VLM-Based Navigation for Autonomous Mobile Robot

Robin Rheem, Jeffrey Zhang, Kushagra Srivastava
roundrobin@berkeley.edu, jzh4ng@berkeley.edu, kushagras@berkeley.edu

ABSTRACT

Autonomous mobile robots operating in real-world environments must perceive obstacles, reason through multiple navigation options, and execute motion under computation and communication constraints. This project presents a hybrid perception and control system that integrates vision-language models, object detection, and geometric reasoning to identify and navigate through obstacles. Implemented on a Pololu 3pi+ 2040 robot with a Raspberry Pi Zero W and a camera, the system offloads perception to a remote server that performs semantic understanding of the scene with VLM reasoning and YOLO-based localization, while the robot executes motion commands using closed-loop feedback. By incorporating a vision-language model, the system gains semantic awareness, extending the system beyond purely geometric navigation–for example, enabling the robot to reason differently about visually distinct objects that may seem geometrically similar within a cluttered environment. The experimental results under some realistic latency constraints demonstrate a reliable gap selection at relatively low speeds, highlighting the key trade-offs between communication, responsiveness, and semantic reasoning for an embedded robotic navigation system.

1. Introduction

Robotic navigation in unstructured environments requires more than simple obstacle avoidance. The robot must continuously interpret its environment and choose an optimal solution from multiple possible paths, all while operating under limited computational resources. Classical navigational approaches tend to rely upon geometric representations (e.g., occupancy grids, range sensing, point clouds, etc.) to detect free space and avoid collisions. While this is effective in a controlled setting, these approaches treat all visible objects as equivalent geometric entities and thus lack the ability to reason about the nature of the environment.

Recent advancements in computer vision have enabled robots to perceive their surroundings with greater precision. Object detection models can localize obstacles in camera images, while vision language models (VLMs) can provide a higher-level semantic description of the scene. Such capabilities open a path to navigation systems that also reason about the obstacles themselves and the relationships between them. This semantic awareness is extremely valuable in real-world applications, where visual complexity, ambiguity, and variability is a common symptom.

Through this project, we explore a hybrid approach that combines semantic environment understanding from a VLM with the precise geometric localization from an object detection model. This objective mimics real-world scenarios in which a robot must choose amongst several potential paths rather than simply avoiding collisions. By implementing this system on a resource-constrained embedded robot and offloading computation to a remote server, we also investigate the tradeoffs between onboard computation, communication latency, and real-time control responsiveness.

2. Related Work

Autonomous navigation has traditionally relied on geometric representations, such as occupancy grids and SLAM to enable collision-free motion. These approaches are effective for spatial reasoning but generally lack semantic awareness. With the rise of deep learning, object detection models such as YOLO have enabled real-time visual localization using singular-camera systems, making vision-based navigation feasible on lightweight platforms.

Recent work has explored semantic navigation, where robots leverage object recognition and scene understanding to inform motion planning. Vision-language models extend this paradigm through enabling high-level semantic reasoning about visual scenes. Hybrid systems that combine semantic reasoning with geometric localization are an emerging research direction. Our work follows this hybrid approach, integrating a VLM for semantic context with YOLO-based localization and geometric gap analysis on a resource-constrained robotic platform.

3. System Design

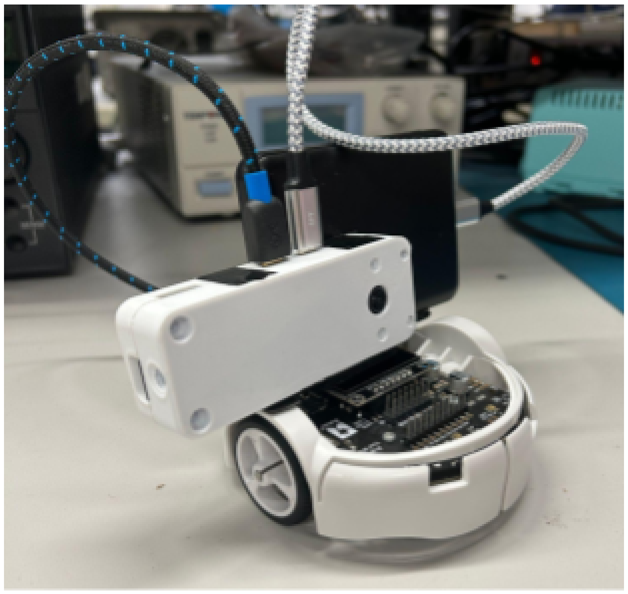

Figure 1. Pololu 3pi+ 2040 executes low-level motion commands, while a Raspberry Pi Zero W captures images and communicates with a remote server for perception and planning. A camera provides visual input for navigation.

The mobile system used in this project is a Pololu 3pi+ 2040 robot, which provides low-level motor control and sensors such as wheel encoders and an Inertial Measurement Unit, including a gyroscope and accelerometer. A Raspberry Pi Camera Zero W is mounted onto the robot and connected to a Raspberry Pi Camera Module, whose objective is to capture images of the environment. *Figure 1* shows the assembled hardware system used in our experiments.

The Raspberry Pi captures camera frames and sends them to a remote server over Wi-Fi using HTTP for perception and action selection, then forwards the resulting navigation commands to the Pololu robot over a serial connection for execution. This creates a simple but effective pipeline: the Pololu focuses on reliable execution of short motion primitives, the Raspberry Pi handles sensing and communication, and the server performs all perception and decision logic.

4. Perception and Navigation

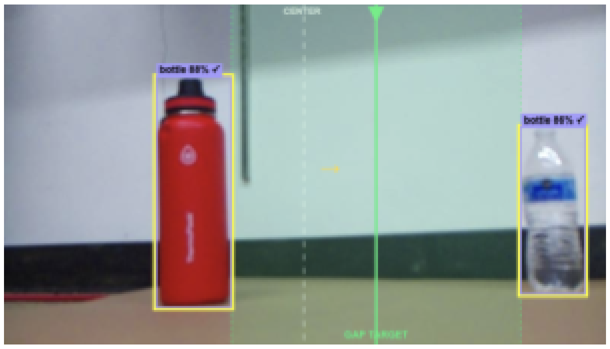

Figure 2. Objects are detected using YOLO and used to compute the largest navigable gap. The gap's position relative to the image center determines the robot's steering action.

The navigation pipeline consists of three phases: semantic identification, object detection, and geometric gap analysis. Figure 2 provides a server-side visualization of the robot's camera feed, including detected objects and the selected navigable gap.

In the identification stage, a vision-language model analyzes the captured image and produces a list of object classes present in the scene that are relevant for the navigation. This information provides contextual understanding of the environment and can be used to guide downstream object detection processes. In the detection stage, a YOLO-based object detection model processes the image and produces bounding boxes, confidence scores, and class labels for the detected objects. These detections provide the precise spatial information necessary to do geometric reasoning.

4.1 Gap Geometry Computation

Let the YOLO detector return N object detections with bounding boxes in pixel coordinates $b_i = (x_{1,i}, y_{1,i}, x_{2,i}, y_{2,i})$, for $i = 1, \ldots, N$, where $x_{1,i}$ and $x_{2,i}$ represents the left and right horizontal edges of object i. Objects are first sorted from left to right using their horizontal center positions $c_i = (x_{1,i} + x_{2,i})/2$.

For adjacent objects in this created ordering, the width of the gap is calculated as $w_k = x_{1,k+1} - x_{2,k}$, for $k = 1, \ldots, N-1$. The largest navigable gap is selected by k*

$= \text{argmax}_k(w_k)$, and the hxorizontal center of the chosen gap is $g_x = (x_{2,k^*} + x_{1,k^*+1})/2$.

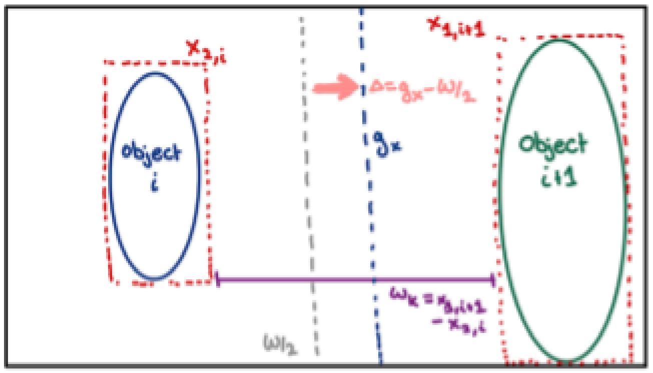

Figure 3. The gap between adjacent bounding boxes is defined by their horizontal edges. The largest gap is selected, and its center is compared to the image center to compute steering corrections.

Figure 3 illustrates this geometric construction. Let W denote the image width in pixels. The offset between the gap center and the image centerline is $\boldsymbol{\Delta} = g_x - W/2$. An action is then selected using a tolerance threshold $\boldsymbol{\tau}$:

- Drive if $|\boldsymbol{\Delta}| \leq \boldsymbol{\tau}$
- Turn right if $\boldsymbol{\Delta} > \boldsymbol{\tau}$
- Turn left if $\boldsymbol{\Delta} < -\boldsymbol{\tau}$

This rule forms a closed-loop feedback controller that incrementally steers the robot toward alignment with the selected gap.

In addition to steering, the system supports safety and termination behaviors. If fewer than two objects are detected or if a "valid" gap can't be established, the controller issues a *stop* action state to prevent random motion. On top of that, the navigation is considered complete when the objects that defined the selected gap are no longer detected for consecutive frames, at which point a *goal* action state is returned. These conditions ensure that the steering commands are applied only when a valid navigable gap exists and provide a clear termination signal once the robot has passed through the gap.

5. Implementation

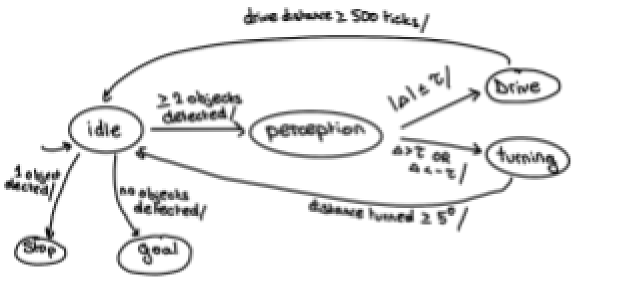

Figure 4. The system alternates between perception-driven navigation and goal detection, issuing steering, stop, or termination commands based on visual feedback.

Figure 4 summarizes the high-level flow of the system as a state machine, highlighting the transition from perception to navigation and goal detection.

The Pololu robot firmware is implemented as an event-driven model. It operates in a loop with a periodic timer and maintains a state machine for command execution. The commands are received over serial as newline-delimited string and mapped to motion commands (e.g., driving forward a fixed distance, turning a fixed angle, etc.). This design ensures determinism and control over the robot.

The Raspberry Pi broker serves as a communication bridge between the robot and server. It continuously captures images from the camera, encodes them as JPEGs in memory, and sends them to the server as HTTP POST requests. The broker also handles serial communication with the robot, including recovery from transient communication failures. The polling interval allows the system to ultimately balance responsiveness with the computational and networking overhead.

The server is implemented using FastAPI and maintains the navigation state across frames. It supports two different modes of operation: an identification mode that uses a VLM to determine relevant object classes and uses llama.cpp as the inference engine because of GPU constraints, and a detection mode that runs YOLO and performs gap analysis. The server stores information about the initial set of objects observed during the start of the navigation and utilizes this information to determine when the robot has passed through the gap. By requiring multiple consecutive frames in which the initial objects are no longer detected, the system can robustly detect goal completion, in spite of a noisy perception.

6. Evaluation

We evaluated the system's ability to correctly identify and navigate through the largest visible gap under realistic operating conditions. Experiments were conducted indoors with two to four stationary obstacles placed at varying horizontal spacings. In each trial, the robot captured images, selected a target gap, and attempted to align itself while moving forward.

Across ten trials with different obstacle configurations, the system correctly identified the largest navigable gap in eight cases. Failures occurred when two obstacles were closely spaced and partially overlapping in the camera view, leading to an inaccurate bounding box estimate. Failures also occurred when the vision-language models and YOLO would confuse an object for two different objects (i.e., mixing up a red water bottle and fire hydrant). Yet, throughout navigation, the robot maintained alignment with the target gap within the acceptable bounds.

We also measured the latency of the perception and control pipeline. The VLM inference took approximately 13 seconds per frame, making it unsuitable for real-time control. On the other end, YOLO inference required approximately 200 ms, while network communication between the Raspberry Pi and the server introduced an additional 800 ms of latency. As a result, the overall system responsiveness was dominated by the communication and VLM inference. Despite these delays, the system remained stable at low speeds, demonstrating the feasibility of offboard perception for embedded robotic platforms.

7. Results, Limitations, Lessons Learned

Ultimately, the system reliably identified and navigated towards the largest visible gap in controlled indoor experiments with multiple stationary obstacles. Across ten trials, the robot selected the correct gap in the majority of cases and maintained stable alignment using closed-loop visual feedback.

The system's performance was primarily limited by latency in the perception and communication pipeline. VLM inference requires approximately 13 seconds, making it unsuitable for real-time control and restricting its role to high-level semantic initialization. In contrast, YOLO inference completed in roughly 200 ms, while communication between the Raspberry Pi and the server introduced approximately 800 ms of round-trip delay. These factors constrained the effective control update rate to the robot and was the motivation for a polling interval that balanced responsiveness with system stability.

Additionally, challenges arose in the serial communication between the Raspberry Pi and the Pololu robot. When commands were issued faster than the robot could process them, the receive buffer could overflow, leading to delayed or unintended actions. This issue was mitigated by explicitly draining the robot's receive buffer during each control cycle, discarding stale or "ignored" commands.

8. Future Work and Implications

In terms of the overall system, future work could focus on reducing the system's latency to enable a more responsive navigation. This can be done via smaller, quantized models on the robot or on nearby edge compute devices that would significantly reduce the reliance on high-latency vision-language inferences.

Additionally, the robot's control performance could also be vastly improved by moving away from reactive steering. For example, predictive control strategies could compensate for any communication delays and reduce possible oscillations via faster motion. On the communication side, having rate-limiting mechanisms or acknowledgment handshakes between the Raspberry Pi and the Pololu robot would also provide stronger guarantees on command delivery and state execution.

On a broader scale, this project ultimately demonstrates the potential of integrating semantic understanding into embedded robotic navigation systems. By combining low-cost hardware with external computational resources, future robotic systems could operate in visually complex environments like warehouses, hospitals, or industrial factories, where geometry alone is insufficient. Ultimately, this work provides a foundation for scalable, semantically-aware robotic systems that can balance embedded robotics control with external perception, enabling more informed navigation decisions within complex environments.