

499 A Real-time Deployment Implementation

500 In this section, we provide additional implementation details that enable the real-time deployment of
 501 ComposableNav on robot hardware, including the MPC-based tracking controller and the real-time
 502 replanning mechanism.

503 A.1 Real-time Motion Control

504 To enable ComposableNav to produce motion controls in real-time, we employ a Model Predictive
 505 Path Integral (MPPI) [12], a sampling-based MPC controller, to track the time-dependent trajectories
 506 planned by the composed diffusion models. During navigation, the MPPI uses the differential drive
 507 kinematic model to predict the robot’s future positions and minimizes the difference between these
 508 predictions and the target time-dependent trajectory over a short planning horizon. It also enforces
 509 constraints on acceleration and velocity to ensure feasible and safe control inputs.

510 A.2 Real-time Replanning

511 To enable real-time replanning using only the robot’s onboard compute, we adopt the adaptive on-
 512 line replanning framework introduced in prior work [49], specifically employing the *Replan from*
 513 *Previous Context* method. The key insight behind this approach is that the current planned trajec-
 514 tory is often already close to an optimal solution for replanning. Instead of discarding the existing
 515 trajectory and replanning from scratch, ComposableNav perturbs the current trajectory by applying
 516 a few steps of the diffusion forward process, $q(x_t | x_{t-1})$, and then partially denoises it to generate
 517 an updated trajectory conditioned on the latest observations.

518 In practice, we perform five diffusion steps to add noise to the current planned trajectory according
 519 to the forward process $q(x_t | x_{t-1})$, followed by five denoising steps. Throughout both the forward
 520 and reverse diffusion processes, we fix the states already visited by the robot and only update the
 521 future segments of the trajectory.

522 We introduce two key implementation optimizations to further improve efficiency: First, since all
 523 fine-tuned diffusion models share the same architecture derived from a common base model, we
 524 use PyTorch’s vectorized mapping operation (`vmap`) to stack and execute them in parallel through
 525 batched inference. Second, we leverage PyTorch’s compilation feature (`torch.compile`) to further
 526 accelerate inference. With these optimizations, ComposableNav achieves real-time replanning using
 527 only the robot’s onboard computing resources.

528 B Derivation for Eq. 6

$$\begin{aligned}
 & p(\tau | \phi^{(1)}, o^{(1)}, \dots, \phi^{(k)}, o^{(k)}) & (8) \\
 &= \frac{p(\tau, \phi^{(1)}, o^{(1)}, \dots, \phi^{(k)}, o^{(k)})}{p(\phi^{(1)}, o^{(1)}, \dots, \phi^{(k)}, o^{(k)})} & \triangleleft \text{Bayes' Rule} \\
 &\propto p(\tau, \phi^{(1)}, o^{(1)}, \dots, \phi^{(k)}, o^{(k)}) \\
 &= p(\tau) p(\phi^{(1)}, o^{(1)}, \dots, \phi^{(k)}, o^{(k)} | \tau) & \triangleleft \text{Bayes' Rule} \\
 &= p(\tau) \prod_{i=1}^k p(\phi^{(i)}, o^{(i)} | \tau) & \triangleleft \text{Conditional Independence} \\
 &= p(\tau) \prod_{i=1}^k \frac{p(\tau | \phi^{(i)}, o^{(i)})}{p(\tau)} & \triangleleft \text{Bayes' Rule}
 \end{aligned}$$

C Composing Diffusion Models via Score Function Interpretation

Diffusion models belong to the family of score-based generative models. These models learn to estimate the score function [46, 11], which is defined as the gradient of the log-probability density with respect to the input, i.e., $\nabla_x \log p(x)$. *Intuitively, the score function indicates the direction in which a data point should be moved to increase its likelihood under the data distribution.*

In the case of diffusion models, the denoising network $f_\theta(x_t, t)$ can be interpreted as being proportional to the score function. The denoising process can thus be viewed as iteratively moving the noisy sample x_t in the direction predicted by the model, gradually transforming it into a high-probability sample from the data distribution.

Given this interpretation, composing multiple diffusion models corresponds to computing the sum of their score functions, $\sum_{i=1}^k f_\theta^{(i)}(x_t, t)$, where each $f_\theta^{(i)}$ represents the score from the i -th diffusion model being composed and $\hat{\epsilon}$ is the composed score. The generative process for composing these models becomes [11]:

$$p_\theta(x_{t-1} | x_t) = \mathcal{N}(x_t - \sum_{i=1}^k f_\theta^{(i)}(x_t, t), \sigma_t^2 \mathbf{I}) \quad (9)$$

This process can be understood as guiding the sample toward regions that are simultaneously high-probability under all the models being composed. Hence, the data can be seen as sampling from the joint distribution defined by all the composed diffusion models.

D Motion Primitives

In this work, we consider six commonly used navigation motion primitives, as listed in Tab. 1. In the following subsections, we present the pseudocode that outlines how we define the specification $\phi^{(i)}$ for each motion primitive using heuristic rule-based functions.

Table 1: Instruction Specifications for Navigation Motion Primitives

Motion Primitive (MP)	Instruction Specification
Pass a person from the left (L)	The robot should pass the person from the left side.
Pass a person from the right (R)	The robot should pass the person from the right side.
Follow behind a person (F)	The robot should stay in a specific region behind the person relative to the person’s position.
Yield to a person (Y)	The robot should not cross the region in front of the person.
Walk through a region (W)	The robot’s trajectory should overlap with the specified region.
Avoid walking through a specified region (A)	The robot’s trajectory should not overlap with the specified region, which may be defined either by a terrain feature or by a person’s location.

D.1 Pass a person from the left

Pseudocode 1 Motion Primitive $\phi^{(L)}$

```

1 def criteria_left ( $\tau$ ,  $O$ ):
2     time_at_each_waypoint = extract_time_at_each_waypoint( $\tau$ )
3     for t in time_at_each_waypoint:
4         region = extract_region_left_of_obs( $O$ , t)
5         robot_position = extract_position( $\tau$ , t)
6         if region.contains(robot_position):
7             return 1
8     return 0

```

550 D.2 Pass a person from the right

Pseudocode 2 Motion Primitive $\phi^{(R)}$

```

1 def criteria_left ( $\tau$ ,  $O$ ):
2     time_at_each_waypoint = extract_time_at_each_waypoint( $\tau$ )
3     for t in time_at_each_waypoint:
4         region = extract_region_right_of_obs( $O$ , t)
5         robot_position = extract_position( $\tau$ , t)
6         if region.contains(robot_position):
7             return 1
8     return 0

```

551 D.3 Follow behind a person

552 For this primitive, we consider a period of time to evaluate whether the robot has successfully fol-
553 lowed the human, which is defined as the final few seconds before the robot reaches the goal.

Pseudocode 3 Motion Primitive $\phi^{(F)}$

```

1 def criteria_left ( $\tau$ ,  $O$ ):
2     time_at_each_waypoint = extract_time_at_each_waypoint( $\tau$ )
3     time_period = get_relevant_time_period(time_at_each_waypoint)
4     for t in time_period:
5         region = extract_region_behind_obs( $O$ , t)
6         robot_position = extract_position( $\tau$ , t)
7         if not region.contains(robot_position):
8             return 0
9     return 1

```

554 D.4 Yield to a person

Pseudocode 4 Motion Primitive $\phi^{(Y)}$

```

1 def criteria_left ( $\tau$ ,  $O$ ):
2     time_at_each_waypoint = extract_time_at_each_waypoint( $\tau$ )
3     for t in time_at_each_waypoint:
4         region = extract_region_in_front_of_obs( $O$ , t)
5         robot_position = extract_position( $\tau$ , t)
6         if region.contains(robot_position):
7             return 0
8     return 1

```

555 D.5 Walk through a region

Pseudocode 5 Motion Primitive $\phi^{(W)}$

```

1 def criteria_left ( $\tau$ ,  $O$ ):
2     time_at_each_waypoint = extract_time_at_each_waypoint( $\tau$ )
3     for t in time_at_each_waypoint:
4         region = extract_region( $O$ , t)
5         robot_position = extract_position( $\tau$ , t)
6         if region.contains(robot_position):
7             return 1
8     return 0

```

556 D.6 Avoid walking through a region

Pseudocode 6 Motion Primitive $\phi^{(A)}$

```
1 def criteria_left ( $\tau$ ,  $O$ ):  
2     time_at_each_waypoint = extract_time_at_each_waypoint( $\tau$ )  
3     for t in time_at_each_waypoint:  
4         region = extract_region( $O$ , t)  
5         robot_position = extract_position( $\tau$ , t)  
6         if region.contains(robot_position):  
7             return 0  
8     return 1
```

557 E Training Diffusion Model

558 E.1 Model Design

559 We build upon a publicly available diffusion model implementation¹. Our denoising network, f_θ ,
560 consists of a 1D UNet architecture augmented with a context encoder. Each observation—whether
561 a dynamic human or a specific region—along with the goal, is first encoded using a multilayer
562 perceptron (MLP). The dynamic human is represented as a predicted future trajectory, estimated
563 under a constant velocity assumption. In contrast, a region is represented as a rectangle defined by
564 the positions of its four corners. These embeddings are then passed through a vision transformer to
565 generate context features, following the approach introduced in prior work [35].

566 E.2 Training

567 To train the base model, we collected trajectories across three types of environments: (1) collision-
568 free trajectories in dynamic settings, (2) trajectories that avoid specified regions, and (3) trajectories
569 that intentionally traverse specific regions. The first two types were generated in simulation by
570 randomly placing obstacles and planning trajectories to avoid them. For the third type, we sampled
571 trajectories from the first two types, randomly selected a region that each trajectory passes through,
572 and re-labeled this region as the observation to form training pairs.

573 We collected approximately 2 million collision-free trajectories and trained the denoising network
574 for 2000 epochs, using a learning rate of 2×10^{-4} and a dropout rate of 0.1. Training followed
575 the classifier-free guidance approach [41], where the model was conditioned on a null context (rep-
576 resented by zero vectors) with a probability of 20%, instead of using features extracted from the
577 context encoder. Following prior work [32], the diffusion model performs a total of 25 denois-
578 ing steps using an exponential noise schedule, generating a trajectory composed of a sequence of
579 fixed-length, time-dependent waypoints.

580 To fine-tune the base denoising network for each motion primitive, we adapted the DDPO imple-
581 mentation². For simplicity, we replaced the original vision-language model (VLM)-based reward
582 function with a heuristic-based one, designed according to the specific definitions of each motion
583 primitive in this work. During each training epoch, we generated 32 different environments and
584 trained the model for a total of 1000 epochs. A noteworthy observation from our experiments is that
585 a significantly lower learning rate greatly enhances training performance. Based on this finding, we
586 adopted a learning rate of 1×10^{-6} , which is also consistent with results reported in the literature [6].

¹<https://github.com/lucidrains/denoising-diffusion-pytorch>

²<https://github.com/kvabblack/ddpo-pytorch>

F Additional Simulation Experiment Details

F.1 Simulation ComposableNav Setup

We evaluate ComposableNav in a $20\text{ m} \times 20\text{ m}$ 2D simulation arena, where dynamic humans are modeled as spheres and regions are modeled as rectangles. For each instruction, 20 environments are randomly initialized, assigning initial positions and speeds to the entities based on the specific requirements of the instruction. The simulation operates at a control frequency of $\Delta t = 0.1\text{ s}$, and each episode lasts for a maximum of 300 timesteps, equivalent to 30.0 seconds.

F.2 Baseline Setup

In this work, we consider three baseline methods: VLM-Social-Nav [28], CoNVOI [27], and BehAV [29]. These baselines fall into two categories: the first two treat the VLM as a black-box policy that proposes a target action (e.g., next waypoint or velocity) for a geometric planner to track, while the third computes composable cost maps for planning.

None of these baseline methods is explicitly designed to solve the problem considered in this work. Therefore, we adapt them for our experimental setup. For VLM-Social-Nav and CoNVOI, we use the latest GPT-4.1 model and disregard the high inference latency associated with invoking a remote VLM and focus on evaluating their prediction accuracy. Additionally, we modify these approaches by providing annotated screenshots of the simulated scenes and prompting the VLMs for reasoning.

For BehAV, whose core idea is to create composable costmaps and plan trajectories over them (similar to Voxposer [48]), we simplify the setup by abstracting away the segmentation vision model. Instead, we provide BehAV with ground-truth annotations obtained directly from the simulation environment and evaluate solely how well costmap-based planning enables instruction following.

For each baseline method, we conducted a grid search over various combinations of motion planner hyperparameters—specifically, different weights of the constituent cost functions—using a small tuning set. We then selected the hyperparameters that achieved the highest overall success rates.

F.3 Supplementary Quantitative Results

F.3.1 Learning Motion Primitives

Beyond demonstrating that the RL fine-tuning procedure enables the learning of effective motion primitives, we further analyze results obtained from comparing the pre-trained and fine-tuned models. As shown in Tab. 2, the pre-trained model, which is trained to generate collision-free, goal-reaching trajectories, consistently achieves these objectives across all evaluated motion primitives. In particular, the pre-trained model achieves a 100% success rate for the “Avoid walking through a region” motion primitive. This outcome is anticipated, as avoiding designated regions is closely aligned with the collision avoidance objective emphasized during the pre-training phase.

Table 2: Comparison of Fine-tuned and Pre-trained Diffusion Models for Representing Motion Primitives.

MP	Pre-trained Model				Fine-tuned Model			
	SR(%) \uparrow	IA(%) \uparrow	CF(%) \uparrow	GR(%) \uparrow	SR(%) \uparrow	IA(%) \uparrow	CF(%) \uparrow	GR(%) \uparrow
L	44.0	44.0	100.0	100.0	100.0	100.0	100.0	100.0
R	37.0	37.0	100.0	100.0	100.0	100.0	100.0	100.0
F	27.0	27.0	100.0	100.0	99.0	100.0	100.0	99.0
Y	48.0	48.0	100.0	100.0	100.0	100.0	100.0	100.0
W	34.0	34.0	100.0	100.0	100.0	100.0	100.0	100.0
A	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0

F.3.2 Composing Motion Primitives

We present the quantitative results of 24 motion primitive combinations in our simulation testbed, as detailed in Tab. 3. Each combination (e.g., “L+R”) corresponds to a specific natural language instruction (e.g., “Pass person 1 from the left and person 2 from the right”), which maps to a distinct

robot motion trajectory illustrated in Fig. 1. Given the similarity between the primitives “pass a person from the left” and “pass a person from the right”—which differ only in direction—we unify them under a general instruction category: “Pass a person” (denoted as P) for motion composition purposes. Accordingly, we evaluate our method, ComposableNav, on its ability to handle both left and right variants using a single instruction specification.

The quantitative results are summarized in Tab. 3. Across all testbed scenarios, ComposableNav consistently outperforms all baseline methods in terms of success rate, particularly as the number of motion primitives in an instruction increases. While baseline methods perform reasonably with simple instructions, their performance deteriorates notably with more complex instruction sets.

Methods relying on Vision-Language Models (VLMs) as black-box policies perform particularly poorly. These models are not designed for such navigation tasks and often fail to maintain planning consistency, especially as instruction complexity grows. Similarly, BehAV performs adequately with one or two motion primitives but suffers as composition complexity increases. In addition, BehAV has the lowest goal-reaching rate, which suggests that such a costmap-based method tends to get trapped in local minima and is unable to complete tasks within the allotted time.

Furthermore, baseline methods exhibit high variance in performance across different instruction combinations. In many cases, they fail to generate any viable, instruction-following trajectory. In contrast, ComposableNav—*despite not being explicitly trained for any possible instruction composition*—demonstrates strong generalization capabilities and consistently higher success rates across a wide range of scenarios.

To complement the quantitative analysis, we provide a qualitative illustration in Fig. 7. Here, we observe that while VLM-based methods may initially steer the robot in the correct direction, they lack the responsiveness and consistency needed for sustained instruction following. These methods often begin to avoid regions or yield to pedestrians, but then fail to complete subsequent specifications. The BehAV method often gets stuck in local minima and fails to reach the goal within the allowed time. In contrast, ComposableNav effectively produces instruction-aligned behaviors, simultaneously satisfying all given specifications—such as avoiding restricted regions and yielding to oncoming pedestrians.

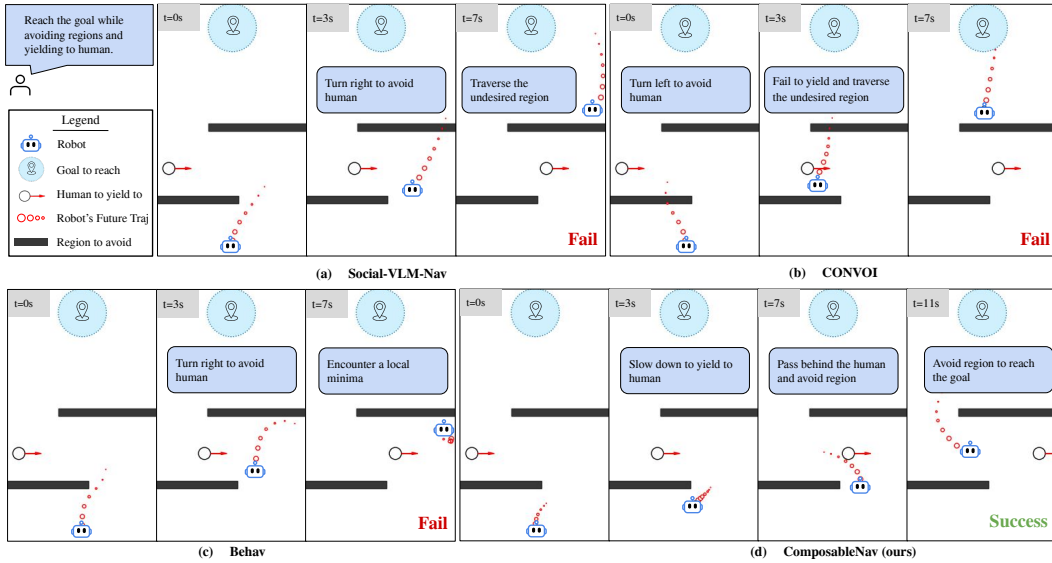


Figure 7: Qualitative Simulation Results

Table 3: Detailed Simulation Evaluation Results

#	Combination	VLM-based Policy								Compose Costmaps				Compose Primitives			
		VLM-Social-Nav (%)				Convoy (%)				Behav (%)				ComposableNav (ours) (%)			
		SR↑	IA↑	CF↑	GR↑	SR↑	IA↑	CF↑	GR↑	SR↑	IA↑	CF↑	GR↑	SR↑	IA↑	CF↑	GR↑
1 Primitive	L	55.0	55.0	100.0	100.0	70.0	75.0	95.0	100.0	65.0	65.0	100.0	100.0	100.0	100.0	100.0	100.0
	R	55.0	55.0	100.0	100.0	70.0	75.0	95.0	100.0	65.0	65.0	100.0	100.0	100.0	100.0	100.0	100.0
	F	5.0	5.0	100.0	100.0	0.0	0.0	95.0	100.0	70.0	75.0	100.0	95.0	99.0	100.0	100.0	99.0
	Y	25.0	25.0	95.0	100.0	40.0	40.0	95.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
	W	0.0	0.0	100.0	100.0	55.0	55.0	100.0	100.0	55.0	55.0	100.0	100.0	100.0	100.0	100.0	100.0
	A	0.0	0.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
Overall		23.3	23.3	99.2	100.0	55.8	57.5	96.7	100.0	75.8	76.7	100.0	99.2	99.8	100.0	100.0	99.8
2 Motion Primitives	L+R	5.0	5.0	100.0	100.0	0.0	0.0	100.0	100.0	0.0	0.0	100.0	100.0	33.0	34.0	86.0	100.0
	P+F	10.0	10.0	100.0	100.0	5.0	5.0	90.0	100.0	0.0	0.0	90.0	100.0	85.0	85.0	99.0	100.0
	Y+P	20.0	20.0	100.0	100.0	55.0	55.0	95.0	100.0	90.0	90.0	100.0	100.0	90.0	93.0	97.0	100.0
	Y+F	5.0	5.0	95.0	100.0	0.0	0.0	80.0	95.0	85.0	85.0	100.0	100.0	85.0	85.0	100.0	100.0
	W+P	0.0	0.0	100.0	100.0	20.0	20.0	95.0	100.0	5.0	5.0	100.0	100.0	60.0	61.0	97.0	100.0
	W+Y	0.0	0.0	100.0	100.0	35.0	35.0	95.0	100.0	40.0	40.0	100.0	100.0	99.0	99.0	100.0	100.0
	A+P	0.0	0.0	90.0	100.0	65.0	65.0	100.0	100.0	50.0	50.0	100.0	95.0	67.0	68.0	99.0	100.0
	A+F	0.0	0.0	100.0	100.0	0.0	0.0	100.0	100.0	35.0	40.0	100.0	35.0	85.0	85.0	100.0	100.0
	Overall	5.0	5.0	98.1	100.0	22.5	22.5	94.4	99.4	38.1	38.8	98.8	91.2	75.5	76.2	97.2	100.0
3 Motion Primitives	P+F+Y	0.0	0.0	100.0	100.0	5.0	5.0	85.0	100.0	0.0	0.0	100.0	100.0	30.0	34.0	91.0	100.0
	P+F+W	0.0	0.0	100.0	100.0	5.0	5.0	80.0	100.0	0.0	0.0	100.0	95.0	58.0	61.0	92.0	100.0
	P+Y+W	0.0	0.0	100.0	100.0	20.0	20.0	90.0	100.0	5.0	5.0	100.0	90.0	38.0	42.0	92.0	100.0
	W+W+Y	0.0	0.0	100.0	100.0	10.0	10.0	90.0	100.0	5.0	5.0	100.0	50.0	87.0	87.0	100.0	100.0
	A+A+Y	0.0	0.0	90.0	100.0	5.0	5.0	100.0	95.0	15.0	75.0	100.0	15.0	86.0	86.0	99.0	100.0
	A+W+Y	0.0	0.0	100.0	100.0	20.0	20.0	100.0	100.0	10.0	10.0	100.0	60.0	0.0	0.0	99.0	100.0
	A+P+F	0.0	0.0	100.0	100.0	15.0	15.0	95.0	100.0	0.0	0.0	100.0	90.0	93.0	95.0	94.0	100.0
	A+W+F	0.0	0.0	100.0	40.0	5.0	5.0	95.0	40.0	45.0	50.0	90.0	85.0	77.0	77.0	98.0	100.0
	Overall	0.0	0.6	98.8	92.5	10.6	10.6	91.9	91.9	10.0	18.1	98.8	80.6	58.6	60.2	95.6	100.0
4 Motion Primitives	A+W+F+Y	0.0	0.0	90.0	60.0	0.0	0.0	90.0	60.0	30.0	30.0	95.0	60.0	33.0	35.0	94.0	98.0
	A+W+F+P	0.0	5.0	90.0	50.0	0.0	0.0	85.0	50.0	0.0	0.0	95.0	10.0	59.0	61.0	72.0	100.0
	A+W+A+Y	0.0	0.0	100.0	50.0	35.0	35.0	100.0	70.0	2.0	20.0	100.0	25.0	46.0	46.0	76.0	100.0
	W+W+Y+A	0.0	0.0	90.0	100.0	5.0	5.0	65.0	100.0	0.0	0.0	85.0	100.0	71.0	78.0	86.0	100.0
	W+P+Y+A	0.0	0.0	70.0	100.0	25.0	25.0	75.0	100.0	5.0	5.0	70.0	100.0	28.0	34.0	78.0	100.0
	W+P+F+A	0.0	0.0	80.0	80.0	0.0	0.0	75.0	100.0	0.0	0.0	80.0	100.0	24.0	31.0	81.0	100.0
	P+F+Y+A	0.0	0.0	80.0	100.0	0.0	0.0	85.0	100.0	0.0	0.0	100.0	100.0	18.0	23.0	82.0	95.0
	A+W+Y+A	0.0	0.0	60.0	100.0	0.0	0.0	75.0	100.0	0.0	0.0	90.0	55.0	0.0	0.0	92.0	99.0
	Overall	0.0	0.6	82.5	80.0	8.1	8.1	81.2	84.4	6.9	6.9	89.4	68.8	34.9	38.5	82.6	99.0

G Additional Robot Deployment Details

G.1 Real-World Deployment Setup

We deploy ComposableNav on a Clearpath Jackal robot equipped with a Zed 2i camera for human tracking and an Ouster LiDAR for generating point cloud data to create an obstacle map for collision avoidance, as shown in Fig. 8. For localization, we apply ENML [50], which provides robust position estimation. The system is built using ROS and consists of a navigation stack with three main modules: a perception module, a diffusion planning module, and an MPC motion planning module. The perception module leverages Zed’s internal human-tracking algorithm to detect and track humans while using Ouster’s point cloud data to detect obstacles for collision avoidance. The diffusion planning module loads diffusion models corresponding to various motion primitives and composes the appropriate models based on instructions and environmental observations. The MPC motion planning module tracks the time-dependent trajectory generated by the diffusion planner and computes real-time motion control commands. All computations are executed entirely onboard, utilizing an Intel i7-9700TE CPU and an NVIDIA RTX A2000 GPU.



Figure 8: Robot Setup.

Table 4: ComposableNav Inference Latency on Robot Hardware

Latency	# of Composed MPs			
	1	2	3	4
Initial Plan (s) ↓	0.144 ± 0.014	0.243 ± 0.009	0.329 ± 0.014	0.413 ± 0.010
Replan (s) ↓	0.027 ± 0.002	0.036 ± 0.004	0.049 ± 0.003	0.060 ± 0.005

G.2 Quantitative Experiment Details

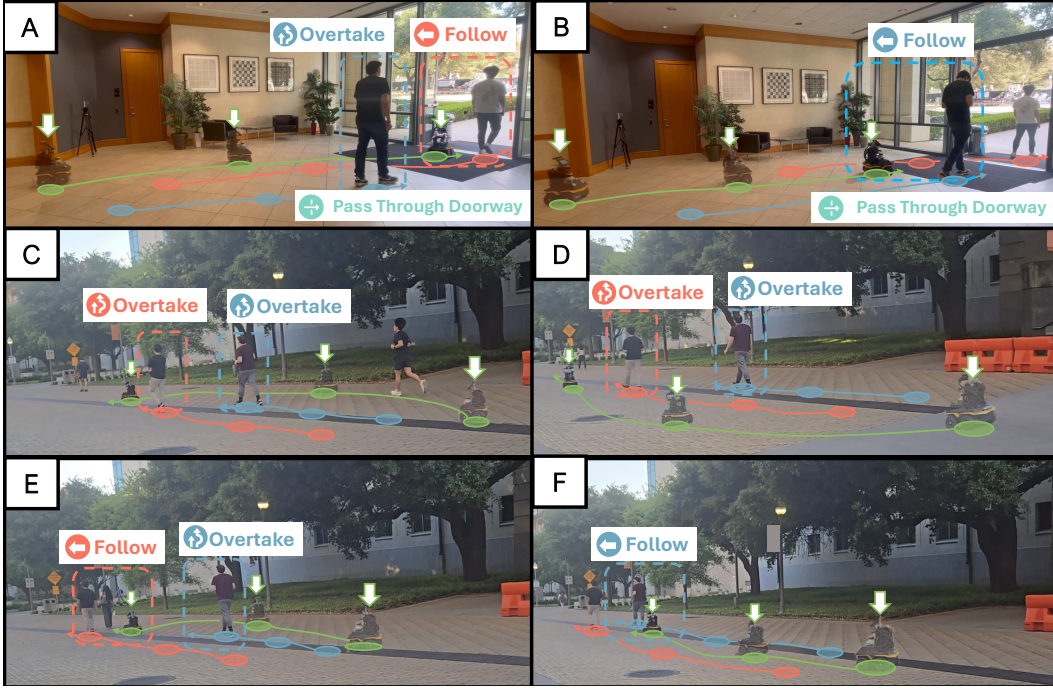


Figure 9: Quantitative Experiment Illustration

We evaluate ComposableNav in two common real-world scenarios: navigating through a narrow doorway and walking outdoors in an open environment, using a total of six instructions—two for the doorway scenario and four for the outdoor scenario, as illustrated in Fig. 9.

Doorway Instructions:

- **Instruction 1:** “Go through the doorway after the person wearing a black shirt and before the person wearing a white shirt” (Fig. 9A)
- **Instruction 2:** “Go through the doorway before the person wearing a black shirt” (Fig. 9B)

Outdoor Instructions:

- **Instruction 3:** “Pass both the person wearing a black shirt and a maroon shirt from the right side of the road” (Fig. 9C)
- **Instruction 4:** “Pass both the person wearing a black shirt and a maroon shirt from the left side of the road” (Fig. 9D)
- **Instruction 5:** “Pass the person wearing a maroon shirt and follow the person wearing a black shirt” (Fig. 9E)
- **Instruction 6:** “Follow the person wearing a black shirt” (Fig. 9F)

Each instruction was tested over 10 trials, and we report the corresponding success rates in Fig. 9. These experiments demonstrate how ComposableNav enables customizable robot behavior that aligns with human preferences. For instance, in the doorway scenario, a human operator might prefer the robot to be polite by following the person in a black shirt, or alternatively, instruct it to hurry and enter after the person in a white shirt. In the outdoor scenario, preferences may include keeping to a specific side of the road to follow the human flow, or adjusting the robot's pace to follow a specific individual, such as someone in a maroon or black shirt.

G.3 Additional Real-World Deployment Demo

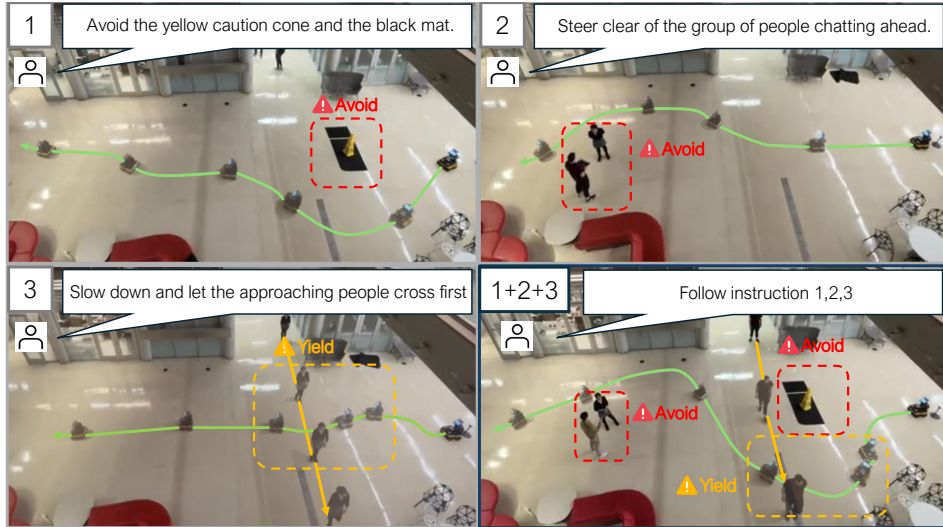
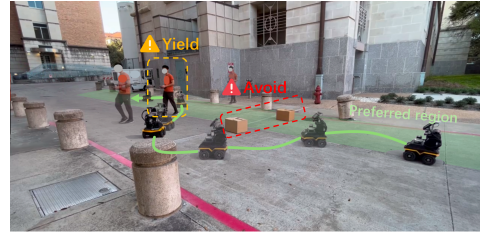


Figure 10: Real-World Composition Experiment



(a) Yield to the person in the white coat and follow the person wearing a dark hoodie.



(b) Avoid walking between the boxes, yield to the person, and stay on the right side of the road.

G.4 Deployment Failure Case Analysis

We conducted a qualitative analysis of the failure cases observed when deploying ComposableNav on the robot and identified two common issues. The first issue stems from human tracking errors. Since both the robot and the human are in continuous motion, the person may temporarily exit the camera's field of view—particularly when the robot turns—causing the system to lose track of them, even if they later reappear. While we applied a simple nearest-neighbor heuristic to reassign the human based on previous tracking data, occasional failures still occur, where the robot is unable to reliably re-identify the person. The second issue arises during replanning. We observed that the newly generated plan can sometimes diverge significantly from the original one. This can lead the MPPI controller to issue large acceleration or deceleration commands, resulting in jerky movements. Consequently, the robot may overshoot its intended state and struggle to stay on the planned path. We plan to address these issues in future work.