

# GC-VLN: Instruction as Graph Constraints for Training-free Vision-and-Language Navigation

## Supplementary Material

Anonymous Author(s)

Affiliation

Address

email

### 1 A Overview

2 This supplementary material is organized as follows:

- 3 • Section B provides the algorithm for the overall pipeline of GC-VLN.
- 4 • Section C provides the details of the approach.
- 5 • Section D provides the details of the hardware used in real-world experiments.
- 6 • Section E reports the results of additional ablation experiments.
- 7 • Section F shows visualization results of failure cases.
- 8 • Section G details the prompts for LLM.

### 9 B Pipeline of GC-VLN

10 In Algorithm 1, we provide an algorithm diagram of GC-VLN.

---

**Algorithm 1** Overall Pipeline of GC-VLN

---

**Require:** Instruction  $\mathcal{I}$ , Observation  $\mathcal{O}$

**Ensure:** Goal Position  $(x, y)$

$\mathcal{G} \leftarrow \text{DecomposeInstruction}(\mathcal{I})$

$\mathcal{L} \leftarrow \text{ConstraintLibrary}()$

$\mathcal{C} \leftarrow \text{ConstructGraphConstraint}(\mathcal{G}, \mathcal{L})$

$\mathcal{T} \leftarrow \text{NewNavigationTree}()$

**while** True **do**

$\{v_i \mid i = 1, 2, \dots, k\} \leftarrow \text{ConstraintSolver}(\mathcal{C}, \mathcal{O})$

$\mathcal{T} \leftarrow \text{UpdateNavigationTree}(\mathcal{T}, \{v_i \mid i = 1, 2, \dots, k\})$

**if**  $\{v_i \mid i = 1, 2, \dots, k\} == \Phi$  **then**

$v \leftarrow \text{Backtrack}(\mathcal{T})$

**else**

$v \leftarrow \text{GetWaypoint}(\{v_i \mid i = 1, 2, \dots, k\})$

**end if**

    Go to  $v(x, y)$

**if** isFinalWaypoint( $v$ ) **then**

        Stop at  $v(x, y)$

**end if**

**end while**

---

## C Details of Approach

### C.1 Instruction Decomposition

LLM is prompted to decompose the linguistic instruction  $\mathcal{I}$ . The decomposition of  $\mathcal{I}$  must meet several rules: 1. Each stage must contain exactly ONE position change. Rotating alone without movement is not a stage. 2. The direction of a stage is equal to the direction of the line from the start to the end of the stage. 3. Each stage comprises two attributes: position and a list of nodes. Each node consists of two attributes: name and position. 4. "waypoint\_position" must belong to one of "front", "right", "left", "back", "unknown". 5. "object\_position" must belong to one of the types "right", "left", "through", "weave", "pass", "near", "back".

### C.2 Constraint Library

As illustrated in Figure 3 of the main text, based on the description in the instruction  $\mathcal{I}$ , the angle and distance information within a constraint can be identified, allowing the constraints to be classified into six types. A unary constraint involves only two nodes, while a multi-constraint involves three or four nodes. For example, the instruction "move forward 3 meters to the left" belongs to type 2, where both  $u$  (blue node) and  $v$  (green node) are waypoints. The instruction "move forward through a door" corresponds to type 3, where  $u_1$  (blue node) and  $v$  (green node) are waypoints, while  $u_2$  (gray node) is a door node. The instruction "walk forward through the space between two chairs" falls under type 6, where  $u_1$  (blue node) and  $v$  (green node) are waypoints, and  $u_2$  and  $u_3$  (gray nodes) are chair nodes.

We formulate the sub-constraints for all six types of constraints. Each type of constraint includes two possible sub-constraints: the angle constraint  $c^a$  and distance constraint  $c^d$ . The constraint types that include angular constraints are: 1, 2, 3, and 6. The constraint types that include distance constraints are: 2, 3, 5, and 6. We formulate the constraint as:

$$c = (\mathbf{1}^a c^a, \mathbf{1}^d c^d), \quad \text{sum}(c) = \mathbf{1}^a c^a + \mathbf{1}^d c^d, \quad \min(c) = \min(\mathbf{1}^a c^a, \mathbf{1}^d c^d) \quad (1)$$

where  $\mathbf{1}^a$  and  $\mathbf{1}^d$  denote indicator functions representing the presence of angle and distance sub-constraints, respectively.

For types 1, 2, 4, and 5, the  $c^a$  and  $c^d$  are:

$$c^a(v | u) = \cos(\Delta\phi) \|v - u\| - [\|v - u\| - (v - u) \cdot (\cos \phi, \sin \phi)] \quad (2)$$

$$c^d(v | u) = \Delta d^2 - (\|v - u\| - d)^2 \quad (3)$$

For type 3, the  $c^a$  and  $c^d$  are:

$$c^a(v | u_1, u_2) = \cos(\Delta\phi) \|v - u_2\| - [\|v - u_2\| - (v - u_2) \cdot (\cos \phi, \sin \phi)] \quad (4)$$

$$c^d(v | u_1, u_2) = \Delta d^2 - (\|v - u_2\| - d)^2 \quad (5)$$

In type 1, 2, 3, 4, and 5, if the angle and distance are not explicitly specified, then  $\Delta\phi = 45^\circ$ ,  $d = 1.5m$  and  $\Delta d = 1.5m$ .

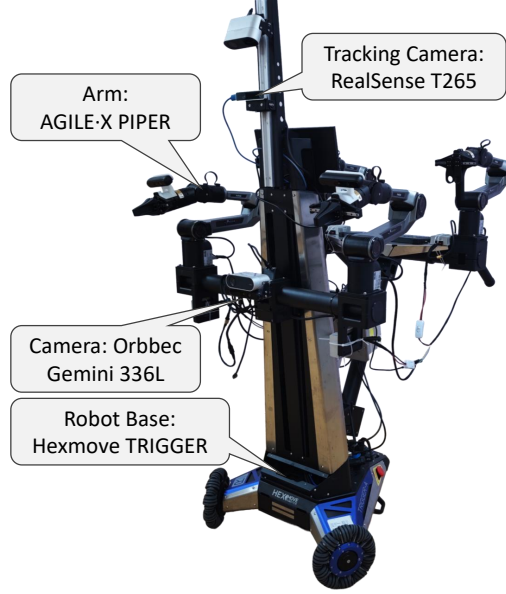


Figure 1: The robot employed for conducting the real-world experimental deployments.

For type 6, the  $c^a$  and  $c^d$  are:

$$c^a(v \mid u_1, u_2, u_3) = \cos(\Delta\phi) \|v - u_1\| - [\|v - u_1\| - (v - u_1) \cdot (\cos\phi, \sin\phi)] \quad (6)$$

$$c^d(v \mid u_1, u_2, u_3) = \Delta d^2 - (\|v - u_1\| - d)^2 \quad (7)$$

$$\phi = \arg\left(\frac{u_2 - u_1}{|u_2 - u_1|} + \frac{u_3 - u_1}{|u_3 - u_1|}\right) \quad (8)$$

$$\Delta\phi = \frac{1}{2} \arccos\left(\frac{(u_2 - u_1) \cdot (u_3 - u_1)}{|u_2 - u_1| \cdot |u_3 - u_1|}\right) \quad (9)$$

### C.3 Topological Sort

To determine the order of nodes in graph constraint  $\mathcal{K}$ , we perform topological sort on the  $\mathcal{K}$ , which ensures that the parent node  $u$  of each constraint  $c(v \mid u)$  is always positioned ahead of its corresponding child node  $v$ . The topological sort satisfies several conditions: 1. The parent node must appear before its child node. 2. Among multiple child nodes of a single node, object nodes must precede waypoint nodes. 3. The order of multiple object child nodes under a single parent node must be consistent with the order in which they are mentioned in the instruction  $\mathcal{I}$ .

## D Hardware Details

We present the hardware details of the robot employed in our real-world experiments in Figure 1. The robot base is the TRIGGER platform developed by Hexmove. A monocular RGB-D camera serves as our input for observations, which is Orbbec Gemini 336L. Our pose input is provided by the RealSense T265 tracking camera. We utilize the PIPER robot arm from AGILE-X for object manipulation.

## E Ablation Study

In Table 1, we report results of additional ablation experiments on hyperparameters of GC-VLN and RxR-CE benchmark. For R2R-CE, we ablate the angle tolerance  $\Delta\phi$  of the constraint type 1, 2, 3, 6, and the distance baseline  $d$  of the all constraint types. For RxR-CE, the settings of ablation are the same as those in the main text.

Table 1: Effect of angle tolerance and distance baseline on R2R-CE. Effect of constraint and constraint solver on RxR-CE.

Constraint Condition on Angle (R2R-CE)					Constraint Condition on Distance (R2R-CE)				
Angle Tolerance $\Delta\phi$	NE	OSR	SR	SPL	Distance Baseline $d$	NE	OSR	SR	SPL
30°	10.2	32.6	30.1	15.9	0.8m	10.6	35.7	28.5	14.7
45° (Ours)	<b>7.3</b>	<b>41.8</b>	<b>33.6</b>	<b>16.3</b>	1.5m (Ours)	<b>7.3</b>	<b>41.8</b>	<b>33.6</b>	<b>16.3</b>
75°	10.2	33.8	31.5	15.6	2.5m	10.1	39.0	30.5	14.0

Graph Constraint (RxR-CE)					Constraint Solver (RxR-CE)				
Method	NE	OSR	SR	SPL	Method	NE	OSR	SR	SPL
Relax constraints in $\mathcal{K}$	10.8	31.5	24.5	9.9	Random Constraint Solver	11.1	21.9	18.0	10.6
<b>Full Approach</b>	<b>8.8</b>	<b>44.4</b>	<b>33.8</b>	<b>13.8</b>	<b>Full Approach</b>	<b>8.8</b>	<b>44.4</b>	<b>33.8</b>	<b>13.8</b>

## F Visualization of Failure Cases

In Figure 2, we further provide visualizations of failure cases for better understanding.

## G Prompts

We provide the prompt used for instruction decomposition in GC-VLN.

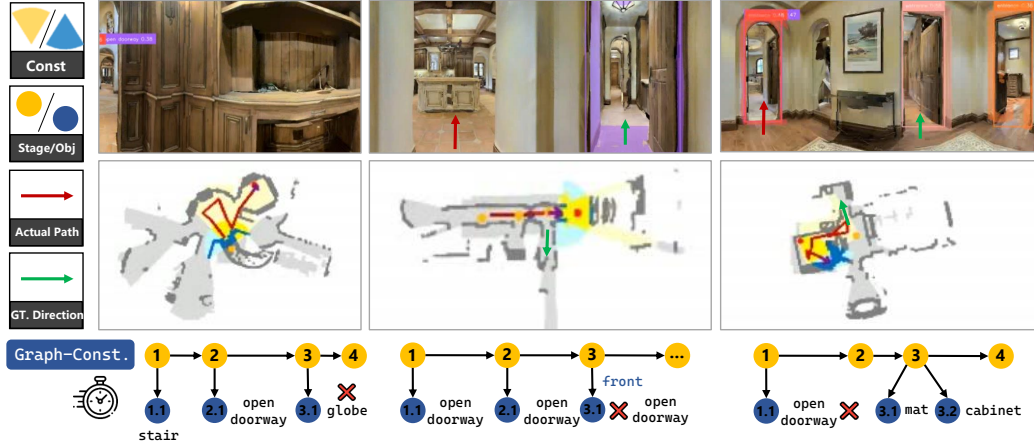


Figure 2: Visualization of three failure cases. In the first case, robot fails to locate the globe. In the second case, the robot mistakes "passing through the door to the right" for "forward" during the construction of the graph constraint. In the third case, the robot initially selects an incorrect path, and coincidentally encounters a correct object, which prevents timely backtracking.

Parse navigation instructions into movement stages in JSON format: {

```
"stage 1": {
  "waypoint position": <position>,
  "connected nodes": [
    {"node": <object>, "object position": <position>},
    ...
  ]
},
...
```

Input Instruction: **<Instruction>**

Rules:

Each stage has one position change. The key "waypoint position" refers to the relative position of the next waypoint with respect to the current one, encompassing both the distance and the angle between them. The key "connected nodes" means the objects involved in the current stage. Each node includes two attributes: the "node" name and its "position" which denotes the relative position with respect to the waypoint.

75 where **<Instruction>** will be replaced by the input instruction.