Photorealistic Simulation for Validating Perception–Planning–Control Pipelines in Autonomous Driving

CS588 Group 9 Project Report

Minkyung Kim, Minhyuk Jang, Dong-Hwan Jang, Jongann Lee

***Abstract*— We present a Highbay vehicle navigation framework that combines photorealistic simulation, obstacle detection, and obstacle-aware planning and control. The system uses a 3D Gaussian Splatting (3DGS) digital twin with Gazebo vehicle dynamics, estimates obstacle locations from RGB-D sensing, and plans safe detours using a hierarchical replanning pipeline. A high-level replanner generates collision-free trajectories, while a low-level Stanley controller tracks the selected path. We evaluate the pipeline in simulation and on the physical GEM e4 vehicle in the Highbay environment.**

I. INTRODUCTION

Autonomous navigation in the Highbay environment requires both realistic testing conditions and safe obstacle avoidance. Direct hardware testing is limited by safety and repeatability, and the default Gazebo simulator camera does not closely match real visual observations. To address this gap, we build a digital-twin-based navigation pipeline that combines Gazebo vehicle dynamics with a pose-conditioned 3DGS renderer [1].

On top of this simulation framework, we implement RGB-D obstacle detection and a hierarchical planner-controller that generates and tracks safe detour trajectories. The goal is to use the reconstructed environment not only for visualization, but also as part of a closed-loop navigation workflow that can be evaluated in simulation and transferred to the GEM e4 platform.

II. RELATED WORK

A. Simulation Platforms for Autonomous Driving

Simulation is widely used in autonomous navigation because it enables safe, repeatable testing before hardware deployment. Conventional simulators such as CARLA [2] provide controllable environments, vehicle models, sensors, and weather conditions for evaluating autonomy algorithms. However, these environments are typically built from manually designed assets, which may not match the appearance of a specific real-world deployment site. This sim-to-real visual gap is especially important for camera-based perception, where performance can depend strongly on lighting, texture, and scene appearance.

Recent work has moved toward data-driven and real-to-sim simulation. VISTA 2.0 [3] synthesizes sensor feedback from real driving data for policy learning and evaluation, while Waymax [4] provides large-scale data-driven simulation for multi-agent driving scenarios. Neural rendering methods further improve visual realism by reconstructing environments directly from captured data. For example, DrivingGaussian [5] applies 3D Gaussian Splatting to dynamic driving scenes. Our project follows this real-to-sim direction in the Highbay environment: Gazebo provides vehicle dynamics and state updates, while a 3DGS reconstruction provides a more realistic front-camera stream for perception and navigation.

B. Autonomous Navigation Pipelines

Autonomous navigation systems are commonly organized either as end-to-end policies or modular pipelines. End-to-end methods directly map sensor observations to control commands, as in early vision-based steering systems [6] and later demonstration-based approaches such as ChauffeurNet [7]. These methods can reduce manual design, but they often require large datasets and can be difficult to interpret or debug, especially in rare or safety-critical scenarios.

Modular systems instead separate perception, planning, and control. This structure makes each component easier to validate and improves debuggability when transferring from simulation to hardware. Our project follows this modular design: RGB-D perception estimates obstacle positions, the replanner generates a safe local detour, and the low-level controller tracks the selected trajectory. This allows the 3DGS simulation, obstacle detection, replanning, and tracking modules to

be evaluated separately before closing the loop on the GEM e4 vehicle.

III. METHODS

The proposed system has three components. First, a 3DGS-based digital twin provides photorealistic front-camera images while Gazebo maintains the vehicle dynamics and state updates. Second, an RGB-D perception module detects semantic obstacles and estimates their relative positions for planning. Third, a hierarchical navigation module performs collision avoidance: the high-level replanner samples feasible detour trajectories around obstacles, and the low-level Stanley controller [8] tracks the selected trajectory using steering and speed commands.

We evaluate the pipeline through three components: visual simulation, obstacle detection, and navigation performance. The 3DGS simulation is compared against real Highbay images, obstacle estimates are checked against ground truth or RViz visualization, and the replanning controller is evaluated using tracking error and obstacle clearance in both simulation and GEM e4 hardware experiments.

IV. PHOTOREALISTIC DIGITAL TWIN

We created a photorealistic digital twin of the Highbay test environment to provide more realistic visual input than the default Gazebo simulator, while reducing the need for repeated GEM e4 hardware testing.

A. Video Collection

We recorded approximately 50 minutes of video using two handheld smartphone cameras and selected around 2,000 frames in total for training 3DGS. These frames were chosen to provide sufficient viewpoint diversity and scene coverage for the subsequent COLMAP-based camera pose estimation and scene reconstruction pipeline.

B. LiDAR Point Cloud Combination

We built a 3D point cloud map of the Highbay environment using pre-recorded GEM e4 LiDAR and GNSS data. We used the *lane 2* recording because it covers the Highbay in a full loop, containing 651 LiDAR sweeps over 65 seconds with roughly 75,000 points per sweep.

We first calibrated the ego-to-LiDAR rotation R_{le}. Pitch and roll were estimated by fitting ground planes to the LiDAR scans and aligning the average ground normal with the ego frame's $+z$ axis. Yaw was estimated by comparing ICP-based LiDAR motion with GNSS-derived motion over a straight segment, where the GNSS heading was computed from finite differences of the (x,y) positions. The resulting calibration was yaw 178.3°, pitch -1.4°, and roll 0.5°.

To combine the scans, we registered consecutive LiDAR sweeps using KISS-ICP [9] and fused the relative ICP poses with GNSS positions in a GTSAM pose graph [10]. The graph was constructed in SE(2), with each frame represented by (x,y,ϕ) and the 6-DoF ICP estimates projected to planar relative factors. This planar formulation is appropriate because the LiDAR mount tilt is already absorbed by the calibrated R_{le}. The noise parameters are listed in Table I; ICP factors during sharp turns were down-weighted by scaling their noise by $(1+4|\omega|)$. Loop closures were also tested, but they did not noticeably improve the final map and were omitted.

Factor	σ
Anchor (frame 0)	0.001
GNSS xy	0.1
KISS-t	0.1
KISS-rot	0.3

TABLE I: Noise parameters used for pose graph optimization.

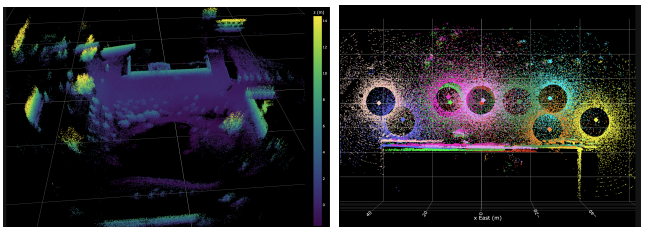


Fig. 1: *Left*: Combined LiDAR point cloud visualization. Colored using the height value. *Right*: 10 selected frames' point cloud. Each frame is given a different color.

Fig. 1 shows the combined LiDAR point cloud. The visualization clearly resembles the Highbay environment, with the north-facing wall of the building clearly visible. However, residual yaw smear can still be visible, especially on distant objects. This is acceptable downstream because 3DGS optimizes Gaussian positions against the input images, so small inconsistencies in the seed are absorbed during training.

C. 3DGS Training

We trained a 3D Gaussian Splatting model to represent the Highbay environment as anisotropic Gaussian

primitives. To obtain a metric and well-aligned initialization, we used the combined LiDAR point cloud as the main seed for the Gaussians.

Since the smartphone videos did not provide camera poses, we estimated frame poses using COLMAP-based Structure from Motion (SfM). The COLMAP reconstruction is defined only up to an arbitrary similarity transform, so we aligned its sparse point cloud to the LiDAR map using a Sim(3) transformation consisting of rotation, uniform scale, and translation. The alignment was initialized with ICP and then manually refined, which placed the recovered camera poses in the LiDAR metric ENU frame.

The LiDAR map provides dense structure inside the Highbay, but it does not cover distant objects or the sky. Therefore, we used approximately 2 million voxel-downsampled LiDAR points for the main scene, 346,894 COLMAP points for out-of-range structure, and 30,000 synthetic sky points sampled from a 500 m hemisphere. The 3DGS model was trained for 30,000 iterations using the default hyperparameters from [11], with two modifications: `densify_until_iter` was reduced from 15,000 to 7,000, and anti-aliasing was enabled.

D. Integration with Gazebo

We replaced the Gazebo-rendered front camera of the GEM vehicle with images rasterized from the trained 3DGS model. In this framework, Gazebo continues to handle vehicle dynamics, physics, and state updates, while the front camera image stream is generated separately from the Gaussian Splat scene and published via ROS. This design preserves the original simulation pipeline while improving the realism of the visual sensor output.

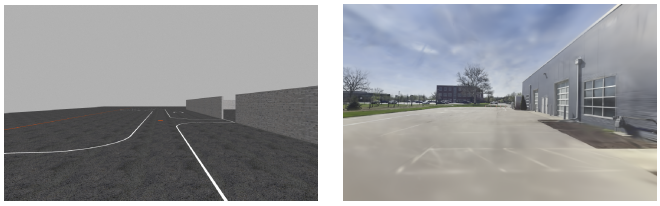


Fig. 2: Comparison between the front-camera view rendered from the Gazebo environment (left) and in the Gaussian Splatting scene (right).

To align with Gazebo's scene, we applied a similarity transformation to match the reconstructed scene's geometry to that of the simulated highbay environment. Once this alignment was established, the pose of the simulated vehicle and its front camera in Gazebo were mapped to the coordinate convention required by the 3DGS renderer. The rendered RGB image was then published to the same ROS topic (Fig. 2). This allowed downstream modules to use the photorealistic image stream without modifying the rest of the software stack.

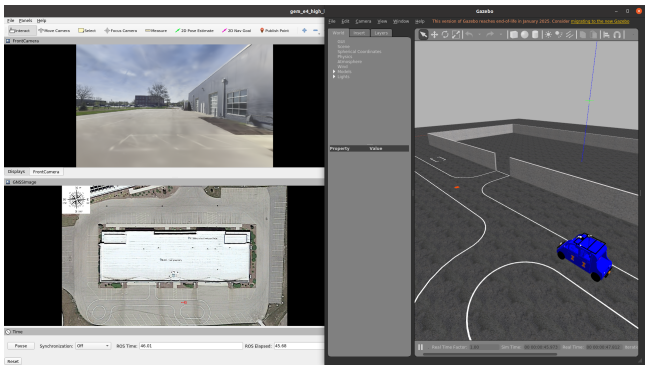


Fig. 3: Integrated Gazebo–3DGS simulation. Top: the Gazebo–3DGS interface, where Gazebo provides vehicle dynamics and state updates while the front-camera image is rendered from the 3DGS model.

E. Object Insertion and Composition

To support closed-loop testing of obstacle-aware navigation in the digital twin, we built a compositional object library of seven assets: four pedestrians, each carrying an umbrella; a traffic cone; a stop sign; and a cube-shape obstacle. Each asset was reconstructed as a separate 3DGS PLY from a short ($\sim$30 s) handheld walk-around video, using the same COLMAP+3DGS pipeline as the main scene, and then transformed into the Highbay frame for rendering.

Pedestrian placement via shared-frame registration. The four pedestrians were captured at the Highbay in a separate session, whose access window only permitted recording under rainy weather; the main scene reconstruction was captured on a sunny day at the same location. Because both recordings cover the same physical site, each pedestrian's per-object COLMAP reconstruction shares many frames with the main Highbay reconstruction. We use these shared frames to estimate the similarity transform $(s, R, \mathbf{t})$ from each pedestrian's independent frame into the Highbay frame by a closed-form Umeyama fit on the per-image camera centers, recovering both metric scale and the original capture pose so the pedestrian PLY merges into the Highbay scene at the position they were standing.

Background removal for indoor captures. The cone, stop sign, and cube were captured indoors, away from the Highbay, so the shared-frame registration does not apply. We instead exploit the geometry of the walk-around itself: the camera trajectory forms a rough ring

around the subject, so the centroid of camera centers approximates the object center, and Gaussians beyond a fixed fraction of the mean camera-to-centroid distance are pruned as background or floaters. The remaining Gaussians are imported into SuperSplat for manual cleanup and placed by hand in the scene.

Compositional rendering. The base scene and the transformed objects are merged into a single Gaussian set by concatenating their per-primitive tensors. Each object Gaussian's 3D mean is mapped by $\mathbf{x}' = sR\mathbf{x} + \mathbf{t}$, its rotation quaternion is composed with the transform's rotation, and its log-scale is shifted by $\log s$; spherical harmonic coefficients and opacities are preserved. The merged PLY is rendered identically to a single-capture scene, with no runtime overhead.

A useful side effect of this design is appearance transfer: although the four umbrella-carrying pedestrians were captured under rainy lighting, when rendered into the sunny Highbay scene YOLO still recovers them as the *person* class (Section V-.2).

V. OBSTACLE DETECTION

To enable obstacle-aware navigation, we implemented an RGB-D obstacle detection module that estimates the relative positions of semantic obstacles around the vehicle. YOLO is used to detect objects in the RGB image, while the associated depth image provides range information for each object's 3D position.

The detector currently focuses on three classes: stop sign, person, and traffic cone. For each detection, YOLO outputs a class label, bounding box, and confidence score. Depth values inside the bounding box are extracted, and the object center is projected into 3D using the camera intrinsics. The resulting obstacle position is then transformed into the vehicle frame and published with its semantic class for visualization and downstream planning.

Obstacle positions are estimated only from RGB-D data; LiDAR or point-cloud visualization in RViz is used only as a qualitative reference to check whether the estimated positions are spatially reasonable.

We validate the obstacle detection pipeline in three settings: a simple Gazebo environment, the 3DGS-based visual environment, and real-world rosbag data. These tests check whether the RGB-D pipeline can detect the target obstacle classes and provide planner-compatible obstacle positions.

1) Simple Gazebo Environment: We first tested the detector in a simple Gazebo Highbay scene with a traffic cone and a stop sign placed in front of the vehicle. As shown in Fig. 5, YOLO detects both objects from the RGB image, while the stereo-depth map provides range estimates for their relative positions.



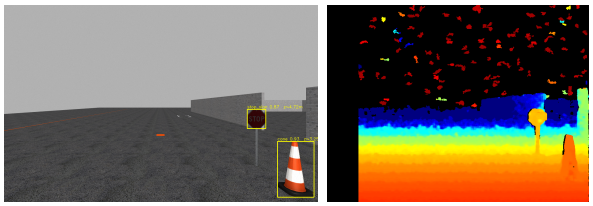
Fig. 4: Object reconstruction and insertion, one example per registration mode (*top*: pedestrian; *bottom*: traffic cone). *Left*: a frame from the walk-around capture. *Middle*: 3DGS reconstruction after background removal, on a black backdrop. *Right*: the object placed in the Highbay scene – the pedestrian via the shared-frame Umeyama fit, the cone (and the sign and cube, not shown) via manual placement in SuperSplat. Insets in the left and right columns show alternative viewpoints.

TABLE II: Distance estimation results in the simple Gazebo environment.

Object	Ground Truth (m)	Estimated (m)	Error (m)
Traffic cone	5.04	4.96	-0.078
Stop sign	6.22	6.00	-0.225

Table II summarizes the distance estimation results. The estimated distances are close to the ground-truth obstacle positions. The traffic cone has an estimation error of -0.078 m, corresponding to approximately -1.5%, while the stop sign has an estimation error of -0.225 m, corresponding to approximately -3.6%. These results indicate that the RGB-D-based pipeline can provide reasonable range estimates for static semantic obstacles in the basic simulation environment.

2) Gaussian-Splatting-Based Environment: We then ran the same detector on the digital-twin Highbay scene populated with the compositional object library



(a) YOLO detections (b) Stereo depth map

Fig. 5: Obstacle detection result in the simple Gazebo environment. The YOLO debug image shows the detected traffic cone and stop sign bounding boxes, and the stereo-depth map provides range information for estimating the relative obstacle positions.

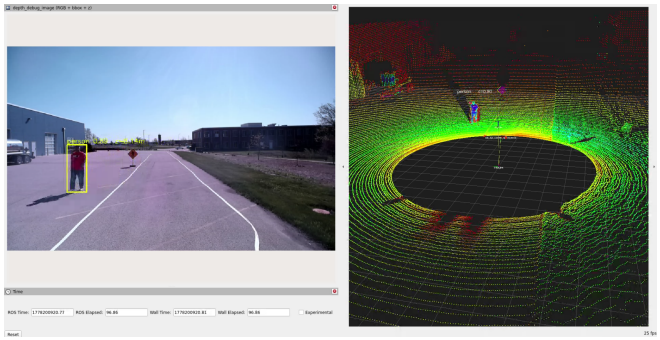


Fig. 6: Qualitative obstacle detection result on real-world rosbag data. The left image shows the YOLO detection result overlaid on the RGB image. The right image shows the corresponding obstacle marker in RViz together with the point-cloud visualization.

described in Section IV-E. Despite the appearance gap between the rainy pedestrian captures and the sunny Highbay scene, YOLO recovers all four pedestrians, the cone, and the stop sign at their inserted poses, providing planner-compatible obstacle positions for the closed-loop experiments in Section VI.

3) Real-World Rosbag Data: We also tested the module on real-world rosbag data. As shown in Fig. 6, the RGB-D pipeline detects a person in the image and projects the estimated obstacle position into RViz. Although LiDAR is not used by the detector, the estimated position is consistent with the point-cloud visualization, providing an initial check that the depth-based projection is reasonable in the vehicle frame.

VI. HIERARCHICAL OBSTACLE AVOIDANCE WITH STANLEY TRACKING

We use a hierarchical navigation architecture that separates path generation from path tracking. The high-level replanner generates a safe local trajectory around obstacles, while the low-level Stanley controller tracks the selected trajectory with steering and speed commands. This separation makes the system easier to validate: the replanner handles safety and feasibility, and the tracker handles real-time execution.

A. Stanley Low-level Controller

Let the vehicle state be $\mathbf{x} = [x, y, \psi, v]^\top$ and the reference path be $\{(x_i^r, y_i^r, \psi_i^r, v_i^r)\}$. At each control step, the nearest reference point is selected, and the front-axle cross-track error e_y and heading error $e_\psi = \mathrm{wrap}(\psi - \psi^r)$ are computed. The steering command follows the Stanley control law

$$\delta = \mathrm{clip}\left(e_\psi + \arctan\frac{k e_y}{k_s + v}, -\delta_{\max}, \delta_{\max}\right), \quad (1)$$

where $k = 0.45$ is the cross-track gain and k_s is a softening constant. The longitudinal command tracks the reference speed v_i^r provided by the replanner.

B. High-level Replanner

The high-level replanner operates in the Frenet frame of the nominal reference path. At each replanning step, it receives the vehicle state and a set of obstacles $\{O_j\}$, where each obstacle is represented by a center $\mathbf{p}_j = (x_j, y_j)$, radius r_j, and velocity $\mathbf{v}_j$. In the current experiments, the obstacles are static, so $\mathbf{v}_j = \mathbf{0}$. The ego state is projected onto the reference path to obtain $(s_0, d_0, \dot{s}_0, \dot{d}_0)$.

Candidate detours are generated by sampling terminal lateral offset d_T, terminal speed v_T, and terminal acceleration a_T over a finite horizon T_h. Each candidate is represented by a quintic lateral trajectory and a quartic longitudinal trajectory:

$$\begin{aligned} d(t) &: (d_0, \dot{d}_0, 0) \to (d_T, 0, 0), \\ s(t) &: (s_0, \dot{s}_0, 0) \to (\dot{s}_T, \ddot{s}_T) = (v_T, a_T). \end{aligned} \quad (2)$$

The Frenet trajectory is mapped back to Cartesian coordinates as

$$\mathbf{p}(t) = \mathbf{p}^r(s(t)) + d(t)\mathbf{n}^r(s(t)),$$

where $\mathbf{p}^r$ and $\mathbf{n}^r$ are the reference position and normal direction.

The planner selects the candidate with the lowest cost

$$J = w_c J_{\mathrm{col}} + w_g J_{\mathrm{goal}} + w_\ell J_{\mathrm{lat}} + w_v J_{\mathrm{spd}} + w_j J_{\mathrm{jerk}} + w_f J_{\mathrm{feas}}.$$

Here, J_{col} penalizes collision with inflated obstacles, J_{goal} encourages forward progress, J_{lat} keeps the path near the nominal lane, J_{spd} tracks the desired speed, J_{jerk} encourages smooth motion, and J_{feas} penalizes violation of speed and curvature limits. The selected trajectory is passed to the Stanley tracker.

C. Simulation Tracking Results

The hierarchical controller was evaluated in the 3DGS-integrated Highbay simulator with multiple static obstacles placed along the route. YOLO detections and the corresponding depth image were used to estimate obstacle positions, which were then passed to the high-level replanner for cost evaluation.

Fig. 7 shows the closed-loop obstacle avoidance result. The first panel shows YOLO detections of a person and a cone in the 3DGS-rendered environment. The person was not present during the original Highbay data collection; instead, it was inserted into the scene as a digital-twin obstacle. The following panels show the resulting detour trajectory, the executed vehicle trajectory, and the tracking error over time. The vehicle tracks the replanned trajectory over the full run, with a mean position error of 0.41 m, an RMSE of 0.66 m, and a maximum error of 2.60 m. These results show that the YOLO-based obstacle detection module can be connected with the replanner and Stanley controller in the realistic 3DGS simulation environment.

D. Hardware Experiments

We also deployed the hierarchical planner on the physical GEM e4 in the Highbay environment. In this experiment, the obstacle pose was assumed to be known so that we could isolate and validate the planning and tracking pipeline. The replanner runs at 2 Hz, and the Stanley controller tracks the selected path at 10 Hz. This assumed obstacle pose can be replaced by the RGB-D obstacle detection module described earlier.

Fig. 8 shows that the vehicle follows the nominal reference on straight segments, performs a lateral detour near the obstacle, and then returns to the reference path. The peak lateral error of 1.22 m corresponds to the commanded detour rather than a tracking failure. Across the full lap, the lateral error has mean 0.27 m and RMSE 0.38 m, while the minimum GPS-to-obstacle surface clearance is 1.84 m. These results indicate that the replanning and tracking pipeline transfers from simulation to the real GEM e4 platform in a controlled static-obstacle setting.

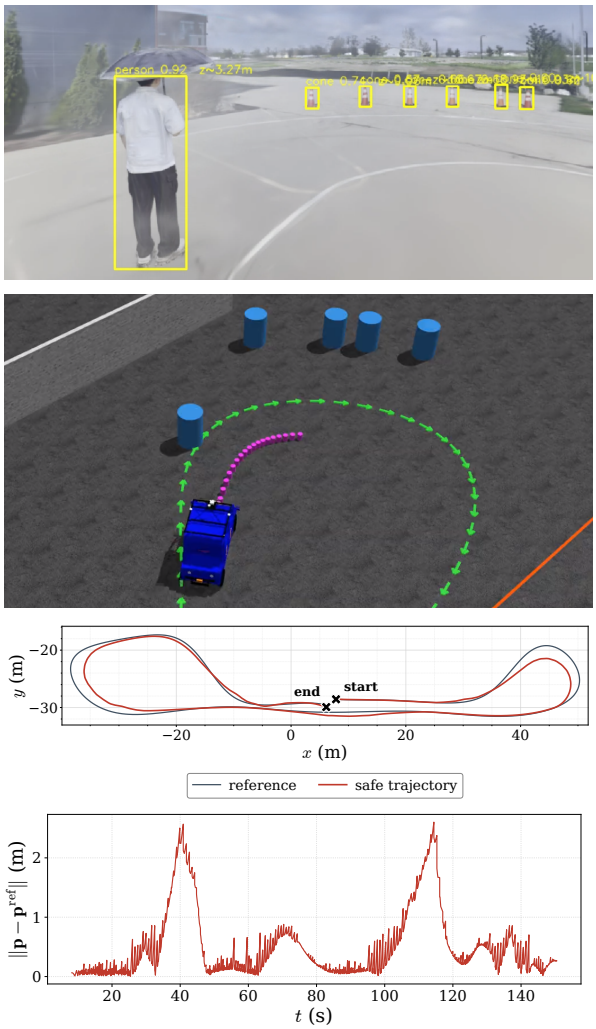


Fig. 7: Hierarchical replanning and Stanley tracking in the 3DGS-integrated Highbay simulator. From top to bottom: YOLO obstacle detection on the 3DGS-rendered camera stream; local detour generation around detected static obstacles in Gazebo; replanned safe trajectory and executed vehicle trajectory; and position tracking error over time.

VII. CONCLUSIONS

This project developed a Highbay navigation pipeline that combines a 3DGS-based digital twin, RGB-D obstacle detection, and hierarchical obstacle-aware planning and control. The 3DGS–Gazebo integration provides more realistic visual simulation, while the RGB-D module converts semantic detections into obstacle positions usable by the planner. The high-level Frenet-frame replanner then generates safe detour trajectories, and the Stanley controller tracks them in real time.

The system was validated in both simulation and hardware. In simulation, YOLO-based obstacle detections were integrated with the replanner, and the vehicle avoided obstacles while tracking the safe trajectory. On the physical GEM e4, the same planning and tracking pipeline was tested with a known obstacle pose, demonstrating that the vehicle could execute a commanded detour while maintaining clearance. Future work will close the full hardware loop by replacing the assumed obstacle pose with RGB-D obstacle detections and testing under more varied obstacle configurations.

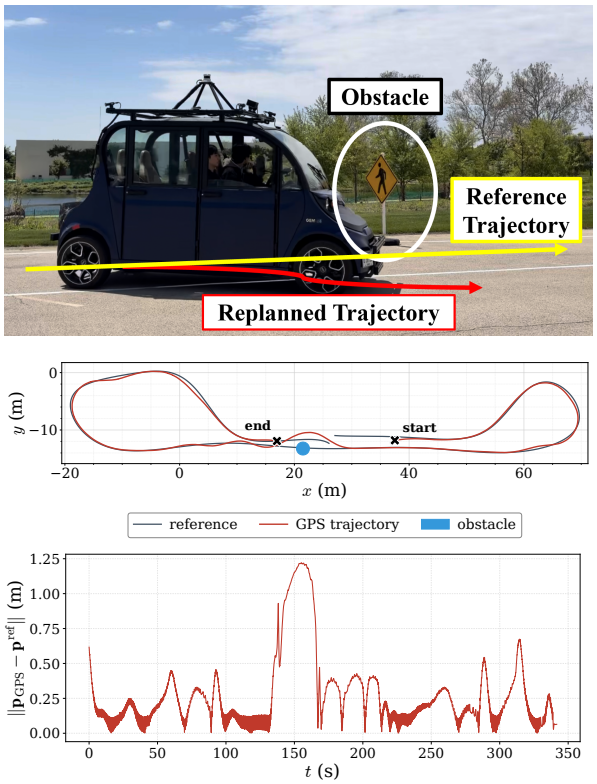


Fig. 8: Hardware validation on the GEM e4. *Top*: Highbay experiment with an obstacle. *Middle*: reference path and GPS trajectory executed by the vehicle. *Bottom*: GPS-to-reference distance over time; the main peak corresponds to the planner-commanded detour.

A second direction is to replace the currently static pedestrian captures with animatable digital humans. Each pedestrian could be rigged to a parametric body model such as SMPL [12], with the per-pedestrian Gaussians bound to the underlying mesh through learned skinning, as in recent animatable Gaussian avatar work [13], [14]. The latter line of work further suggests that new pedestrians could be added from a single image, without re-running the per-object walk-around capture in Section IV-E, while composite dynamic-Gaussian frameworks for driving scenes [15] indicate that such moving agents can be inserted into a 3DGS scene at simulator rates. Driving the body pose with a walking gait would then produce naturally moving pedestrians in the digital twin, enabling closed-loop evaluation of dynamic obstacle avoidance. The high-level replanner already accepts non-zero obstacle velocities $\mathbf{v}_j$ in its Frenet-frame formulation (Section VI), but this capability has so far only been exercised on static obstacles.

References

[1] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3d gaussian splatting for real-time radiance field rendering," *ACM Transactions on Graphics*, vol. 42, no. 4, July 2023.

[2] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," in *Conference on Robot Learning*, 2017, pp. 1–16.

[3] A. Amini, T.-H. Wang, I. Gilitschenski, W. Schwarting, Z. Liu, S. Han, S. Karaman, and D. Rus, "Vista 2.0: An open, data-driven simulator for multimodal sensing and policy learning for autonomous vehicles," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 2419–2426.

[4] C. Gulino, J. Fu, W. Luo, G. Tucker, E. Bronstein, Y. Lu, J. Harb, X. Pan, Y. Wang, X. Chen, *et al.*, "Waymax: An accelerated, data-driven simulator for large-scale autonomous driving research," *Advances in Neural Information Processing Systems*, vol. 36, pp. 7730–7742, 2023.

[5] X. Zhou, Z. Lin, X. Shan, Y. Wang, D. Sun, and M.-H. Yang, "Drivinggaussian: Composite gaussian splatting for surrounding dynamic autonomous driving scenes," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2024, pp. 21 634–21 643.

[6] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[7] M. Bansal, A. Krizhevsky, and A. Ogale, "Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst," *arXiv preprint arXiv:1812.03079*, 2018.

[8] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, *et al.*, "Stanley: The robot that won the darpa grand challenge," *Journal of Field Robotics*, vol. 23, no. 9, pp. 661–692, 2006.

[9] I. Vizzo, T. Guadagnino, B. Mersch, L. Wiesmann, J. Behley, and C. Stachniss, "Kiss-icp: In defense of point-to-point icp–simple, accurate, and robust registration if done the right way," *IEEE Robotics and Automation Letters*, vol. 8, no. 2, pp. 1029–1036, 2023.

[10] F. Dellaert and G. Contributors, "borglab/gtsam," May 2022. [Online]. Available: https://github.com/borglab/gtsam)

[11] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3d gaussian splatting for real-time radiance field rendering," https://github.com/graphdeco-inria/gaussian-splatting, 2023, accessed: May 10, 2026.

[12] M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, and M. J. Black, "SMPL: A skinned multi-person linear model," *ACM Transactions on Graphics*, vol. 34, no. 6, pp. 248:1–248:16, 2015.

[13] Z. Li, Z. Zheng, L. Wang, and Y. Liu, "Animatable gaussians: Learning pose-dependent gaussian maps for high-fidelity human avatar modeling," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[14] Y. Wu, X. Chen, W. Li, S. Jia, H. Wei, K. Feng, J. Chen, Y. Li, A. He, W. Zhang, B. Ni, and W. Zhang, "SinGS: Animatable single-image human gaussian splats with kinematic priors," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.

[15] Y. Xiong, X. Zhou, Y. Wang, D. Sun, and M.-H. Yang, "DrivingGaussian++: Towards realistic reconstruction and editable simulation for surrounding dynamic driving scenes," *arXiv preprint arXiv:2508.20965*, 2025.