLOv11 uses an improved CSPDarknet backbone network. By introducing the Cross Stage Partial (CSP) structure, redundant computations are effectively reduced, improving the model's inference speed. The Nano and S versions have significant differences in the depth and width of the Backbone, with the Nano version using lighter layers and channel designs to suit low-computation devices.
- **Neck:** YOLOv11 utilizes PANet (Path Aggregation Network) as the feature fusion layer, further optimizing detection performance for multi-scale features. The Spatial Pyramid Pooling (SPP) module enhances the network's ability to detect both large and small objects.
- **Head:** YOLOv11's output layer supports auto-anchor, which automatically adapts to objects of different sizes, making training more convenient.

E. YOLOv11 Model Training

In this project, we trained both the YOLOv11-nano and YOLOv11-s model versions separately to evaluate their performance across different scenarios. The entire training process strictly followed the sub-dataset division and parameter optimization strategies to ensure model stability and generalization capability.

As mentioned in Section II-C, we constructed two sub-datasets using a balanced subset approach: a subset with a maximum of 10,000 samples per class for training the YOLOv11-s model, and a subset with a maximum of 30,000 samples per class for training the YOLOv11-nano model.

For both rounds of training, the training set, validation set, and test set accounted for 70%, 15%, and 15%, respectively. This division ensures a sufficient number of training samples while providing a reliable foundation for model performance validation and testing.

The following hyperparameter configurations were uniformly used in both rounds of training:

- **Initial Learning Rate:** 0.01, adjusted empirically to balance model accuracy and training speed.
- **Batch Size:** 128.
- **Optimizer:** The default optimizer configuration from the YOLO SDK.
- **Training Method:** Mixed precision training using FP16 to reduce memory usage and accelerate computation.
- **Data Augmentation:** The automatic image enhancement functionality provided by the YOLO SDK was enabled, including random cropping, horizontal flipping, and brightness adjustment.

Besides, to prevent model overfitting, we set up an early stopping mechanism. If the performance on the validation set does not improve within 20 training epochs, the training process will automatically stop. All key metrics during training (such as training loss, validation accuracy, etc.) were monitored in real-time through TensorBoard. The training was conducted on a platform equipped with an NVIDIA 4090 GPU, which has powerful computational capabilities and a large memory capacity, supporting the training of deep learning models on large-scale datasets. However, even with this hardware configuration, due to the complexity of the YOLOv11 model, the training speed for both models was relatively slow, approximately 3 iterations per second. This made the entire training process time-consuming and placed higher demands on parameter tuning, especially for subsets with a larger number of samples.

F. Stereo Camera Preparation

Fig. 4. Mounted Two Cameras

To achieve precise stereo vision distance measurement, we performed detailed intrinsic and extrinsic calibration for two Akyta 1080p cameras. The calibration process is divided into two parts: monocular camera calibration and stereo calibration. These steps, conducted through the combined use of OpenCV and MATLAB, ensure the accuracy of both intrinsic and extrinsic parameters.

The two cameras are mounted on the same horizontal rig via cold shoe interfaces (see Fig. 4), minimizing mechanical errors that could impact the subsequent distance measurement accuracy.

We first conducted monocular calibration for both the left and right cameras to obtain each camera's intrinsic parameters, including focal length, principal point position, and distortion coefficients. The specific steps are as follows:

1) **Generation of 3D coordinates for chessboard corners**: We generated the 3D coordinate matrix for the chessboard corners using NumPy's `mgrid` function. The chessboard grid size and total number of points were pre-set.
2) **Corner detection in images**: Using OpenCV's `findChessboardCorners` function, we detected the chessboard corners in images captured by the cameras. If corner detection was successful, we refined the corner locations at the subpixel level using `cornerSubPix`.
3) **Calculation of camera intrinsic parameters**: We input the detected corner points and their corresponding 3D coordinates into OpenCV's `calibrateCamera` function, which calculated each camera's intrinsic matrix and distortion coefficients. The calibration results include:
   - **Intrinsic matrix**: Describes the camera's focal length and principal point offset.
   - **Distortion coefficients**: Describe the lens distortion level of the camera.

After completing monocular calibration, we performed stereo calibration to obtain the extrinsic parameters for the two cameras, including the rotation matrix and translation vector. The specific steps are as follows:

1) **Simultaneous image capture and corner detection**: Both cameras simultaneously captured chessboard images, and corner detection was performed on each image.
2) **Stereo calibration**: The corner points and their corresponding 3D coordinates were input into OpenCV's `stereoCalibrate` function, which computed the rotation matrix, translation vector, essential matrix, and fundamental matrix.
3) **Image rectification**: Using the calibration results, we rectified the images to perform epipolar correction, ensuring the views from both cameras were aligned along the same epipolar line.

Furthermore, we found that MATLAB's Stereo Camera Calibration App provides more accurate results for camera calibration. Compared to OpenCV, MATLAB offers more intuitive tools for evaluating calibration quality, especially the visualization of reprojection errors (see Fig. 5 and Fig. 6). Here is some interpretations of Figure 6:

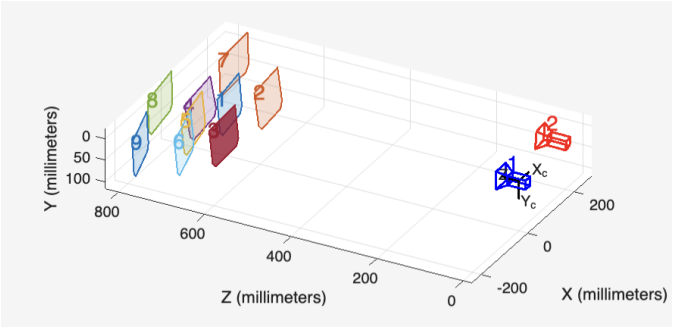

Fig. 5. Projection Angles Showing in MatLab Stereo Calibration Program

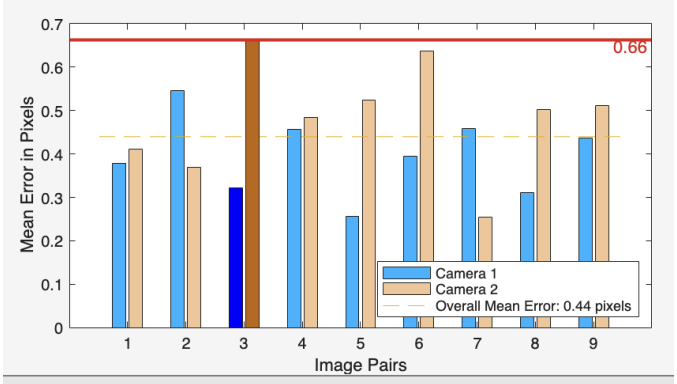

Fig. 6. Reprojection Errors Showing in MatLab Stereo Calibration Program

- **Reprojection error significance**: Reprojection error is an important indicator of calibration accuracy, representing the pixel deviation between the actual detected chessboard points and the reprojected points calculated using the intrinsic and extrinsic parameters. The lower the error, the more accurate the calibration result.

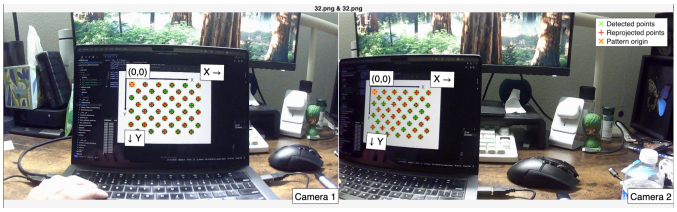

Fig. 7. Before Calibration

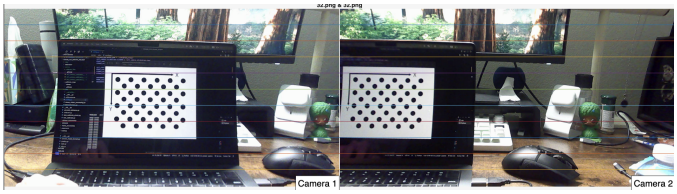

Fig. 8. After Calibration

- **Calibration strategy**: We set the reprojection error threshold (the red line in Fig. 6) to 0.5 pixels and continually captured new images to ensure the calibration images met the quality requirements. The final calibration results had an average reprojection error of 0.44 pixels.

Through the above steps, we obtained the following calibration results:

1) **Intrinsic matrix**: The focal length and principal point position for each camera.
2) **Extrinsic matrix**: Describes the relative position and orientation between the two cameras.
3) **Rectified images**: The rectified images (see Fig. 7 and Fig. 8) show consistent viewpoints and epipolar alignment after distortion correction, providing accurate input for subsequent depth calculation.

G. Depth Estimation Algorithm Selection

In this project, we chose the Semi-Global Block Matching (SGBM) algorithm to address the problem of stereo vision depth estimation. [4] This choice was based on SGBM's advantage in balancing real-time performance and accuracy, as well as its computational resource requirements, which align with the characteristics of low-cost edge computing platforms.

Compared to traditional stereo vision algorithms such as triangulation and SAD, as well as feature-based algorithms like SIFT matching, SGBM offers higher reliability and stability. Moreover, when compared to deep learning-based methods such as Marigold, SGBM demands less computational power, making it a suitable choice for systems deployed on low-cost edge computing devices.

H. Image Pre-processing for SGBM

Before inputting images into the SGBM algorithm, we performed multiple preprocessing steps to enhance matching accuracy and the robustness of the algorithm. First, we used OpenCV's `initUndistortRectifyMap` method, along with previously obtained intrinsic and extrinsic calibration parameters, to compute the rectification mapping parameters and the Q matrix. Epipolar rectification aligns corresponding points of the same scene in both images to the same horizontal line.

Next, we converted the images from the left and right cameras into grayscale, applied histogram equalization to improve contrast, used a bilateral filter to reduce noise while preserving edges, and undistorted the images based on the pre-computed camera intrinsic parameters to eliminate lens distortion effects.

Through these steps, we generated a set of optimized input images, significantly improving the matching performance of the SGBM algorithm.

I. SGBM Parameter Tuning

For the SGBM algorithm, we experimented with two key parameters: block size and the number of disparities (`numDisparities`). Increasing these values improves the matching accuracy but also slows down the process. After several iterations, we selected a block size of 5 and `numDisparities` of 128. These parameters were chosen to optimize performance based on the computational resources of the platform and the desired depth estimation accuracy.

J. SGBM Algorithm Workflow

1) **Disparity Calculation:** After rectifying the input stereo images, SGBM calculates the disparity for each pixel by sliding a matching window across the images.
2) **Semi-Global Optimization:** SGBM aggregates matching costs from multiple directions, reducing the blurring effects of local windows.
   - *Matching Cost:* The cost of matching pixel p and disparity d is calculated using the following formula:
     $$C(p,d) = |I_l(p) - I_r(p-d)|$$
     where $C(p,d)$ is the matching cost, I_l and I_r are the left and right images, respectively, and d is the disparity value.
   - *Cost Aggregation:* The aggregated cost is computed as:
     $$S(p,d) = C(p,d) + \min(S(p',d), P_1, P_2)$$
     where P_1 and P_2 are the smoothness and jump penalties, respectively.
3) **Disparity Map Generation:** The output is a disparity map representing the disparity value for each pixel.

K. Distance Estimation

Using the disparity map generated by SGBM at the last step, we convert it into a 3D point cloud using OpenCV's `cv2.reprojectImageTo3D` function. The process involves the following steps:

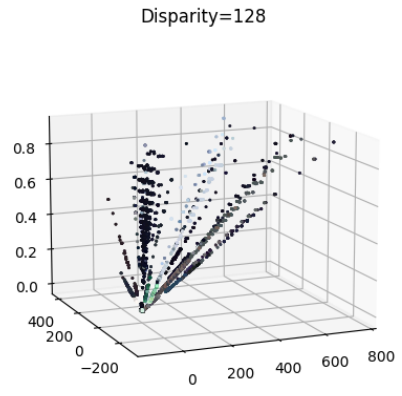

Fig. 9. An Example 3D Projected Disparity Map

1) **Mapping Disparity to 3D Points:** The disparity values are mapped to 3D coordinates in the camera's coordinate system using the following equation 1:

$$\mathbf{X} = Q \cdot \mathbf{d} \tag{1}$$

where Q is the projection matrix, and $\mathbf{d}$ is the disparity.

2) **Euclidean Distance Calculation:** The Euclidean distance D of each point is calculated as equation 2:

$$D = \sqrt{x^2 + y^2 + z^2} \tag{2}$$

Outliers, such as points with zero disparity or those affected by noise, are filtered out during this process.

Fig 9 is an example of a disparity map generated from the algorithm. Each point represent a obstacle detected from the camera.

L. Optimization of Distance Estimation Stability and Speed

In practical testing, we found that although the SGBM algorithm's distance estimation accuracy and robustness met basic requirements, there were challenges in terms of real-time performance and distance stability. Therefore, we optimized the algorithm from two aspects: distance stability and speed, which significantly improved the system's overall performance.

1) Distance Stability Optimization: In the initial design, we chose the depth value of the center point of the object's bounding box as the distance to the object. However, during testing, we observed that due to calibration errors, lack of texture, or local inconsistencies in the disparity map output by SGBM, the depth information of the center point might be missing in certain frames, causing the distance result to fluctuate to extreme values. Furthermore, due to the irregular shape of objects, the center point of the bounding box may not accurately reflect the object's true location. To address these issues, we uniformly sampled multiple depth points within the bounding box to avoid uncertainties from a single pixel. For these measurement points, we set a depth threshold to discard obvious extreme values, and then averaged the valid depth points to obtain the final distance estimate.

This approach significantly improved the stability of the distance estimation, reducing errors caused by local anomalies, and ensured the smoothness and reliability of the results.

2) Distance Speed Optimization: To improve the speed of distance estimation, we implemented several optimizations to reduce computational workload, including calculation reuse, input image cropping, and resolution reduction:

1) **Disparity Map Reuse:**
   - In the original implementation, each bounding box triggered a separate SGBM disparity map calculation. However, in reality, all target objects within the same frame share the same disparity map.
   - We decoupled the disparity map calculation from the bounding box processing, ensuring that only one SGBM computation is performed per frame, and the result is used for all detected targets.
2) **ROI Restriction:**
   - The computational load of the SGBM algorithm is proportional to the size of the input image. Using the detection results from YOLO, we limited the distance estimation to the target bounding box areas.
   - Vertical Direction: Since epipolar rectification has been completed, the bounding box range in the vertical direction for both cameras is consistent, so we can directly crop out the areas outside the bounding box.
   - Horizontal Direction: The position of the target object in the right camera is slightly shifted to the left compared to the left camera, so we can crop the right side of the right camera's bounding box.
   - Finally, we generated the ROI based on the cropped range of the right camera and ensured the corresponding ROI in the left camera.
3) **Resolution Downsampling:**
   - After determining the ROI, we downsampled the image using OpenCV's `pyrDown` method to reduce the resolution, thus further reducing the computational workload. The downsampled image still provides sufficient detail to meet the distance estimation requirements.
4) **Post-Processing Restoration:**
   - After completing the SGBM computation, we restored the disparity map to its original resolution via resizing and interpolation, while also restoring the cropped areas with padding to ensure the distance estimation results align correctly with the target positions in the original image.

The optimized system performance shows a significant improvement over the initial implementation: the per-frame distance estimation time was reduced from 600ms to 60ms, a tenfold speed-up. The optimized system achieves a real-time performance close to 16 FPS, meeting the real-time requirements for mobile scenarios.

M. Text-to-speech Implementation

After detecting all the objects within the views of cameras and their distances, to prevent overwhelming the user with too much information at once, we chose to prioritize reading the object closest to the user. This object is most likely the one the visually impaired user will encounter first. The system employs Text-to-speech technology using pyttsx3 to provide real-time audio notifications about obstacle information, promptly alerting users to avoid potential hazards.

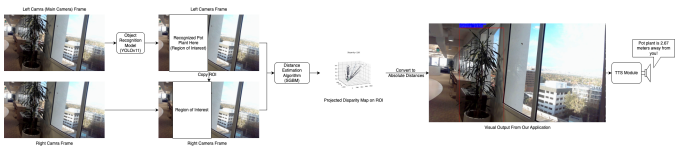

Fig. 10. Overall Workflow of Our Application

N. Overall Application Workflow

Fig. 10 summarizes and illustrates the overall workflow of our application.

III. Result and Discussion

In this section, we are going to discuss the results of the object detection model we trained and distance estimation algorithm we implemented.

A. YOLOv11 Model Performance

The evaluation results on the test set show that the YOLO11-s model achieves an overall performance metric of mP@50 = 0.562 and mAP@50:95 = 0.408.

From the Table II, we can observe that the performance metrics vary across categories. For example, the "Person" category has a higher detection accuracy (mAP@50 = 0.67), while certain small-object categories such as "Dishwasher" and "Slide" show lower detection performance.

TABLE II
Evaluation Results for Different Classes

Class	Images	Instances	Box(P)	R	mAP@50	mAP@50-95
all	16372	147703	0.648	0.527	0.562	0.408
Person	10820	52160	0.811	0.535	0.67	0.47
Trash bin Can	1421	2060	0.547	0.355	0.381	0.275
Chair	4640	11830	0.671	0.387	0.48	0.348
Wheelchair	731	1068	0.736	0.655	0.705	0.477
Desk	3837	6233	0.606	0.344	0.419	0.298
car	3014	11758	0.755	0.53	0.621	0.431
Storage box	1835	4374	0.46	0.12	0.169	0.11
Barrel	666	1445	0.488	0.248	0.267	0.194
Street Lights	1889	5299	0.673	0.368	0.448	0.27
Stroller	1386	1629	0.712	0.54	0.623	0.426
Stool	1175	1782	0.537	0.352	0.373	0.293
Cabinet/shelf	4146	10390	0.607	0.519	0.555	0.397
Potted Plant	1331	2503	0.595	0.374	0.429	0.28
Trolley	712	1378	0.596	0.414	0.455	0.302
Motorcycle	332	1477	0.664	0.468	0.54	0.331
Ladder	1156	1426	0.708	0.569	0.612	0.433
Bicycle	713	1549	0.681	0.466	0.535	0.337
Bench	1355	2299	0.519	0.198	0.231	0.157
Dog	559	1445	0.689	0.525	0.599	0.44
Bathtub	594	617	0.736	0.809	0.819	0.672
Toilet	874	917	0.834	0.883	0.915	0.826
Luggage	952	1494	0.582	0.428	0.455	0.289
Backpack	1115	1641	0.53	0.162	0.186	0.102
Bed	1136	1539	0.793	0.732	0.779	0.635
Dining Table	562	1469	0.515	0.378	0.382	0.251
Nightstand	770	1444	0.836	0.882	0.896	0.691
Couch	1915	3055	0.644	0.702	0.709	0.567
Refrigerator	1334	1434	0.744	0.728	0.786	0.642
Traffic cone	318	1488	0.733	0.513	0.569	0.359
Piano	1303	1471	0.637	0.457	0.529	0.353
Side Table	1070	1314	0.554	0.534	0.504	0.378
Coffee Table	1039	1163	0.596	0.632	0.596	0.492
Parking meter	288	418	0.616	0.435	0.449	0.309
Cat	1164	1432	0.816	0.8	0.844	0.694
Fire Hydrant	1176	1214	0.841	0.641	0.736	0.511
Crosswalk Sign	172	220	0.542	0.645	0.583	0.358
Printer	550	624	0.646	0.657	0.698	0.499
Billiards	138	484	0.669	0.655	0.653	0.446
Swing	450	801	0.677	0.62	0.672	0.473
Dishwasher	159	159	0.331	0.723	0.547	0.472
Washing Machine	172	293	0.656	0.696	0.702	0.557
Slide	560	793	0.656	0.537	0.585	0.4
Treadmill	51	114	0.616	0.447	0.444	0.318

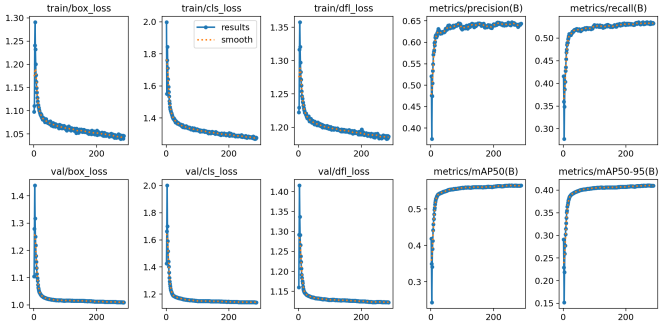

Fig. 11. Results in Training and Validation Dataset

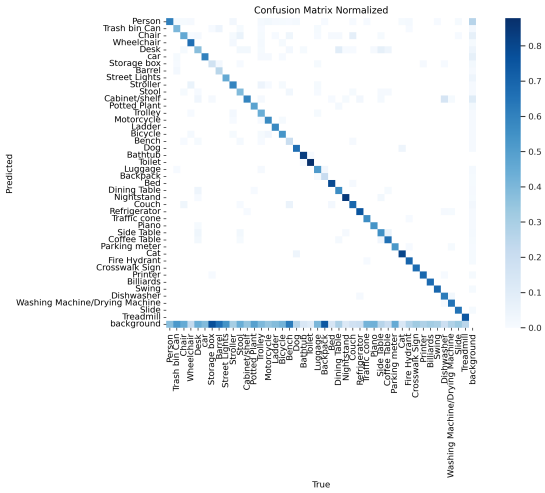

Fig. 12. Confusion Matrix

This aligns with the distribution characteristics of the sample sizes in the dataset, where categories with relatively large sample sizes and typically larger objects in the images tend to have better training results. On the other hand, categories with limited sample sizes, smaller object sizes, and greater susceptibility to occlusion have poorer training results.

The training process concluded at epoch 287 due to early stopping. Both the training and validation loss curves demonstrate a stable decreasing trend (see Fig. 11). Specifically:

- **Box Loss:** The box loss gradually converged to 1.06 for the training set and 1.02 for the validation set.
- **Class Loss:** The class loss rapidly decreased from an initial 1.38 to 1.14.
- **DFL Loss:** The DFL loss showed consistent behavior on both the training and validation sets, converging to 1.19 and 1.12, respectively.

At the same time, accuracy metrics such as mAP and Recall steadily increased throughout the training process, indicating that the model gradually learned effective feature representations.

Fig. 12 displays the normalized confusion matrix results for each class. The predictions for most categories are concentrated along the diagonal, indicating that the model overall ex-

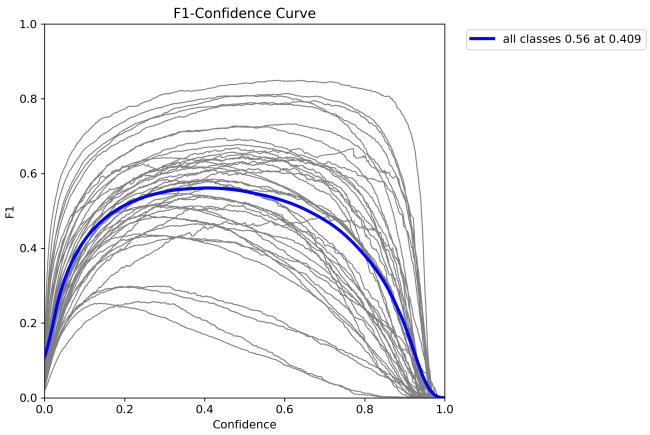

Fig. 13. F1 Curve

hibits high classification accuracy. However, some categories show confusion between one another, such as:

- **"Chair"** and **"Desk"**.
- **"Bicycle"** and **"Motorcycle"**.

These phenomena may be attributed to the visual similarities between these categories.

As shown in Figure 13, the F1-confidence curve demonstrates the model's performance at various confidence thresholds. The overall trend indicates that selecting a confidence threshold of 0.409 yields the best F1 score (F1 = 0.56). This result provides a reference for setting the confidence threshold during the model deployment phase.

B. Distance Estimation Algorithm Performance

In our testing across various scenarios, we found that the distance estimation met the needs of our project. Specifically, the system was able to correctly identify the relative distances: far objects were correctly identified as far away, and closer objects were correctly identified as being nearer. And the order of magnitude for absolute distance (between camera and the object) was definitely correct (see Fig. 14). These distances was sufficient for our system to find the nearest obstacle. Besides, We also conducted real-time tests [5], where the system maintained a frame rate of around 20 FPS, ensuring that the distance estimation could be used in dynamic, real-time environments.

Despite this, we made significant efforts to optimize the system for better accuracy across different conditions. Extensive camera calibration was performed, but we observed that the Mean Square Pixel (MSP) error between the two cameras was higher than expected, as shown in Fig. 6. This indicated that the intrinsic parameters of the cameras, such as focal length, lens distortion, and alignment, might not have been accurate enough. Although we carefully calibrated the cameras, the hardware's suboptimal quality and potential misalignment seemed to limit the accuracy of the distance estimation.

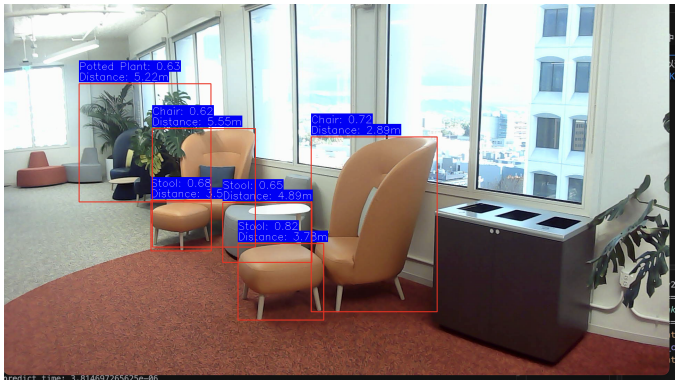

Fig. 14. Example Visual Outputs from Our Program

We recognize that improving measurement accuracy requires high-quality, well-calibrated cameras. While we took every possible step to mitigate calibration errors, the performance of the distance estimation algorithm was ultimately constrained by the limitations of the hardware. Without access to higher-quality camera setups or more advanced calibration techniques, achieving the desired level of absolute distance accuracy was challenging. Nevertheless, the system still demonstrated promising results in terms of relative distance estimation. We believe that with improved hardware or further refinement of calibration methods, the accuracy of the distance estimates could be significantly enhanced.

IV. Conclusion and Future Works

Although the core functionalities of this project have been successfully implemented and the expected goals have been achieved, there are still many unfinished tasks and potential directions for future expansion. The following are several key areas for future work:

A. Deployment on Edge Computing Platforms

Currently, the system has not yet been deployed on the Raspberry Pi 5 and Hailo 8L TPU combination platform. As a low-cost edge computing device, deploying on the Raspberry Pi is a critical step for achieving system lightweight and portability. However, the model translation for the Hailo 8L TPU poses a significant technical challenge. We plan to further compress the model size and reduce computational resource requirements through int8 quantization and half-precision (FP16). The optimized model will be translated into the .har format supported by Hailo TPU, followed by deployment and testing on the Raspberry Pi.

B. Improvement of Distance Estimation Accuracy

The current system's distance estimation accuracy is still limited by the quality of the cameras used, which suffer from significant distortion and jelly effect, along with low frame rates and image quality. These factors directly affect calibration accuracy and the stability of depth estimation. Additionally, the current reprojection error is still difficult to reduce below 0.5. We plan to use higher-performance and

more stable cameras to reduce distortion and jelly effects, improve image quality and frame rates, and attempt to reduce the reprojection error to below 0.1. Furthermore, we aim to validate the accuracy and robustness of SGBM's distance estimation.

C. Introduction of Semantic Segmentation

Currently, the system's object detection based on bounding boxes has certain limitations when dealing with irregularly shaped objects. The distance points for complex-shaped obstacles may deviate from the object's surface, leading to inaccurate depth values.

In the future, we plan to explore object detection methods based on semantic segmentation to more accurately define object boundaries and surface areas. However, semantic segmentation models typically require higher computational power. Therefore, we will need to balance the improvement in accuracy with the computational demands. Additionally, the feasibility of integrating semantic segmentation into the existing framework and its impact on performance will also be an important area of research.

REFERENCES

[1] Apple Inc., "Accessibility," 2024, Apple. [Online]. Available: https://www.apple.com/accessibility/

[2] S. Shao et al., "Objects365: A Large-Scale, High-Quality Dataset for Object Detection," 2019 IEEE/CVF International Conference on Computer Vision (ICCV), Seoul, Korea (South), 2019, pp. 8429–8438

[3] N. Jegham, C. Y. Koh, M. Abdelatti, and A. Hendawi, "Evaluating the Evolution of YOLO (You Only Look Once) Models: A Comprehensive Benchmark Study of YOLO11 and Its Predecessors," 2024, arXiv:2411.00201. [Online]. Available: https://arxiv.org/abs/2411.00201

[4] H. Hirschmuller, "Accurate and efficient stereo processing by semi-global matching and mutual information," 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), San Diego, CA, USA, Jun. 2005, pp. 807-814. DOI: 10.1109/CVPR.2005.56.

[5] Uni 404, "CS5330 Group5 Final Proj Demo," YouTube, Nov. 23, 2024. https://www.youtube.com/watch?v=9njp5Nq8DAc