TraVI: Trajectory from Vision and Instruction

Dante B. Cometto
Institute for Experiential Robotics
Northeastern University
Boston, United States
cometto.d@northeastern.edu

Daniel Korn
Institute for Experiential Robotics
Northeastern University
Boston, United States
korn.d@northeastern.edu

Luke Stevenson
Institute for Experiential Robotics
Northeastern University
Boston, United States
stevenson.luk@northeastern.edu

***Abstract*—Common language may be the most natural interface that can be used to interact with a robot. Similarly, RGB vision may be the most human-relatable sensor. Motion planning often requires structured commands, alongside a detailed map of the environment that is annotated with free space. The Trajectory from Vision and Instruction (TraVI) system aims to bridge this gap by enabling the use of a vision language model (VLM) to assist in planning to accomplish arbitrary, natural language motion commands using an RGB camera and, optionally, a time-of-flight depth camera. The system is implemented using a VLM to interpret the user's input text alongside an image from the robot's first-person perspective in order to output pixel waypoints, using geometry and available depth data to project the pixel waypoints into world space, and finally interpreting the position waypoints into an actionable trajectory with motion inputs for the robotic platform. The TraVI system succeeds in demonstrating functionality, and a comparison of VLM options reveals that the Gemini 3 Flash Preview model enables obstacle avoidance and effective planning to meet the user's command.**

***Index Terms*—Robotics, Command and Control Systems, Path Planning, Vision Language Model**

Code: https://github.com/lukesteve03/TRAVI

Hardware Demo Video: https://youtu.be/52PgEoX_his

I. Introduction

The interface in which the user interacts with a robot can take many forms, but natural language, when implemented well, can be the most convenient. Additionally, navigation is a hard problem in robotics: large datasets with many computations are required to translate the complex, dynamic world into a usable representation and to make decisions with it.

Recent work in Vision Language models (VLMs) has evaluated their ability to draw paths of travel onto images when given a prompt [1]. Building upon this idea, our project aims to explore whether a VLM using a natural language command and an RGB camera image is sufficient for a robot to make sound navigation plans.

Ultimately, the TraVI system implements a planning and execution pipeline that enables robots to enact arbitrary motion commands using a VLM to interpret the input command alongside a first-person RGB image and time-of-flight depth data.

II. Motivation

This research aimed to design and implement a system that would turn an input image and command into an executable path a robot can follow. Tim Windecker's NaviTrace paper, which benchmarks a variety of VLMs' ability to annotate a path on 2D images, served as the conceptual foundation for this project [1]. Traditional methods of trajectory planning require construction of a map. Using natural language to give instructions regarding navigation tasks is an intuitive way for human operators to command robotic platforms. Also, one command given in natural language can describe an entire path consisting of multiple waypoints, providing for the ability to describe more complex navigation tasks, rather than only setting a single goal point.

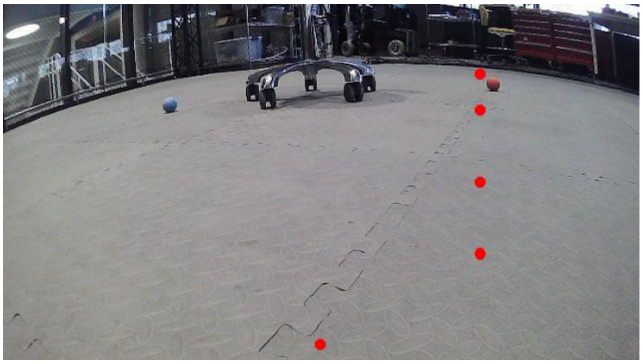

Fig. 1. *Qwen3-VL generated trajectory for "Go to the orange ball"*

A. Literature Review

This project was primarily inspired by the results of NaviTrace, which established a benchmark for evaluating a VLM's ability to predict 2D navigation traces in the image pixel space, across a variety of robot types [1]. However, a system has not been implemented that uses these traces to implement a functional planner.

The most related work has been done in Daeun Song's VL-TGS paper, where a lidar provided the data used to generate candidate trajectories, then a VLM, using data from an RGB camera, was used to evaluate and choose a trajectory [2]. We will build upon this work in two primary ways. First, we will use the VLM itself to generate and choose a trajectory without the use of lidar, which decreases the hardware cost. Second, VL-TGS focused on outdoor navigation, and we will expand this to include the indoor setting.

VLMs and large language models (LLMs), due to their general world knowledge, have a history of being applied

in similar situations. This includes using VLMs for trajectory generation for predicting pedestrian trajectories for autonomous vehicle navigation, as in [3], using LLMs for guiding manufacturing machines, as in [4], and controlling a robot arm to complete tasks, as in [5].

III. Problem Statement

In this project, we sought to develop a navigation system that uses a function from natural language commands and images to generate a trajectory. To be precise, let the pose of a robot be described by the manifold M. Let $\mathcal{T}$ be the set of possible trajectories $\gamma : [0, 1] \to M$ starting at the robot's current pose. Also, let $\mathcal{S}$ be the set of possible natural language commands, and let $\mathcal{I}$ be the set of possible images.

The core problem we worked to solve was the creation of a function

$$F : \mathcal{S} \times \mathcal{I} \to \mathcal{T}$$

that, when given a natural language command and a current image from the robot's point of view, creates a trajectory from the robot's current pose to a pose that adequately achieves the goal of the command. We ultimately expanded this function to additionally accept a depth image from a time-of-flight camera, which was used to improve depth estimation.

IV. Proposed Solution

Our team implemented this system algorithmically, creating an interface that provides the natural language command and image to a VLM, and developed baseline functionality that transforms the trajectory into an actionable plan. The processing pipeline for this project begins with a verbal natural language input and ends with a 3D action plan, and can be seen in full in Algorithm 1.

Algorithm 1 TraVI Trajectory Generation Function

1: Initialize: A VLM with prompt background information p
2: Input: user command in natural language s, current first-person image from the robot i
3: Retrieve a list of waypoints $w_{uv,k}$ in pixel space from providing command s and image i to the VLM
4: Transform pixel waypoints $w_{uv,k}$ into position waypoints $w_{xyz,k}$ in world space {using geometry or another source of depth data}
5: Convert waypoint positions into a trajectory plan τ
6: Output: plan τ

An overview of the system can be seen graphically by processing phase in Figure 2.

A. Code Overview

Implementation of our proposed solution relies on stable code base architecture and choosing the correct supporting frameworks. The system is split into two sections, Docker and the Robotic Operating System (ROS 2), each handling different parts of the pipeline.

1) Docker: A containerized environment framework allowing for dynamic sizing and resource management. Docker is the starting point for the system; handling image and text inputs via an API, processing them through the object detection and the vision-language model, then returning a list of points and writing 6 files to the output directory. The following files are saved for every scenario for analysis and evaluation:

- `annotated_output.jpg` - Input image with VLM prediction points overlaid
- `command.txt` - Input command
- `input_image.jpg` - Input image
- `waypoints_2d.json` - JSON file of VLM predicted points
- `yolo_output.jpg` - Object detection output image
- `yolo_result.json` - Object detection coordinates

Internally, the docker container accepts an image and parses the input text, and provides them to the yolo11n object detection model, creating an objective point if available. The object detection, as well as the input command and input image, are then passed to the VLM model. The code supports both local VLMs (e.g Qwen3-VL) and cloud models (e.g Gemini, OpenAI), and will load the configured parameters to generate the list of 2D pixel waypoints points.

B. ROS 2

ROS 2 is a node based system that allows for each node to publish and/or subscribe to a data stream, termed a topic. This allows for quick and efficient data transfer between scripts, and is extremely beneficial in robotics. Jango, the robot this reserach is centered around, is a custom quadruped robot and is designed on ROS 2, already publishing topics such as odometry, joint commands, joint states, IMU data, RGB camera images and time-of-flight depth images. Taking advantage of these preexisting topics, we built three new nodes to implement our problem solution. From our docker section we have a list of coordinates corresponding to our trajectory in 2D space, and the first ROS 2 node accepts this list, and, using a depth projection algorithm and the time-of-flight depth image topic, converts them into 3D waypoints representing the robot's path in 3D world space. These waypoints are published to a topic with a new node so they can be viewed in simulation. A the third node converts these waypoints into Twist velocity commands so the plan can be executed.

C. Planner (VLM)

The TraVI system is centered around a vision languages model's capability to plan a trajectory based on an input image and input text prompt. The system we have designed and implemented has support for both local model and cloud model capabilities, as well as a configurable prompt and number of planned pixel waypoints. The system prompt, derived from the NaviTrace research [1], details the robot camera and physical embodiment context, trajectory rules, and output format, constraining the VLM's output to a list of coordinates corresponding to points on the image. It is to be noted that

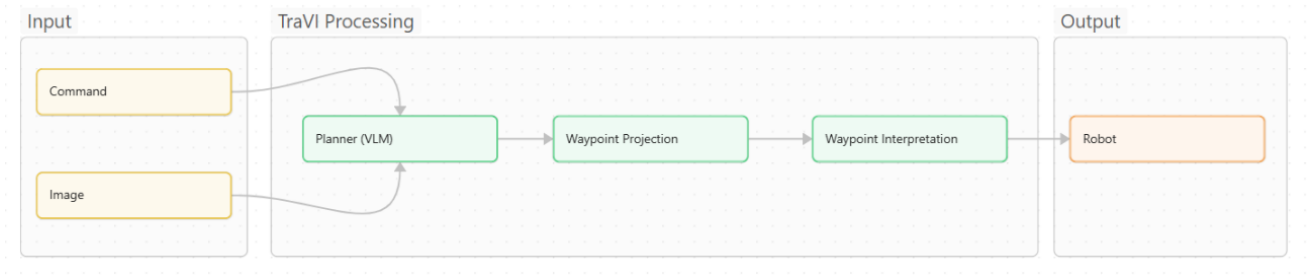

Fig. 2. Flowgraph of Processing Pipeline

performance in this system stems almost directly from the model and prompt selection. To increase successful trajectory planning likelihood and trajectory accuracy, object detection was implemented to create the final destination point. All of these components together create a configurable, reliable, and stable system with the VLM making consistently feasible trajectory plans across input commands and contexts.

At this stage, the user is provided a visual of the planned trajectory overlaid on the robot's first person image, and the user is able to iterate with the VLM on the plan. Once the user is satisfied with the plan, the user is able to indicate to the robot that the plan should be executed. More detail can be found in Section IV-G.

D. Waypoint Projection

Once the VLM Planner has generated the list of waypoints, and the plan is approved to be executed, the system must then transform the waypoints from pixel space into world space. This is done in a two stage process.

First, in order to provide good initial estimate for use with the depth data from the time-of-flight camera, we naively raycast the pixels to the ground plane. This requires two major assumptions: that the ground plane is flat and level at $z = 0$, and that the camera frame's pose in the world frame only depends on its pitch angle θ_C and the translation to the camera's optical center $\mathbf{O_c}$.

The raycasting process can be seen in Figure 3.

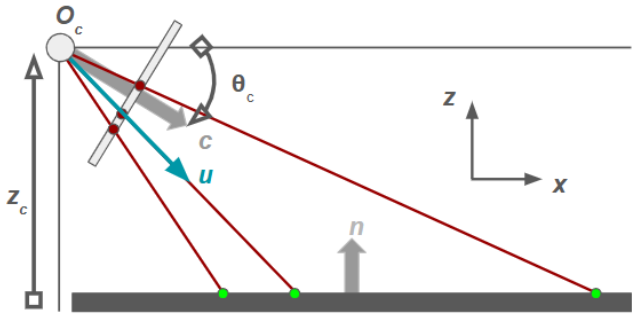

Fig. 3. Raycasting from Pixel Space to World Space

We begin by calculating the direction vectors $\mathbf{u}$ for each pixel waypoint $w_{uv,k} = [u_k, v_k]^T$ by applying the transform T_{BC} from the camera frame C to the base frame B such that

$$\mathbf{u} = T_{BC} w_{uv,k},$$

and scaling to a length of 1 gives us the unit direction vector

$$\hat{\mathbf{u}} = \frac{\mathbf{u}}{||\mathbf{u}||},$$

in world space.

Now, we can parametrize the ray for the pixel as

$$r(t) = \mathbf{O_c} + t\hat{\mathbf{u}} \tag{1}$$

for the camera's optical center's position $\mathbf{O_C}$ expressed in the base frame.

This ray already contains useful information. We can see immediately that, if the z-component of this vector is nonnegative, then the ray will never intersect the ground plane. Of the rays with negative z-components, we can solve

$$\tilde{t} = \frac{O_{C,z}}{\hat{u}_z}$$

for the value of parameter t at intersection with the ground plane, and can thus recover the final intersection point $w_{xyz,k}$ via substituting $t = \tilde{t}$ in the ray's equation in Equation 1.

An example of an identically sized ball raycasted throughout the image can be seen in Figure 4. As can be seen in the image, the assumptions that simplify the naive raycasting method do not always hold, and thus we can fuse this method with depth data for a better world-space waypoint position estimate.

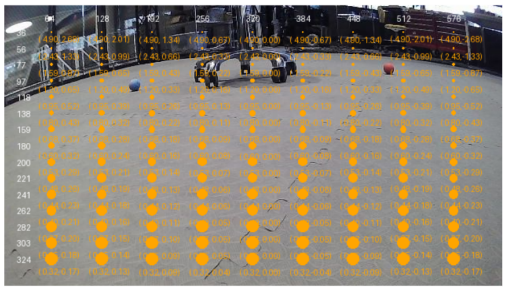

Fig. 4. Raycasting Example

While naive raycasting captures the shape of the 2D trajectory in 3D, it assumes a perfectly flat ground plane and an ideal camera pose. In practice, system error accumulates, causing

real depth inference to be lost. To ground our raycasting in reality, we utilize an anchoring method using a single depth measurement from the time-of-flight camera. The anchor depth d_{anchor} is measured from the bottom row of the depth image as it best captures the ground plane. The furthest value from this measured data is taken giving us the real world distance from the camera's optical center $\mathbf{O_c}$ to the ground along the ray closest to the robot.

Let $\mathbf{p}_k$ denote the naive raycasting ground intersection for waypoint k. The ray-cast distance for each waypoint is defined as:

$$d_{\text{ref},k} = \|\mathbf{p}_k - \mathbf{O_c}\|,$$

which is the Euclidean distance from the camera origin to the ground plane along the k-th ray.

Using the first waypoint ($k = 0$) as the reference, we compute a global scaling factor

$$\lambda = \frac{d_{\text{anchor}}}{d_{\text{ref},0}}$$

that captures the ratio between the true measured depth and the naive raycast estimate, providing a scaling factor for each waypoint.

Applying λ to every waypoint, the correctly scaled point $\mathbf{p}_k^{\text{scaled}}$ is calculated by scaling each ray's magnitude while preserving its direction:

$$\mathbf{p}_k^{\text{scaled}} = \mathbf{O_c} + \lambda \cdot d_{\text{ref},k} \cdot \hat{\mathbf{u}}_k.$$

Finally, the vertical component is forced to the ground plane:

$$p_{k,z}^{\text{scaled}} = 0,$$

so scaling only adjusts the horizontal magnitude (x, y) while maintaining all waypoints on the same plane. This preserves the shape of the trajectory while grounding the depth in reality according to the data collected from the robot.

E. Waypoint Interpretation and Execution

After the 3D waypoints are published in ROS2, the waypoint interpreter converts them into an action plan that can be executed by generating time-stamped `geometry_msgs/Twist` messages. Rather than creating a static plan, the node used a closed-loop state machine that monitors each waypoint and the robot's position using odometry feedback.

Let the robot's current position be $\mathbf{q} = (x, y)$ with heading ψ, determined from the odometry topic. For the current target waypoint $\mathbf{w}_k = (w_x, w_y)$, we define the displacement and desired heading as

$$\begin{aligned}\Delta\mathbf{p} &= \mathbf{w}_k - \mathbf{q},\\ d_k &= \|\Delta\mathbf{p}\|,\\ \psi_{\text{des}} &= \text{atan2}(\Delta p_y,\, \Delta p_x),\end{aligned}$$

and the heading error as

$$e_\psi = \text{wrap}(\psi_{\text{des}} - \psi),$$

where $\text{wrap}(x)$ normalizes the angle to $(-\pi, \pi]$.

For each waypoint, the executor iterates through three states:

Turning. The robot rotates in place at a fixed angular velocity ω_{turn} in the direction of e_ψ until the heading error reaches our set tolerance e_ψ^{tol}:

$$\begin{aligned}v_x &= 0,\\ \omega_z &= \text{sgn}(e_\psi) \cdot \omega_{\text{turn}}.\end{aligned}$$

If the timeout T_{turn} is reached, the executor moves to the driving stage even if there is heading error. This is used to prevent the robot from turning indefinitely.

Driving. The robot walks forward at a constant velocity v_{drive} with a steering correction to maintain alignment with the target waypoint:

$$\begin{aligned}v_x &= v_{\text{drive}},\\ \omega_z &= \text{clamp}(K_p \cdot e_\psi,\ -\omega_{\max},\ \omega_{\max}),\end{aligned}$$

where K_p is the steering gain and $\omega_{\max}$ is the angular speed limit. If the orientation error exceeds $2.5 \cdot e_\psi^{\text{tol}}$ during locomotion, the node returns to the turning state to minimize heading error before moving forward.

Arrival. When the distance to the current target waypoint is less than or equal to the predetermined arrival tolerance $d_k < d_{\text{tol}}$, the robot pauses and either waits for the user to indicate that it should continue trajectory following, or that it has arrived at the commanded the target, in which the execution process ends.

This control architecture allows for a closed-loop trajectory following system, uses odometry and waypoint tracking to properly understand and execute the generated plan. Separating the planning and execution phases, as well as providing operator input and control, prevents poor plans from being executed, and maintains an appropriate level of autonomy while preserving human oversight.

F. Hardware

The robot used to test this system is a custom 12 degree-of-freedom quadruped named Jango, which can be seen in Figure 5. It is powered by a Raspberry Pi single board computer running ROS 2 to poll sensors, calculate odometry, and send movements.

Previous development of this system exposed several ROS 2 topics that are vital to this research including odometry (Open loop, EKF + IMU), IMU (precorrected), ToF camera depth data (as a PointCloud2), RGB images, a robot description (in Unified Robot Description Format), current joint states, and a method of processing joint commands. For the experiments conducted, the robot was calibrated and tuned to create a stable forward and rotation gait, ensuring that plan execution was not bottlenecked by hardware. The robot's role that of an interaction point and verification method, as all decision making is done by the VLM and planing sequence. Execution of the planned trajectory is handled by the waypoint executor node described in the previous section, which subscribes to the odometry topic and publishes `geometry_msgs/Twist` velocity commands at a fixed rate.

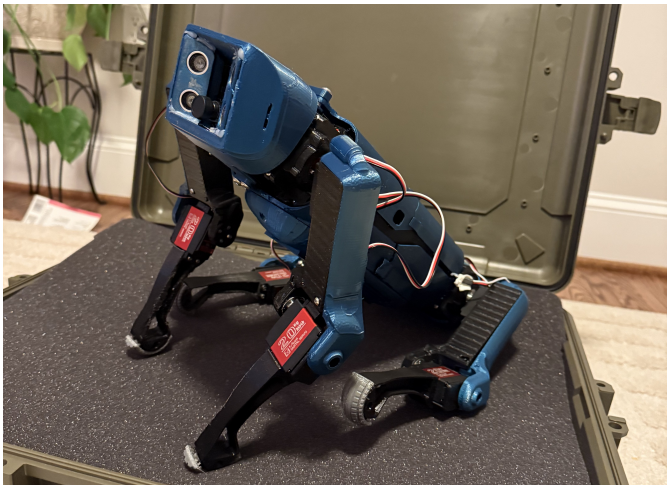

Fig. 5. Jango

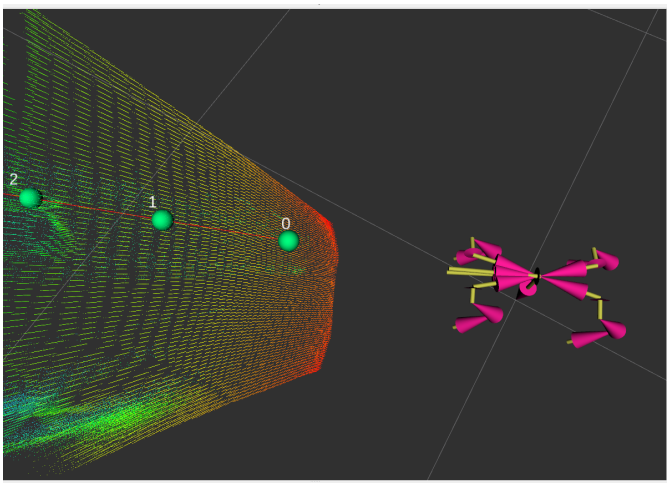

Fig. 7. Rviz2 simulation showing the robot, pointcloud2 depth data, waypoints, and trajectory.

G. User Interface

To make the system deployable in the field, a control system was developed to run each segment of the pipeline individually. Accessing the system through a user interface that is connected to the robot and a server-side computer, the user first takes a picture with the robot's on-board RGB camera. The user then inputs a natural language command, and sends both the captured image and command to the VLM where the generated list of coordinates will be displayed numerical as well as overlaid on the image. Next, the user presses a button to convert the 2D coordinates into 3D waypoints and publish them to a ROS 2 topic to be verified in simulation. Once verified, the user can execute the plan and the robot will pause at each waypoint along the trajectory and wait for the user to advance. This pipeline allowed for trials to be run quickly and reliably. The user interface can be seen in Figure 6.

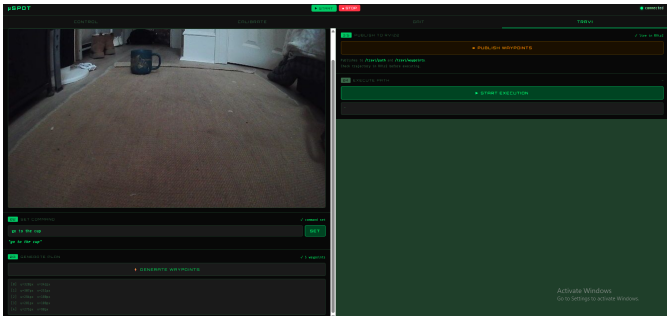

Fig. 6. User interface, separating the pipeline into segments

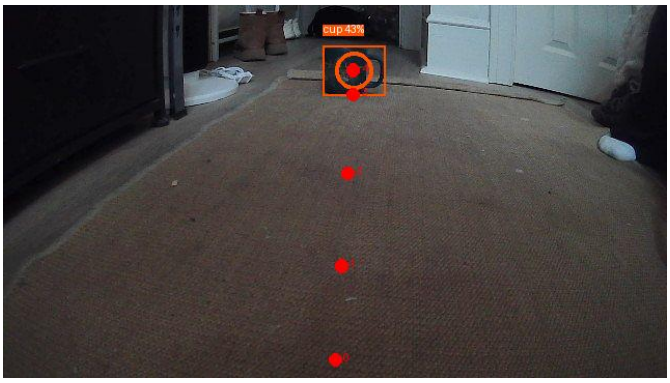

Fig. 8. Example predicted trajectory overlaid on the input image with object detection.

V. Results

A. Experiment Description

In order to assess our planning system's performance, we compared the plans generated to those of a human expert, allowing us to holistically assess performance. Additionally, we are also able to compare the system's performance across multiple VLMs: here, we compare the results we obtained from using the locally run Qwen3-VL-4B VLM to the larger, network-accessible models GPT 4.1 Mini and Gemini 3 Flash Preview.

The comparison is done using three metrics. Let $\gamma(t) : [0,1] \rightarrow M$ represent the planned trajectory at normalized time t, and $\gamma_{\text{hum}}(t) : [0,1] \rightarrow M$ represent the human expert path. Let the total number of waypoints be N, and the planned waypoint at time t_k be $w_{xyz,k}$ for $k \in \{0,\ldots,N\}$ where $w_{xyz,0}$ is the plan's starting position. Similarly, let the total number of waypoints in the human path be N_{hum}, and the waypoint of the human path at time t_k be $w^{\text{hum}}_{xyz,k}$ for $k \in \{0,\ldots,N\}$ where $w^{\text{hum}}_{xyz,0} = w_{xyz,0}$ is the starting position.

The first metric is target error, which is the distance of the final point in the planned trajectory from the target point that is specified from the input command as interpreted by the human expert. We calculate this as

$$L_{\text{target}}(\gamma, x_{\text{goal}}) = ||x_{\text{goal}} - w_{xyz,N}||,$$

which is the Euclidean distance from the last point of the trajectory to the true goal point.

Our second metric will be the root mean square error (RMSE) of the waypoints of the planned path with their corresponding points (by identical normalized time) on the human path. We calculate this as

$$L_{\text{RMSE}} = \sqrt{\frac{1}{N}\sum_{k=1}^{N}\left(w_{xyz,k} - \gamma_{\text{hum}}(t_k)\right)^2},$$

where we calculate $\gamma_{\text{hum}}(t_k)$ by interpolating between the human path's waypoints.

Our third metric will be the the difference in total length of the planned path from the human expert path, calculated as

$$L_{\Delta\text{dist}} = \left(\sum_{k=1}^{N} ||w_{xyz,k} - w_{xyz,k-1}||\right) - \left(\sum_{k=1}^{N_{\text{hum}}} ||w^{\text{hum}}_{xyz,k} - w^{\text{hum}}_{xyz,k-1}||\right).$$

These three metrics allow us to compare output trajectory quality across VLMs, measuring whether the system reaches the target point, how similar the path is to the expert path, and whether any additional path length is planned than necessary.

In order to conduct the test, we collected a set of 9 natural language commands and 9 images, and applied the system using each VLM to each image. We annotated a human expert path in pixel space, which prevents any potential pixel-space to world-space conversion error from influencing the trajectory planning comparison. Then, we compute the three metrics for each planned trajectory. For consistency, all VLMs were tested with the same prompt for each image and same number of output waypoints ($n = 5$).

B. Experimental Results

The planned trajectories from each VLM alongside the human expert trajectory can be seen from the robot's first-person perspective in Figure 9, and from an overhead perspective in Figure 10. Additionally, the metrics for each planned trajectory can be seen in histogram form in Figure 11 and as a violin plot in Figure 12.

The trajectories tested in this comparison were first-shot: the user did not have a chance to iterate with the VLM. This increases the final distance metric, as the VLM sometimes does not correctly interpret the user's command. However, this is not unexpected, as common interactions with LLMs often require substantial steering from the human operator. This application is no different, and the functioning system features the aforementioned planning phase separated from the plan execution phase by operator approval.

Even despite this, notice that the TraVI system often produces competent, useful trajectory plans that avoid obstacles and closely match the human expert plan. Without any prior knowledge of the area, and with only an RGB camera and a complimentary time-of-flight depth camera, the TraVI system enables autonomous trajectory generation to meet arbitrary

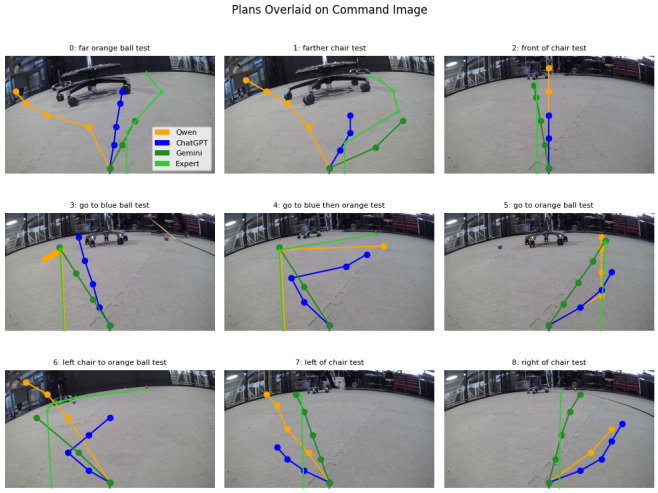

Fig. 9. Planned Trajectories Overlaid on First-Person Image

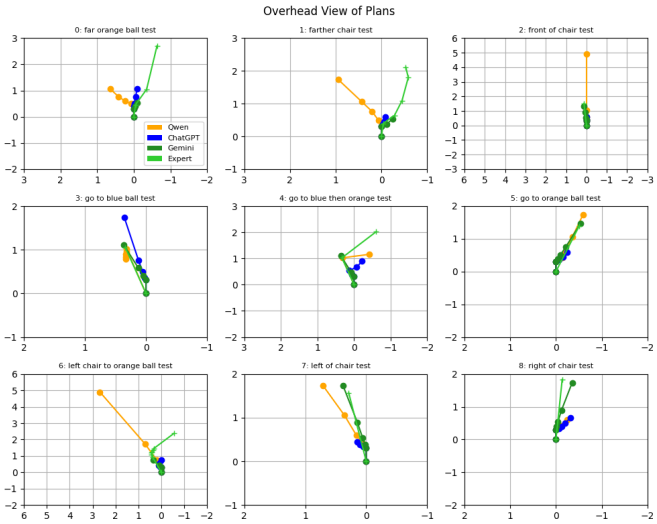

Fig. 10. Planned Trajectories from Overhead

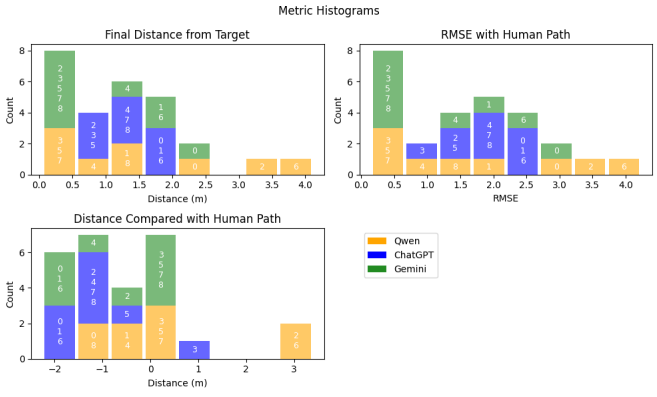

Fig. 11. Planned Trajectory Metrics Histogram with Test Index Marked

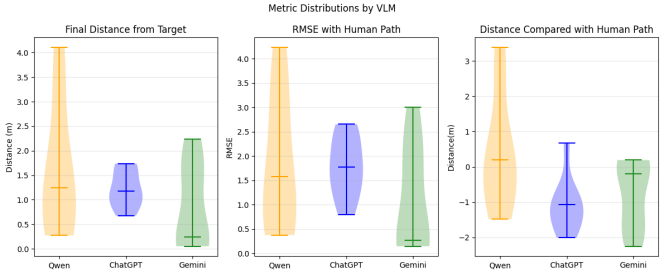

Fig. 12. Planned Trajectory Metric Distributions with Median Marked

movement commands.

In these tests, the Qwen model featured significantly higher variance in trajectory output, and both the ChatGPT model and Gemini model were more consistent. However, ChatGPT consistently stopped short of the desired point. Of the three models, Gemini correctly understood the operator intent and created a useful trajectory the most often, reaching the goal in over 55% of first-shot plans. The Gemini model also produced the most humanlike plans, frequently having the lowest RMSE with the human expert path. Despite this, ChatGPT was the least variable, and with further specific training and prompt engineering, it may be possible to correct its consistent early stopping.

C. Demonstration on Hardware

In our demonstration (link: https://youtu.be/52PgEoX_his), we show the robot successfully using TraVI by capturing an image, generating a trajectory, and enacting the plan. The video details the system's usage and how each step of the pipeline is executed. Once the trajectory has been predicted, the waypoints are approved in simulation, and then the plan is executed. Once the robot reaches a waypoint, it pauses and waits for user acknowledgement to advance.

VI. Conclusion

TraVI shows that, given a first-person RGB image and a natural language command, a Vision Language Model can generate plausible navigational routes for a quadruped robot. The full pipeline, from image capture through 3D waypoint generation to executable Twist commands, was completely validated. The waypoint interpreter successfully converts 3D waypoints into Twist commands, and the operator approval step is used to prevent bad plans from reaching the robot.

Trajectory quality was compared across three different VLMs: one locally run model, Qwen3-VL-4B, and two larger network-accessible models, GPT-4.1 Mini and Gemini 3 Flash Preview. Of the three models, Gemini was the most reliable to understand operator intent, reaching the goal in over 55% of first-shot plans and generating the most human-like trajectories as measured by RMSE against expert paths. ChatGPT was the most consistent but it showed a systematic tendency to stop short of the target, suggesting that targeted prompt engineering or fine-tuning could potentially correct this bias. Qwen, while requiring no internet connection, showed significantly higher variance, which highlights the capability difference between a locally run 4B-parameter model and the larger cloud-hosted alternatives.

The limitations of the current system include both the VLM first-pass reliability and the absence of closed-loop replanning, which means that the robot currently runs a fixed plan without reacting to environmental changes mid-execution. Future work should aim at fine-tuning a VLM on robot-specific navigation data to try and improve the first-pass reliability, as well as implementing a closed-loop architecture in which the planner is recalled at each waypoint using a fresh camera frame to enable a dynamic obstacle response.

References

[1] T. Windecker, M. Patel, M. Reuss, R. Schwarzkopf, C. Cadena, R. Lioutikov, M. Hutter, and J. Frey, "NaviTrace: Evaluating Embodied Navigation of Vision-Language Models," Mar. 2026, arXiv:2510.26909 [cs]. [Online]. Available: http://arxiv.org/abs/2510.26909

[2] D. Song, J. Liang, X. Xiao, and D. Manocha, "VL-TGS: Trajectory Generation and Selection Using Vision Language Models in Mapless Outdoor Environments," *IEEE Robotics and Automation Letters*, vol. 10, no. 6, pp. 5791–5798, Jun. 2025. [Online]. Available: https://ieeexplore.ieee.org/document/10960724

[3] S. Moon, H. Woo, H. Park, H. Jung, R. Mahjourian, H.-g. Chi, H. Lim, S. Kim, and J. Kim, "VisionTrap: Vision-Augmented Trajectory Prediction Guided by Textual Descriptions," in *Computer Vision – ECCV 2024*, A. Leonardis, E. Ricci, S. Roth, O. Russakovsky, T. Sattler, and G. Varol, Eds. Cham: Springer Nature Switzerland, 2025, vol. 15064, pp. 361–379, series Title: Lecture Notes in Computer Science. [Online]. Available: https://link.springer.com/10.1007/978-3-031-72658-3_21

[4] Z. Boya, W. Tian, and Z. Pai, "Large language model-driven dynamic trajectory planning for human-guided robot assembly," *Manufacturing Letters*, vol. 44, pp. 1387–1394, Aug. 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2213846325001920

[5] T. Kwon, N. D. Palo, and E. Johns, "Language Models as Zero-Shot Trajectory Generators," *IEEE Robotics and Automation Letters*, vol. 9, no. 7, pp. 6728–6735, Jul. 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10549793/