Autonomous Target Navigation with Unitree Go2's quadruped robot

Alexandre Huou (342227), David Farah (341017), Dylan Vairoli (326603), Hod Kimhi (385031)
COM-304 Final Project Report

***Abstract*—We addressed the challenge of autonomous target navigation for quadruped robots, focusing specifically on the Unitree Go2 quadruped robot. Utilizing reinforcement learning (RL) techniques, we developed a model capable of navigation and exploration in search of a pre-defined target. We trained our own model in a simulated environment mimicking real-world conditions. Finally, we deployed it on a Unitree Go2 to test our model in real-world scenarios. Our results show that the robot was able to recognise and follow the target in an unknown environment using both color and depth sensors.**

I. Introduction

Autonomous robots control is a rapidly evolving field. Thanks to the improvements in computing power, larger and more sophisticated models are continuously developed, leading in turn to more advanced robots and usages.

We aim to participate in this expansion by taking advantage of reinforcement learning and simulated environments. By training our own model, we have full control over the task definition and execution. We trained our agent to explore an unknown environment in search of a pink ball using both color and depth sensors. We then deployed the trained model on a real Unitree Go2 robot to evaluate it in real-world conditions.

II. Related Work

Recently developed technologies such as large-scale datasets (Matterport3D [1], Habitat-Matterport3D [2]) and visually and physically realistic simulators (Habitat [3], NVIDIA Isaac [4]) enable diverse Embodied AI tasks.

Among them, we can find *PointGoal navigation (PointNav)* where the target is a single position, and *ObjectGoal navigation (ObjectNav)* where the target is described as a semantic label for an object. We define our task as *TargetGoal navigation*, where the target is a pre-defined object randomly placed in an unknown environment.

We were heavily inspired by the already present features in Habitat for both PointNav and ObjectNav. The research realised in the context of the Habitat Challenge competitions [5] demonstrate concrete applications and follow the same approach as we did. However, our model specializes in finding one specific target, contrary to a variety of targets represented with a semantic label in ObjectNav.

III. Method

After having understood the overall robot system architecture, we split our work in two distinct phases: first manually controlling the robot, then train a model for target navigation in simulated environments using reinforcement learning.

A. Component overview

1) Hardware: The Unitree Go2 EDU is an advanced quadruped robot designed primarily for educational use. It integrates state-of-the-art technologies to provide robustness, versatility, and advanced sensory capabilities, making it ideal for navigating complex environments.

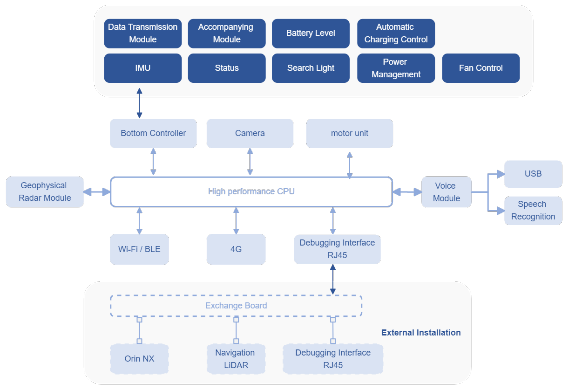
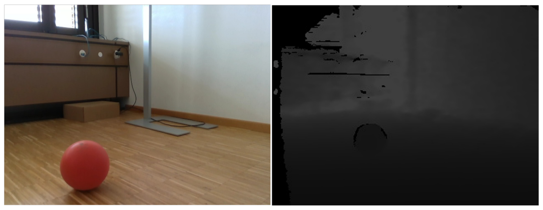


Fig. 1: Hardware Architecture of the Unitree Go2 [6]

This robot features many advanced sensors, providing our model with many useful data:

- **Front Camera**
  RGB camera with 1280x720 resolution and 120° field of view. It was used to test our initial policy, but our final policy only uses the Intel RealSense D435i camera to have a consistent resolution and viewpoint for both RGB and depth input.
- **Auxiliary Intel RealSense D435i**
  This combined RGB and depth camera with 640x480 resolution is an auxiliary sensor which is not directly included in the robot. It was connected to the Go2 on-board NVIDIA Jetson board through a USB-C port. The sensor's data is exposed through ROS topics published by the Quadruped Robotic's SDK[7]. Note the difference in FOV for RGB ($69° \times 42°$) and depth ($87° \times 58°$).
- **4D LiDAR L1**
  $360° \times 90°$ omnidirectional scanning, providing data in the form of a point cloud. As there is no Habitat support for LiDAR sensors, this sensor was not used in this project
- **Foot Force Sensors**
  Real-time foot pressure feedback. This sensor was not used in this project.



(a) RGB (b) Depth (rescaled)

Fig. 2: RGB and depth data from the D435i camera

2) Network interface: The only way to access the Go2 internally is through the onboard Jetson, as the onboard Raspberry Pi password is unknown and the MCU is set to Basic Mode, which prevents any file editing. The simplest way to SSH in the Jetson is via LAN as the Jetson has a fixed IP. To remove the dependence on a physical wire connection we used a Wi-Fi USB adapter, which required installing drivers onto the Jetson for RTL8821CU chip used in our adapter. This allowed for wireless SSH connections, provided that the adapter and the computer initiating the SSH connection were on the same network, as we decided not to enable port forwarding.

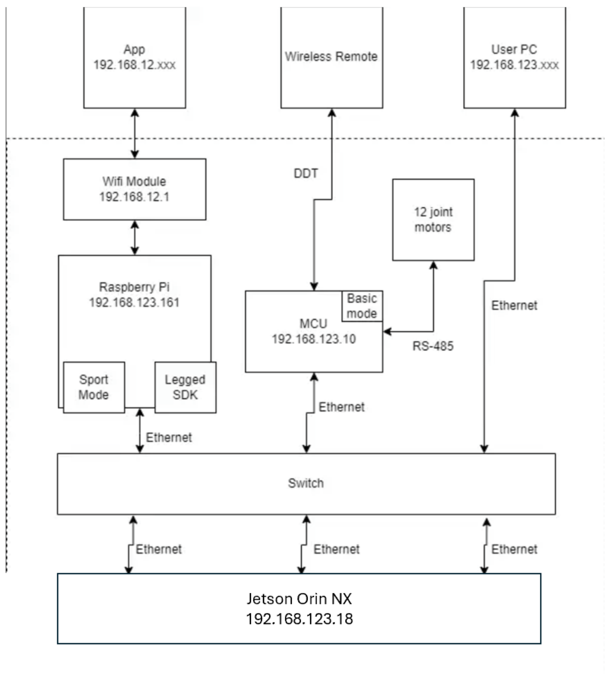


Fig. 3: Internal Network Layout of the Go2 [8]

B. Manual control

As a first step, we accessed the robot internal ROS topics via LAN to understand the internal working and the interactions between the components. Using the official *Unitree SDK v2* [9] we were able to send velocity-based control commands to the Go2. We then created our custom controller which wrapped the velocity commands, allowing us to provide instructions for a position-based control such as *move forward*, *turn left* or *turn right*, matching our action space.

Once we had a good grasp of the overall system, we needed to control it over WiFi. We initially sent commands to the robot over from the computer over WiFi through an exposed HTTP URL using the WebRTC protocol and the community-made Unitree Go2 ROS2 SDK [10]. This allowed us to run our ROS workspace and navigation model on our local machine, with better performance, and only send the output actions to the robot. However, this setup did not allow for access to the auxiliary D435i depth camera, as we were limited to one-way communication.

The second alternative was to deploy our code directly onto the robot and run it on its auxiliary computer. This gave us direct access to all the necessary sensor inputs and ROS topics using the private SDK from *Quadruped Robotics* [7]. Although we could access a greater number of topics and directly tie in to the D435i auxiliary sensor input compared to the initial setup, we faced performance issues as the Go2's computer was not as powerful. To combat this performance deficiency, we launched only the minimal ROS nodes necessary for operation, which solved the issue.

C. Training

For training our model in simulation, we also had to choose between two viable simulation frameworks. The first one, Habitat [3] developed by *Meta*, was recommended by the course staff at the project outset and had already been used in competitions pertaining to ObjectNav and PointNav. Unfortunately, this simulator has no support to train a model using a LiDAR sensor for data input, from which the Unitree Go2 benefits. Moreover, we found rather poor documentation and community support online. These reasons moved us to consider the use of Isaac Sim [4] instead, developed by *NVIDIA*. Compared to Habitat, Isaac Sim is better documented, has a more intuitive interface, and provides support for LiDAR sensor input. However, we had to start from scratch since the course staff did not have any starting point material for use with Isaac Sim, as they did with Habitat. Furthermore, Isaac Sim requires an NVIDIA RTX GPU, which only one member of the team had, limiting overall training time and learning bandwidth.

We decided that two members of the team will each look into one of the two simulators, to then compare which one produced the better model and was more interesting to use. We finally opted for Habitat, since we did not have time to train a complete model on Isaac. We still believe that Isaac has a strong potential and found it easier to use, and would recommend it going forward.

To get used to Habitat, we first trained a simple model in a single empty environment without any obstacles. As no randomization was been used in any way we expected the model to severely overfit to its environment. Due to this, we then went under a more complete training process. We created 9 different training environments with various floors and walls which were then populated by one of 50 random layouts of obstacles, composed of chairs, shelves, and cardboard boxes.

With this added randomization, we aimed to improve the generalisation of our model to many rooms and obstacles.

The observation space we used is composed of an RGB camera and a depth sensor, both with configurable resolutions and fields of view. We chose to stay in a discrete action space (*stop*, *move forward*, *turn left*, *turn right*) to keep our model simple and save computing power, considering the limited hardware and performance capabilities available on the Unitree Go2 robot. We produced two policies with these observation and action spaces, trained in very different conditions.

Finally, we tested our models in the real-world by deploying them on the Go2 in rooms similar to the simulation environments and in rooms that looked very different to evaluate its robustness to various spaces.

IV. Experiments

A. Simulation

The main parameters that we can update to alter the model performance are the following:

- **Sensors resolution**
  The RGB and depth image resolution directly affects the training result. Using a too high resolution is unnecessary for our visually simple task (recognizing a bright pink ball) and increases the training durations and the risks of overfitting to the environment.
- **Simulated environment quality**
  The materials, textures, and obstacles meshes of our simulated environment directly influence the resulting model's capabilities, as it is based on RGB data. We note, however, that this aspect is less important with lower resolution, due to blurring effects removing minor differences in material and lighting.
- **Randomisation**
  Having multiple randomly textured and populated environments is crucial to avoid overfitting and improve the navigation of our agent in unknown environments.

We used the PPO classes provided by Habitat Baselines (`PPOTrainer`, `PPOAgent`), which implement the *PPO RL algorithms* [11], along with our custom task based on the `PointNavResNetPolicy` policy with the point goal defined as our target (a pink ball).

Our first training pipeline used a single empty environment with both an RGB and a depth image of resolution 144×144. Deploying the resulting model on the robot did not yield any meaningful result as the model constantly instructed the *turn left* action. We did not evaluate this model in simulation as it did not feature any obstacles nor randomisation of any kind, so it would not be able to navigate in unknown environments.

Our final training pipeline featured 9 environments with random textures and obstacle layouts. We reduced the sensor resolution to 64×48 to match the aspect ratio of the D435i camera and to further reduce possible overfitting.

We evaluated this model's performance in environments using the same assets as the training ones, but with unknown and random layouts. It achieved 82% success rate, which we consider good enough.

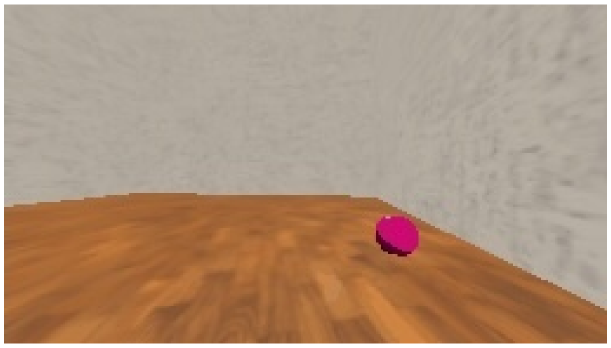


Fig. 4: Simulation footage of an empty environment

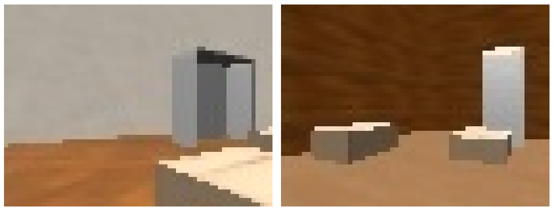


Fig. 5: Simulation footage of random environments

B. Sim-to-Real

Once the model has been trained and tested in simulation, it must be loaded onto the Unitree Go2 and tested in the real world. The model was run locally on the Go2's hardware in a ROS environment, allowing for direct in-band management and communication to gather RGB and depth sensor input and send output control signals reflecting the defined action space. Real-world evaluation of the model was done through repeated testing in an environment with random variation of the target's position, and recording success and failures. Unfortunately, the real-world performance was significantly worse than simulation performance. Our agent was not able to navigate successfully in a room full of obstacles (cardboard boxes). It was however capable of following the ball in an empty room, even though it had troubles stopping in time.

Many causes can be found to explain this difference between real-world and simulation performance:

- **Unrealistic assets**
  The textures and models we used do not represent well enough the real world. We only used three kind of obstacles and the textures were very simplistic. Realistic aestheticism was not exactly reached.
- **Not enough variety**
  We only used 9 different combinations of walls and floor textures and 3 different kind of obstacles. This does not represent the whole variety of indoor rooms that we can encounter in the real world. For example, we only used wooden floors for our training, which makes our model poorly generalize to grey floors.
- **Naive randomisation**

The obstacles layout was been done purely at random. We did not account for the nature of the obstacle. For example a shelf should not be placed randomly in the middle of a room but rather against a wall. Using a better layout algorithm could lead to a better representation of real-world environments.

- **Too high sensors resolution**
  We could have tried using lower resolution RGB and depth images to prevent overfitting. Since the ball stands out clearly in the room, a resolution lower than 64×48 could have been used.

V. Conclusion and Limitations

This project successfully demonstrated the application of reinforcement learning techniques in navigating a quadruped robot, the Unitree Go2, through unknown environments. Our model, trained in simulated environments, showed promising results in recognizing and following a target. However, transitioning from simulation to real-world applications introduced several challenges. The real-world performance was notably lower than in simulation. Future work should ensure accuracy of simulation environments and prevent overfitting through increased variety in environments, and increase environment quality through a greater diversity of assets and textures along with use a better randomisation algorithm.

Our work remains limited due to the overfitting of our model and the under-utilization of advanced sensors like the 4D LiDAR due to compatibility issues with our chosen simulation framework. Future work could explore the integration of additional sensory inputs into the training process, the use of more robust and generalizable training process, and improvements in hardware-software integration to enhance real-world performance. Additionally, further exploration into adaptive learning techniques could mitigate the transition gap between simulated and real-world environments. Using Isaac Sim [4] would seem wise due to LiDAR sensor support, despite the drawback of needing an NVIDIA RTX GPU to run it.

VI. Individual contributions

- Alexandre worked on the robot's hardware, its connection to a computer via wi-fi (for the first SDK before the one with WebRTC from the shop), built Isaac Scenes and worked on the TA's Habitat implementation.
- David worked on the habitat framework to the point where we could train the first model, learned general API usage and configuration setup. General support with networking setup and solving hardware issues. Started developing a task in Isaac sim towards the last 3 weeks of the project with native ROS2 support considerations and photo-realistic assets.
- Dylan developed the ROS architecture and implemented the manual control of the robot using three different SDKs for different use cases. He also designed and trained both Habitat models and added support for autonomous navigation nodes on the robot.
- Hod created Singularity environments for use on the SCITAS clusters, set up the networking (Wi-Fi, SSH via LAN, WAN) and the various ROS environments on the Go2's onboard computer to gather sensor input and control the robot, and implemented the ability to load and call a trained model with inputs to get an action output.

References

[1] A. Chang, A. Dai, T. Funkhouser, M. Halber, M. Nießner, M. Savva, S. Song, A. Zeng, and Y. Zhang, "Matterport3d: Learning from rgb-d data in indoor environments," 2017.
[2] S. K. Ramakrishnan, A. Gokaslan, E. Wijmans, O. Maksymets, A. Clegg, J. Turner, E. Undersander, W. Galuba, A. Westbury, A. X. Chang, M. Savva, Y. Zhao, and D. Batra, "Habitat-matterport 3d dataset (hm3d): 1000 large-scale 3d environments for embodied ai," 2021.
[3] M. Savva, A. Kadian, O. Maksymets, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, V. Koltun, J. Malik, D. Parikh, and D. Batra, "Habitat: A platform for embodied ai research," 2019.
[4] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State, "Isaac gym: High performance gpu-based physics simulation for robot learning," 2021.
[5] J. Ye, D. Batra, A. Das, and E. Wijmans, "Auxiliary tasks and exploration enable objectnav," 2021.
[6] U. Robotics, "Go2 sdk development guide," 2024. [Online]. Available: https://support.unitree.com/home/en/developer
[7] Q. Robotics, "Quadruped ros2 repository for unitree go2," https://github.com/MYBOTSHOP/qre_go2/tree/foxy-nvidia, 2024.
[8] DroneBlocks, "Go1 architecture diagram," 2022. [Online]. Available: https://www.youtube.com/watch?v=Wnd5IUNbXnI&ab_channel=DroneBlocks
[9] U. Robotics, "Unitree robot sdk version 2," https://github.com/unitreerobotics/unitree_sdk2, 2024.
[10] F. Tamás, "Unitree go2 ros2 sdk," https://github.com/abizovnuralem/go2_ros2_sdk/tree/master, 2024.
[11] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.