Autonomous navigation for blind on Jetson Nano

STAT5205.2025Spring.final report.yh3719
Yilong Huang yh3719
Columbia University

Abstract

This project presents a real-time navigation aid for visually impaired individuals, leveraging a YOLOv5n-seg model deployed on an NVIDIA Jetson Nano 2GB. The primary goal was to achieve efficient semantic segmentation of walkable areas, obstacles, and humans from a first-person camera view, enabling safer and more independent navigation. Key challenges included the scarcity of suitable first-person segmentation datasets, necessitating the creation of a custom dataset of approximately 3000 annotated frames, and the significant computational and memory constraints of the target embedded platform. These were addressed by employing the lightweight YOLOv5n-seg architecture, optimizing the model with TensorRT using FP16 precision, and meticulously configuring the Jetson Nano environment. The system successfully achieved real-time inference at approximately 12-14 FPS with a mean Average Precision (mAP@0.5) of 0.952 post-optimization, demonstrating the feasibility of deploying advanced visual perception models on low-cost, resource-constrained hardware for assistive technology. The original goal of developing a real-time navigation aid was accomplished.

1. Introduction

Visual impairment affects a significant portion of the global population. According to the World Health Organization (WHO), at least 2.2 billion people suffer from vision impairment worldwide [1]. Many could benefit from a portable system that assists safe navigation in outdoor environments. This project aims to develop a low-cost, real-time navigation aid using computer vision deployed on embedded hardware.

Deep learning has greatly advanced visual scene understanding, particularly in object detection and semantic segmentation. While object detection localizes discrete objects via bounding boxes, semantic segmentation provides pixel-level classification, offering a detailed map of the environment. For pedestrian navigation, pixel-wise understanding is crucial to distinguishing walkable areas from obstacles [2]. Recent models like YOLOv5-seg integrate detection and segmentation into a single, efficient network suitable for real-time applications [3].

This project focuses on detecting walkable paths and obstacles from a first-person perspective using a chest-mounted monocular camera. The technical objective is to achieve real-time inference ($\geq$10 FPS) with high segmentation accuracy for critical classes: walkable area, obstacle, and pedestrian. Deployment targets the NVIDIA Jetson Nano 2GB, whose limited GPU memory and compute capacity necessitate lightweight models and optimization for embedded inference within a 5–10 W power envelope.

Given these constraints, YOLOv5n-seg was selected as the core model. As a compact variant of the YOLOv5 family, YOLOv5n-seg just has 2.6M parameters, which enables efficient segmentation with minimal resource usage [3]. Coupled with TensorRT optimization [4], it achieves real-time semantic scene interpretation on the Jetson Nano, aligning with the project's goals of balancing speed, accuracy, and power efficiency for assistive navigation.

2. Summary of the Related Work

This work is situated within the context of existing research on real-time object detection and semantic segmentation, leveraging publicly available implementations. The primary technical foundation is the YOLOv5 model series developed by Ultralytics [3]. YOLOv5 comprises a family of lightweight and efficient convolutional neural network (CNN) architectures optimized for real-time object detection. Subsequent extensions, notably YOLOv5-seg, incorporate a segmentation head, enabling simultaneous object detection and instance segmentation [3], a significant advancement over earlier detection-only models.

The 'nano' variant, YOLOv5n-seg, is specifically optimized for edge deployment due to its reduced model size and computational complexity. This optimization allows it to maintain robust segmentation performance even with constrained memory and computational resources, rendering it highly suitable for platforms such as the Jetson Nano.

Prior projects, such as the DeepWay project [5] and the Outdoor Blind Navigation project [6], have established the feasibility of employing deep learning for assistive navigation systems. These systems, however, often utilized more computationally intensive networks or multi-model pipelines, posing challenges for real-time inference on Jetson Nano with just 2GB memory.

Consequently, in this project, the YOLOv5n-seg model is adopted as the core component for semantic scene understanding. The proposed methodology distinguishes itself by its reliance on a single, compact segmentation model for end-to-end perception and navigational decision-making, specifically tailored for operation under stringent computational constraints.

3. Methodology

3.1. Objectives and Technical Challenges

The development of a real-time navigation aid on the Jetson Nano 2GB necessitated a core design choice: utilizing a single, unified segmentation model (YOLOv5n-seg) to minimize resource consumption. This decision, informed by prior experience, led to several technical challenges:

- Dataset Scarcity: The absence of suitable first-person segmentation datasets for outdoor navigation required the manual collection and annotation of approximately 3000 frames from a chest-mounted camera to capture a relevant user perspective.
- Hardware Limitations: The Jetson Nano's restricted GPU memory and processing power constrained model size and inference batch capabilities. Overcoming this involved careful optimization, primarily through 16-bit half-precision (FP16) inference with TensorRT, and selection of the lightweight YOLOv5n architecture. Thermal throttling and power envelope considerations also influenced the design for sustained operation.
- Deployment Complexity: Deploying the model on the Jetson Nano introduced software hurdles. These included ensuring python version and library compatibility (PyTorch, CUDA, TensorRT), managing insufficient swap memory during model conversion, optimizing the inference pipeline to prevent CPU bottlenecks, and successfully converting and running the YOLOv5n-seg model with NVIDIA TensorRT.

These intertwined objectives and challenges guided the project's focus on a streamlined, single-model architecture to achieve the goal of real-time, on-device visual assistance.

3.2.Problem Formulation and Design Description

The model takes live video frames from a chest-mounted USB camera as input. The output is a segmented view identifying walkable areas (free path) versus obstacles, which can then be used to alert the user. Core processing occurs on the Jetson Nano, which runs the neural network to analyze each frame.

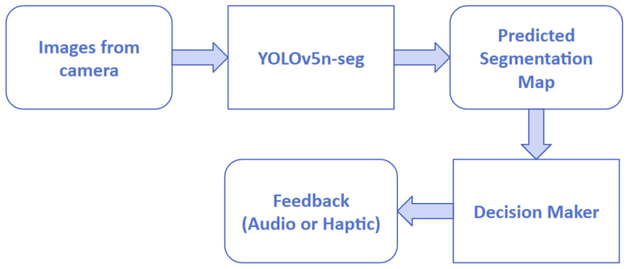


Figure1. A system block diagram

The navigation problem is formulated as a segmentation task where the model categorizes pixels as "walkable_area," "obstacle," or "human". "Walkable_area" represents safe ground (e.g., sidewalk, path), "pedestrian" identifies other people as moving obstacles, and "obstacle" covers static impediments (e.g., walls, trash bins, vehicles). This segmentation approach directly maps camera pixels to navigability, which is intuitive for path planning.

Rule-based decision logic is then applied to the segmentation output to assist the user. After the YOLOv5 produces a segmented image, the system interprets the mask using a simple decision tree. For example, if the walkable_area directly in front of the user is continuous and clear, no action or gentle positive feedback is given. If an obstacle or human appears directly ahead within a certain pixel-defined distance, a warning is flagged. Logical checks (e.g., "The camera view is divided into a 3×3 grid, is the bottom-center cell entirely clear and walkable?" and "is a pedestrian detected in the center of the path?") determine the guidance, translating raw segmentation into actionable advice.

Consider a scenario: a user is walking on a sidewalk (segmented as green walkable area). If a human (segmented yellow) crosses their path, taking up a significant area in the center, the decision logic would interpret this as an impending collision and might instruct the user to slow down or veer. Conversely, a pedestrian off to the side might be ignored. This illustrates how segmentation and decision rules collaboratively form a navigation aid.

This system design is well-suited for the problem because it directly addresses the spatial understanding needed by visually impaired users—identifying free space versus obstacles—with minimal hardware.

4. Implementation

4.1 Data

Approximately 3000 image frames were collected using a chest-mounted camera to represent a walking user's visual field. This involved recording short video clips in various outdoor environments (e.g., campus and streets). Efforts were made to ensure dataset diversity by capturing different scenarios: straight paths, turns, various obstacles (trash cans, bicycles), and pedestrians, to facilitate robust model learning.

The frames were annotated using the Computer Vision Annotation Tool (CVAT), an open-source web-based tool for image and video labeling [7]. Polygon masks were drawn for 3 classes: walkable area, human, and obstacles to be used for YOLO training.

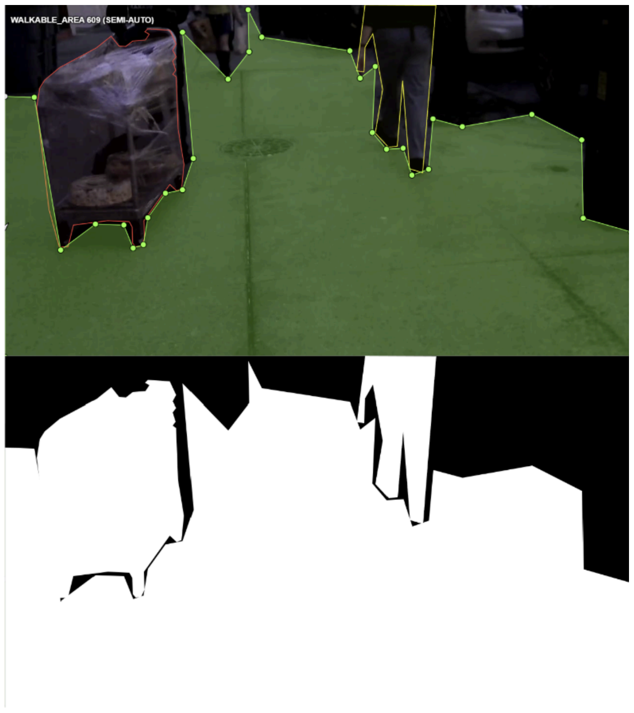


Figure 2. An example annotation frame of data collected on the street and its bitmap.

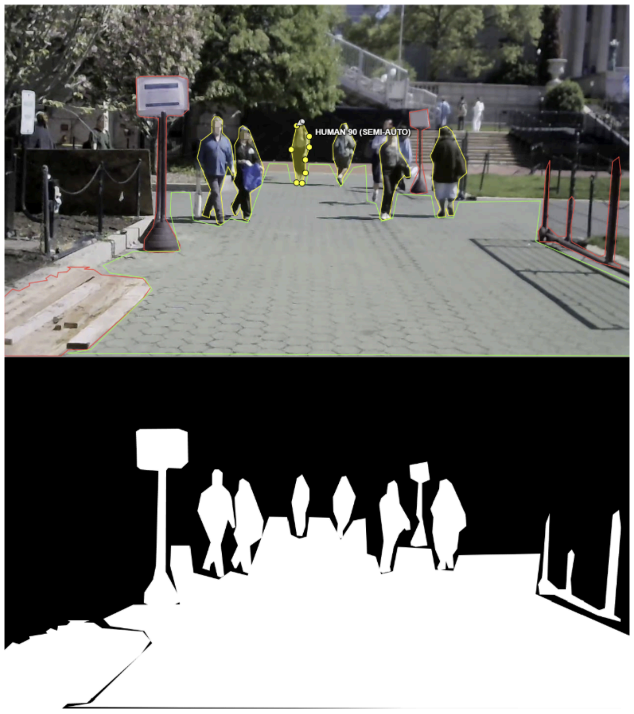


Figure 3. An example annotation frame of data collected on the campus and its bitmap.

Figure 2 and 3 illustrate an example annotation where the walkable area is shaded green, humans are outlined in yellow and the obstacles are annotated in red. This manual annotation produced ground truth masks for training the segmentation model.

The final dataset of approximately 3000 annotated images was split into 80% for training, 20% for validation during training. Data augmentation was not employed, as images exhibiting overexposure and underexposure had already been collected during the data acquisition process. Furthermore, a program is planned to be developed that will automatically restart the camera upon detection of overexposed or underexposed conditions, thereby enabling better adaptation to varying environmental lighting.

Challenges in labeling included the time-consuming and sometimes subjective nature of precisely outlining walkable areas, especially at boundaries. Despite its modest size, the custom dataset was tailored to the task, providing necessary examples for training a specialized model.

4.2 Model Selection and Training

YOLOv5n-seg, the nano variant of YOLOv5 with an instance segmentation head, was chosen for its lightweight nature (smallest in the YOLOv5 family) yet capability to segment multiple classes, offering a good balance between accuracy and speed for the limited computational budget. Using a single network for segmentation conserved memory by avoiding the overhead of separate networks.

Training was performed on a Google Cloud Platform with a Nvidia L4 using the PyTorch framework. Transfer learning from pretrained weights “YOLOv5n-seg.pt” (trained on COCO) has been used to accelerate convergence. Key training hyperparameters included image size (720 x 720 pixels), batch size (32), number of epochs (500), and optimizer settings. In addition, the number of classes was also adjusted to 3 from about 80 in the data.yaml and yolov5n-seg.yaml.

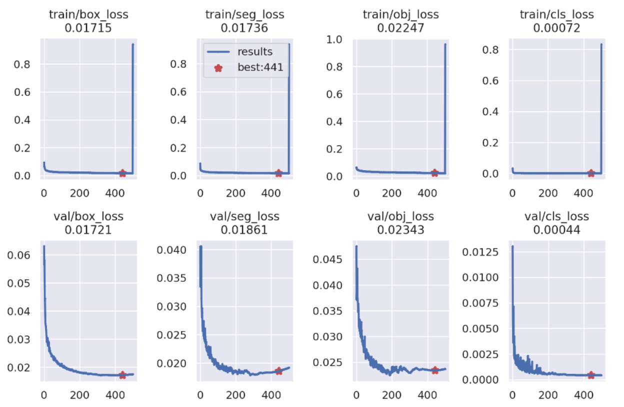


Figure 4. The training and validation loss curves on GCP.

Figure 4 shows that all loss components, including box loss, segmentation loss, objectness loss, and classification loss, converge smoothly without signs of overfitting. The final validation segmentation loss stabilizes at approximately 0.018 at about 400 epoch, confirming good generalization performance.

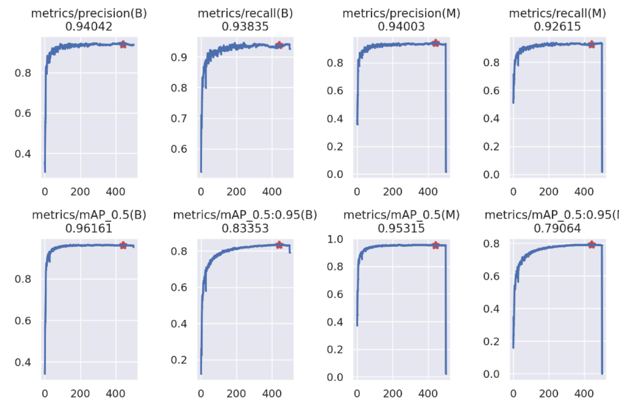


Figure 5. The precision, recall, and mean Average Precision (mAP) metrics during training and validation on GCP.

Figure 5 indicates that the model achieves a high detection and segmentation accuracy, with mAP@0.5 reaching 0.96 for the training set and 0.95 for the validation set.

4.3 Deployment on Jetson Nano

4.3.1 Conversion to TensorRT

After training the YOLOv5n-seg model in PyTorch, it was necessary to optimize the model for deployment on the Jetson Nano. The conversion process involved exporting the model to the ONNX (Open Neural Network Exchange) format, and subsequently using NVIDIA TensorRT to compile it into a highly efficient TensorRT engine suitable for inference.

TensorRT is an SDK designed to maximize inference performance on NVIDIA GPUs by applying optimizations such as layer fusion and precision reduction [4] . In this project, the model was converted to use FP16 (half-precision floating point) operations, reducing memory usage and accelerating inference on the Jetson Nano, which supports fast FP16 execution. Challenges encountered during conversion included ensuring that the custom segmentation outputs of YOLOv5n-seg were properly supported and selecting a TensorRT version compatible with the JetPack release installed on the Nano.

4.3.2 Jetson Nano Setup

Setting up the Jetson Nano for deploying the YOLOv5n-seg model required careful environment configuration due to software compatibility challenges. YOLOv5 requires Python version ≥3.8, but the default JetPack 4.6.1 release for the Jetson Nano 2GB only officially supports Python 3.6 through NVIDIA's provided L4T PyTorch container [8] . This discrepancy made it impossible to directly run the YOLOv5 inference scripts.

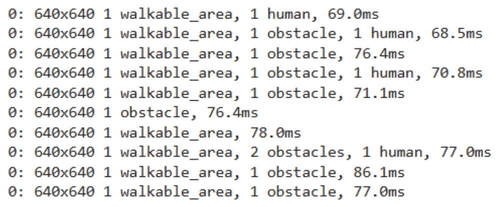
Initial attempts to resolve this involved consulting community resources such as [9], which suggested manually installing a custom Ubuntu 18.04/20.04 image with Python 3.8 support. However, mismatches between the manually installed Python environment, PyTorch versions, and the CUDA runtime caused further complications, leading to unstable builds and library incompatibility.

After further investigation, the Ultralytics official documentation [10] was consulted, which provided guidance on deploying YOLOv5 on NVIDIA Jetson devices. Following these recommendations, a compatible system image was located and downloaded, providing a properly integrated Python 3.8, PyTorch, and CUDA environment suitable for YOLOv5. Flashing this image onto the Jetson Nano and installing the necessary dependencies finally enabled successful deployment and real-time execution of the YOLOv5n-seg model.

This meticulous setup process was crucial for achieving stable, real-time segmentation performance on the resource-constrained embedded device.

4.3.3 Inference Result and Discussion

After deploying the .engine model to Jetson Nano, live inference tests are conducted on it. During inference, the TensorRT engine consumed a significant portion of the available 2GB RAM. FP16 optimization and efficient memory management prevented out-of-memory errors. CPU usage was moderate due to Python overhead but remained within operational limits. Visual outputs were displayed for debugging during development, although in a final wearable system, this would be replaced by haptic or auditory feedback.



```
0: 640x640 1 walkable_area, 1 human, 69.0ms
0: 640x640 1 walkable_area, 1 obstacle, 1 human, 68.5ms
0: 640x640 1 walkable_area, 1 obstacle, 76.4ms
0: 640x640 1 walkable_area, 1 obstacle, 1 human, 70.8ms
0: 640x640 1 walkable_area, 1 obstacle, 71.1ms
0: 640x640 1 obstacle, 76.4ms
0: 640x640 1 walkable_area, 78.0ms
0: 640x640 1 walkable_area, 2 obstacles, 1 human, 77.0ms
0: 640x640 1 walkable_area, 1 obstacle, 86.1ms
0: 640x640 1 walkable_area, 1 obstacle, 77.0ms
```

Figure 6. Screenshot of live inference output

Figure 6 shows an average processing time of approximately 70–80 milliseconds per frame, corresponding to about 12–14 frames per second (FPS). This result exceeded the initial real-time design goal (~10 FPS) and demonstrated smooth real-time performance during outdoor walking scenarios.

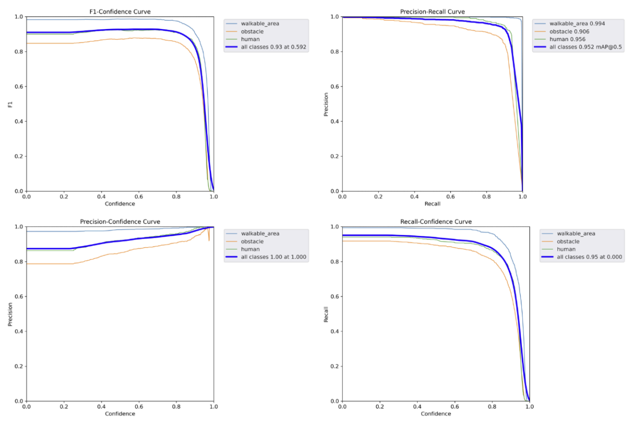


Figure 7. The metrics of validation set after TensorRT conversion on Jetson Nano.

The model was evaluated on the validation set both before and after TensorRT conversion. On GCP during training, the model achieved a mean Average Precision (mAP) of 0.953 at IoU 0.5. After TensorRT conversion on Jetson Nano, the mAP slightly dropped to 0.952, as shown in Figures 5 and 7.

Despite this minor degradation (~0.1% decrease in mAP@0.5), the model preserved strong precision and recall characteristics, demonstrating successful retention of segmentation quality after optimization. This verification step ensured that the optimization and conversion process did not introduce significant degradation in model performance.

The small drop in validation performance is acceptable given the significant reduction in memory footprint and runtime latency achieved through TensorRT optimization. Overall, the deployment successfully achieved the goals of real-time inference with acceptable accuracy loss, validating the feasibility of the approach.

4.3.4 Decision-Making Module

In addition to segmentation, a standalone decision-making module was developed to translate the model's output into real-time navigation commands. The script analyzes the segmented walkable area and detected obstacles to generate intuitive navigation instructions such as proceed straight, veer left/right, slow down, or stop.

The logic incorporates multiple spatial analysis steps, including: assessing the continuity and width of the walkable path, evaluating the position and proximity of obstacles relative to the user, scoring potential threats based on distance, size, and spatial relation to the walkway, prioritizing threats to decide the safest immediate action.

Navigation decisions are output in a structured format, including a severity warning level and a human-readable message.

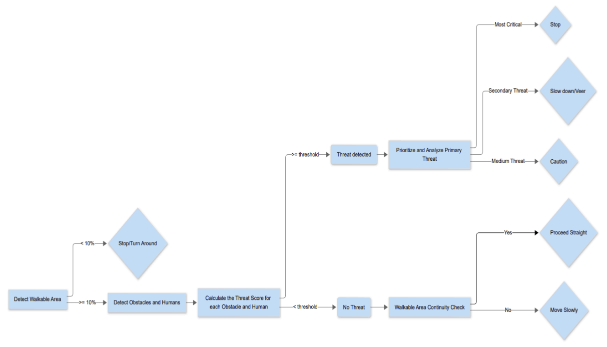


Figure 8. Flowchart of the Navigation Decision-Making Module.

Although the decision-making module was tested separately using offline simulation scenarios, it has not yet been fully integrated into the live inference pipeline on the Jetson Nano. Future work will involve tightly coupling this decision module with the real-time segmentation output to complete the assistive navigation loop.

5. Results

5.1 Project Results and Observations

To verify the system's real-world performance, real-time inference was conducted on a chest-mounted monocular camera setup using the Jetson Nano.

The visualizations confirm that the system can operate reliably in dynamic real-world conditions, offering accurate semantic understanding to support navigation.

Despite operating on a resource-constrained device, the segmentation quality remained high, and latency was sufficiently low for real-time application. Minor challenges, such as occasional misclassification of downward stairs or overexposed scenes, were observed and will be addressed in future improvements.

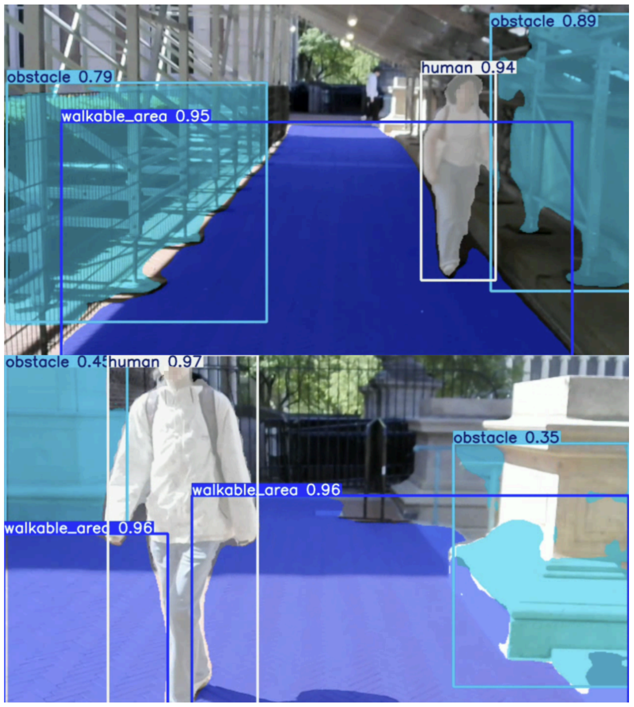


Figure 9. Screenshots of live inference output

Figure 9 shows successful segmentation of walkable areas (blue mask) and detection of obstacles and humans.

Additionally, a decision-making module was developed to translate segmentation outputs into navigation commands, which will be integrated into the real-time system in future work.

5.2 Comparison of the Results Between Projects

5.2.1 Comparison to DeepWay.v2

DeepWay.v2 [5] is an open-source navigation assistant for the visually impaired that also targets the Jetson Nano platform. It adopts a different architecture: training a U-Net for path segmentation (classifying the lane as left, center, or right) and supplementing it with additional networks such as MobileNet for object detection. However, DeepWay's multi-model architecture resulted in lower runtime performance, achieving only around 3–5 FPS when running the full pipeline on the Nano.

In contrast, the current project employed a single, efficient YOLOv5n-seg model, achieving approximately 11 FPS. This trade-off highlights that while DeepWay attempted richer scene analysis (lane-specific guidance and object classification), it came at the cost of real-time performance on constrained hardware.

Regarding datasets, DeepWay collected around 10,000 images to train its networks, while this project used a smaller dataset of about 3,000 images. Despite the smaller data size, leveraging a more modern multitask architecture (YOLOv5n-seg) enabled effective segmentation and obstacle detection with reasonable generalization. It is expected that expanding the dataset in future work could further improve accuracy.

5.2.2 Comparison to Outdoor-Blind-Navigation

The Outdoor-Blind-Navigation project [6] targeted a more powerful embedded platform, the Jetson Xavier NX. Their system used multiple networks in parallel: a YOLOv4-tiny model for object detection and two custom classifiers for sidewalk alignment and turning detection. Thanks to the Xavier NX's significantly greater computational capacity (more memory and more CUDA cores), they could concurrently run these models, likely achieving 15–30 FPS.

Compared to this, the current system is simpler, relying on a single model for immediate path segmentation and obstacle recognition.

Although SightWalk offers richer scene understanding (identifying cars, bicycles, signs, etc.), such a complex pipeline would not be feasible on the Nano without substantial sacrifices in speed or stability, and classifying different classes of static obstacles is meaningless to the blind navigation system.

5.3 Discussion

5.3.1 Benefits of the Segmentation-Only Approach

Using a single segmentation model, rather than multiple detection and segmentation networks, was a crucial design decision. This approach provided rich pixel-wise scene understanding at low computational cost. It also simplified the downstream decision-making logic, since walkable areas and obstacles were explicitly segmented. Running multiple networks would likely have caused unacceptable delays or memory overflow on the Nano. The success of the single-model approach underscores the value of multitask architectures like YOLOv5n-seg for resource-limited applications.

5.3.2 Limitations: Staircase Misclassification

One notable limitation observed was the occasional misclassification of staircases as walkable flat surfaces, especially when handling descending staircases. This likely occurred because the training dataset lacked specific annotations distinguishing stairs from normal ground. Since visual segmentation relies purely on pixel patterns, and stairs often resemble flat ground in color and texture, the model had no mechanism to differentiate them. This limitation highlights an important gap between segmentation output and navigational affordance, as stairs require different handling for safety.

5.3.3 System Reliability

Throughout testing, the system operated reliably without memory crashes or instability. Minor slowdowns were occasionally observed under prolonged use, but no critical failures occurred. Nevertheless, extended operation could introduce concerns related to power supply stability and thermal management, underscoring the importance of conducting long-term endurance testing. Overall, the system exhibits sufficient robustness for real-world prototyping.

6. Future Work

Building on the current prototype, several directions are planned to enhance the system's robustness, safety, and usability:

- **Improving Staircase Detection**:

Given the current model's occasional misclassification of stairs as walkable ground, an immediate priority is to expand the dataset to include explicit examples of stairways and steps. Adding a new class for "stairs" or labeling stairs as obstacles will help the model differentiate flat surfaces from elevation changes, reducing the risk of unsafe navigation. Alternatively, integrating a lightweight depth-sensing module could provide additional cues for elevation detection without significantly increasing computational load.

- **Refining the Decision Tree Logic**:

The initial decision logic, based on segmentation outputs, will be further refined to handle more complex navigation scenarios, such as: early warnings when obstacles are detected far ahead, dynamic adjustment of walking direction based on obstacle distribution and specialized handling for uncertain cases (e.g., partially obstructed walkable areas). The decision tree will evolve into a more adaptive system, possibly incorporating confidence scores from the model into navigation decisions.

- **Integration of Feedback Mechanisms**:

Future versions will connect the navigation decisions to real-time feedback systems, such as: audio alerts (e.g., earphones providing directional cues) and haptic feedback (e.g., vibration motors on a wearable device signaling direction or warnings). User-centered studies will be needed to calibrate the feedback timing and intensity to avoid overwhelming or distracting the user.

7. Conclusion

This project successfully developed and deployed a real-time visual navigation aid for the visually impaired on a Jetson Nano 2GB platform. The primary objective of segmenting walkable areas, obstacles, and humans in real-time using a single, efficient deep learning model was achieved. By training a YOLOv5n-seg model on a custom-collected dataset and optimizing it with TensorRT for FP16 inference, the system attained a processing speed of 12-14 FPS and a strong mAP@0.5 of 0.952 on the validation set after conversion, with minimal accuracy degradation compared to pre-conversion performance.

Key lessons learned underscore the efficacy of the single-model segmentation approach for resource-limited embedded applications, the critical role of dataset customization in specialized tasks, and the significant performance gains achievable through careful model optimization and environment configuration on edge devices. The project also highlighted challenges inherent in embedded AI deployment, such as software compatibility and meticulous setup procedures. While the system demonstrated robust performance, a notable limitation was the occasional misclassification of staircases as walkable surfaces, primarily due to their underrepresentation in the training data.

Future research will focus on improving staircase detection by expanding the dataset and potentially integrating lightweight depth sensing. Further refinement of the decision-making logic and integration of user feedback mechanisms, such as audio or haptic alerts, are also crucial next steps. This work validates the potential of leveraging compact, optimized deep learning models on affordable embedded systems to create practical and impactful assistive technologies.

8. Acknowledgement

I would like to thank Ultralytics for the YOLOv5 model series and the developers of the Computer Vision Annotation Tool (CVAT) for their open-source contributions. I also acknowledge NVIDIA for providing the Jetson development platform and the TensorRT™ software development kit. The public availability of projects such as DeepWay and SightWalk provided valuable context for this research.

Finally, my sincere thanks are extended to Jiangao Han for assistance during the data collection phase.

9. References

[1] World Health Organization, “World report on vision” 2019. [Online]. Available: https://www.who.int/publications/i/item/world-report-on-vision
[2] Y. Lei, S. L. Phung, A. Bouzerdoum, H. Thanh Le and K. Luu, "Pedestrian Lane Detection for Assistive Navigation of Vision-Impaired People: Survey and Experimental Evaluation," in IEEE Access, vol. 10, pp. 101071-101089, 2022, doi: 10.1109/ACCESS.2022.3208128.
[3] G. Jocher, "YOLOv5 by Ultralytics," version 7.0, May 29, 2020. [Online]. Available: https://github.com/ultralytics/yolov5. doi: 10.5281/zenodo.3908559.
[4] NVIDIA Corporation, TensorRT Documentation, Version 10.10.0, Apr. 2025. [Online]. Available: https://docs.nvidia.com/deeplearning/tensorrt/latest/index.html
[5] S. Singh, "DeepWay.v2: Autonomous Navigation for Blind People," GitHub repository, 2021. [Online]. Available: https://github.com/satinder147/DeepWay.v2
[6] A. Oberai et al., "SightWalk: Open-Source Sidewalk Navigation Software for Visually-Impaired Individuals using Multithreaded CNNs," GitHub repository, 2021. [Online]. Available: https://github.com/team8/outdoor-blind-navigation
[7] CVAT.ai Corporation, Computer Vision Annotation Tool (CVAT), Version 2.25.0, Nov. 6, 2023. [Online]. Available: https://www.cvat.ai/. doi: 10.5281/zenodo.4009388.
[8] NVIDIA, “L4T PyTorch Container,” [Online]. Available: https://catalog.ngc.nvidia.com/orgs/nvidia/containers/l4t-pytorch.
[9] Jetson Docs Community, “Python support on Jetson Nano,” [Online]. Available: https://jetson-docs.com/libraries/python/overview.
[10] Ultralytics, “Deploy YOLOv5 on NVIDIA Jetson,” [Online]. Available: https://docs.ultralytics.com/guides/nvidia-jetson/.

10. Appendix

10.1 Code Availability

The code and relevant outputs are in the github: https://github.com/Elon-H/STAT5205Final