Introduction to ROS Group Project: Autonomous Quadruped

Shaolong Tang,Xinyuan Zhu,Run Yuan,Xiaoyue Hu,Yang Tang
Team 9

July 28, 2024

1 Task Division

	Perception	Mapping	Path Planning	Gait Controller	Presentation & Report
Xiaoyue Hu					
Shaolong Tang					
Yang Tang					
Run Yuan					
Xinyu Zhu					

Figure 1: Task Division

2 ROS graph

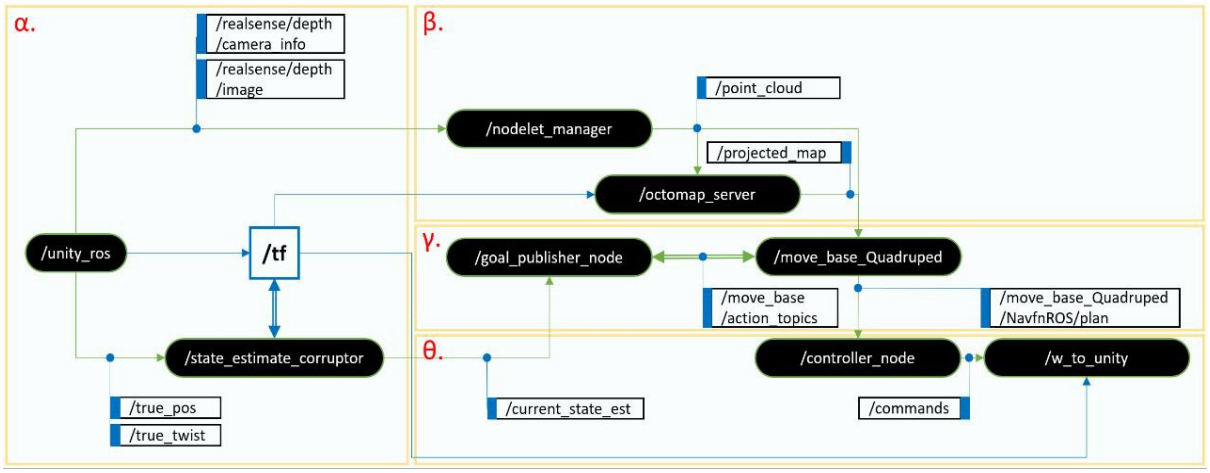

Figure 2: ROS graph

3 Description of ROS node and package

ROS Package	Functionality
depth_image_proc	depth image to point cloud
filtered_cloud	filter out the ground points noise
octomap	point cloud to occupancy grid
move_base_Quadruped_plan	Generate path planning
controller_pkg	Used to control the movement of the robot dog
goal_publisher_node	publish goal position to move_base to calculate path
path_height_checker	Used check the height of the occupied grids
controller_key_control	Use to control the use keyboard to control

4 Implementation Details

4.1 Perception pipeline

First, we used Rviz to confirm the rostopics for the RGB and depth images and to verify their positions. Then, we converted the depth images into a point cloud. Next, the point cloud was converted into an occupancy grid. The perception pipeline and the visualization launch file are located at AutonomousQuadruped/src/perception/launch.

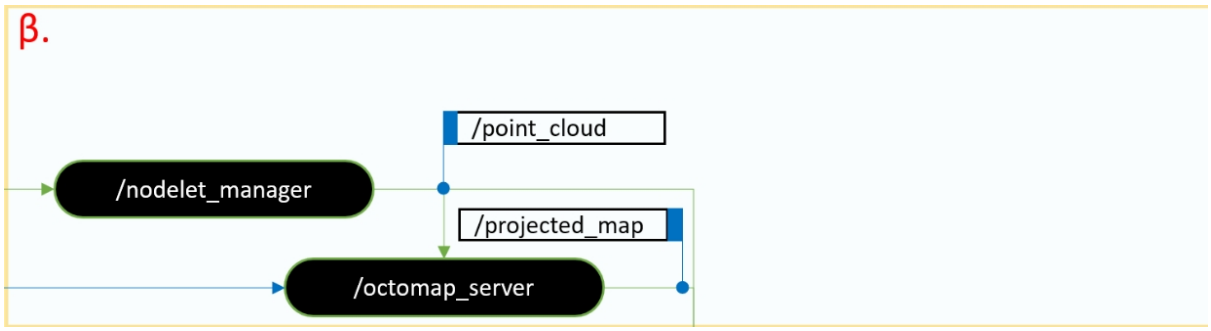

Figure 3: Perception

4.1.1 Depth image to Point cloud

We used the depth_image_proc recommended in the Tips to convert the depth images into a point cloud.[1]
• Configure the launch file as required.
• Remap the topic camera_info and image_rect to subscribe the depth information of the image.
• Remap topic points which will publish the point cloud information.

4.1.2 Filtered point cloud

We eliminate the influence of ground noise on the generated occupied grids by setting the ground filter in the octomap configuration file.

4.1.3 Occupancy grid

We used the Octomap and the node octomap_server recommended in the Tips to convert the point cloud into occupancy grids[2], resulting in a 2D projected occupancy grid (which can be used for navigation) and a 3D occupancy grid (used to determine the height of steps).
• Configure the launch file as required.
• Subscribe the point cloud from the topic /point_cloud
• Set the resolution of the voxel to 0.10
• Adjust the range of the z axis to filter out the steps voxel (keep the point cloud in z axis between 0.15 and 5)
• Publish topics including information about occupancy grid and the projected map in octomap_full and projected_map. The result can be visualized in rviz.

TODO: We plan to use the 2D projected map without steps(we filtered) as the map for move_base to use for planning, and then check the generated plan if the waypoints are on the steps or not. But we didn't manage to intergrate it into actual controller_node in the end. It will be illustrated in the following part.

4.2 Path Planning

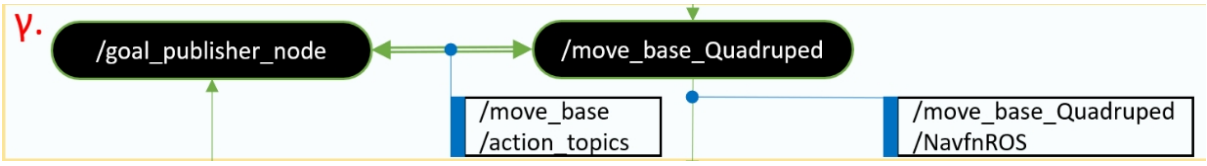

Figure 4: Path Planning

4.2.1 Planning node

We use move_base as external package for our planning module, it uses Navfn as default global path generator, which is A* based planning algorithm. And a /goal_publisher_node created by ourselves to send goal pose to move_base by using actionlib(move_base has an interface for actionlib). We didn't set single goal pose(we don't manage to create a global map), so we have to send multiple intermediate poses to continuously guide our robot dog. Once one intermediate position is reached, then publish the next position.(See /goal_publisher_node for more details)

There are several params in the move_base package that are sensitive for the performance:
1. footprint: [[-0.2, -0.2], [-0.2, 0.1], [0.2, 0.1], [0.2, -0.2]]
2. inflation_radius: 0.2
3. update_frequency: 40.0 and publish_frequency: 80.0 for the global_cost_map

4.3 Controller

4.3.1 Controller node

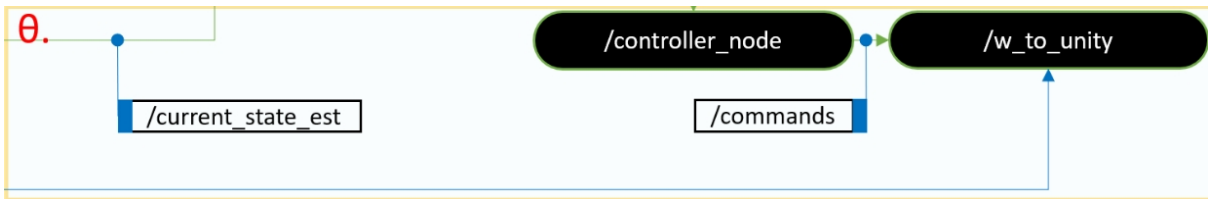

Figure 5: Controller

The controller node subscribes to two topics:
/current_state_est and /move_base_Quadruped/NavfnROS/plan, the node receives global paths from /move_base_Quadruped/NavfnROS/plan(we don't use local path). And we first filter most of the waypoints from the plan(we keep one for every 40 points, otherwise the waypoints might be too dense and hurt the actual movement of the robot), and we only keep first 10 waypoints from the first 400 points.

Since the plans are updated with time, this list of 10 waypoints are also continuously updated with time to keep aligned with the changing path plan.

So for each waypoint from the 10 waypoints, we send the first one into the controller_node. And actual control steps are illustrated in Figure 6. We set the initial pose from /current_state_est, and calculate the angle between initial orientation and desired orientation between robot and current waypoint. Then we rotate the robot according to the calculated angle until it's within certain angle tolerance. After that we move the robot straight forward toward the waypoint until it reached(There's also a position tolerance here)
After the waypoint is reached, then pop the next waypoint until all 10 waypoints are poped out.
So the there's actually three state in the controller_node, MOVING_FORWARD, ROTATION, IDLE(initial state and when the waypoints are reached)

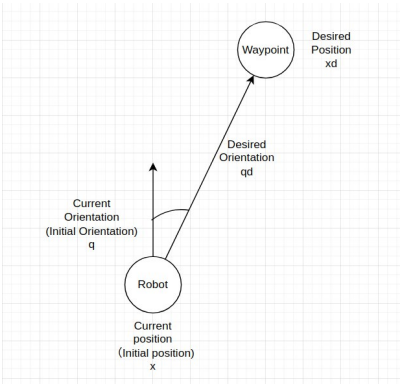

Figure 6: Robot Control

4.4 Quadruped Dynamics

The controller package provides 5 angular velocities, which simulate the gait pattern of a quadruped animal. The former 2 values represent phase difference legs. Then, the 3rd and the 4th angular velocities indicate amplitude values for legs, which displays as joint angle for lower legs. And the last value controls frequency for legs. According to the

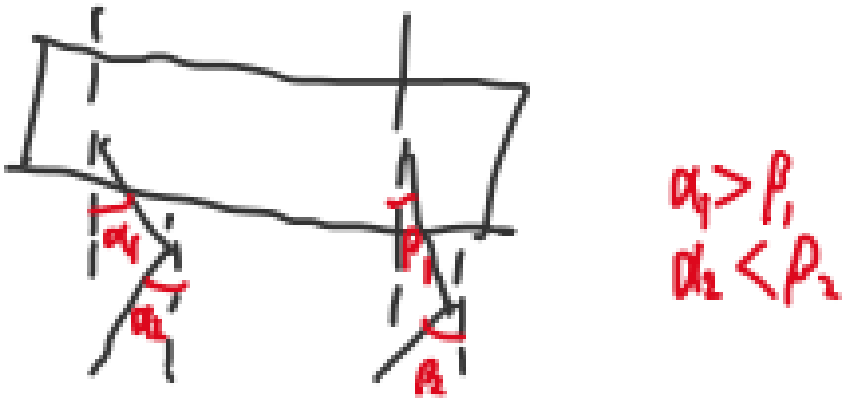

Figure 7: A quick sketch of standby mode

study by Yecheng Shao et al. [3], quadruped animals like dogs keep changing its phases between legs while moving at different gaits. Among these possible gaits, we picked trot as our forward moving gait, because this one walks steadily keeping the body stable.[4]

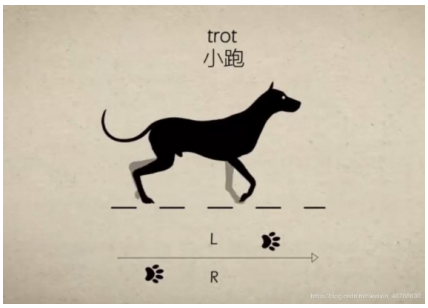

Figure 8:

TABLE I
CPG PARAMETERS FOR VARIOUS GAITS

Gait	T	β	$\bar{\varphi}$
Trot	$0.5s$	0.5	$[0, \pi, \pi, 0]$
Pacing	$0.5s$	0.5	$[0, \pi, 0, \pi]$
Bounding	$0.3s$	0.4	$[0, 0, \pi, \pi]$
Four-legged Walk	$0.6s$	0.75	$[0, 0.5\pi, \pi, 1.5\pi]$
Three-legged Walk	$0.45s$	2/3	$[0, 2\pi/3, 4\pi/3]$

Figure 9:

Rotating is relatively easier to be realized, which only requires the diagonal phase to be 45. However, getting our quadruped over slopes and step cases is unarguably the most challenging task in gait control. We devised 3 possible solutions for it:

Figure 10:

1: Long jump
Stretch its hindlegs and aim higher forward. Then make a jump to get its center of gravity up the slope. This option shall solve most slope climbing tasks easily. But we could not figure out a feasible set of control parameters to jump. So we postpone this plan.

Figure 11:

Figure 12:

2: Raising forelegs
This is the solution we ultimately applied. Firstly, it rotates its foreleg back, stretching its forelegs to reach the slope. After it lands its forelegs upon the slope, trot or do high amp walking to climb up the slope. This approach requires exact control over input duration, as raising forelegs for too long may consequent in tripping over. But we do manage to find a nice set of control parameters. And this one works most stable
3: Pace up while bending its forelegs This one works specifically for continuous slopes

Figure 13:

instead of staircases. Besides, it requires tilt information from the body to keep the body horizontally steadily. We save this one as a backup plan for going up slopes.

Still, we must emphasize that these 5 control values do not perfectly reflect the action control from a real quadruped animal. In particular, frequency is the only value we have to make it move, which is essentially only rotating the joints at given frequency. But we need a sudden change of rotation speed to make the robot jump. Therefore, jumping, canter and gallop could not be realized. In addition, the robot could only rotate its joints in a certain plane, which prevents it from shifting sideways. And the action input is severely influenced by hardware performance. If the UNITY.exe runs only in low frame rates like 23 or even less, we would have inconsistent result under the same control sequence due to lagging.

4.5 Implementation of message

To integrate path data with OctoMap 3D data, we need to use a custom ROS data type. This custom data type will combine path points and floating-point height values into a single entity, thereby generating a path_statues list containing these combined data. Specifically, each path_statues will include a path point and its corresponding height information.

This data structure effectively represents the position and height of path points in 3D space, providing more accurate information for robot navigation and path planning.

By using this approach, we can output a list of path points with height information, achieving comprehensive utilization of both path and environmental data.

Leveraging the mechanism of rostopics, we can ensure efficient transmission and sharing of these data between different nodes within the ROS network, thereby providing more reliable and precise navigation support for the robot.

5 Experiment and Result visualization

5.1 Experiment on filtering method of point cloud on ground noise

We try to use the SACSegmentation algorithm of Point Cloud Library to filter out the ground points in points cloud from detph image.

Although it shows a good effect on ground point noise in point cloud, in occupied grids, the effect is the same as the filter ground effect in octomap, so in occupied grids we still choose to use the built-in function of octomap to reduce the number of nodes.

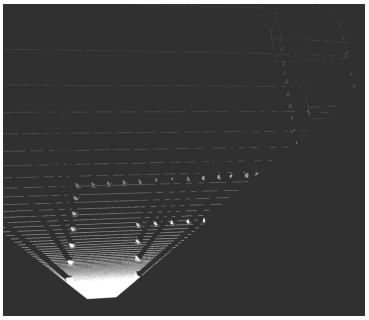

Figure 14: Points cloud without filter

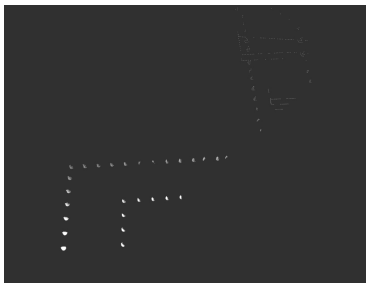

Figure 15: Points cloud with filter

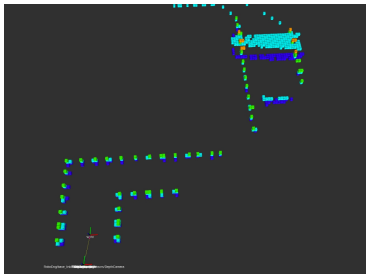

Figure 16: Occupied grids with ground filter

5.2 Comparative experiment of two different methods of slope inspection

In order to detect the slope on the path point and output the corresponding judgment value. We chose two different methods to detect the occupied grids of the slope.

The first method is to obtain whether there is a slope by subscribing to the message output by rostopic:project_map, that is, if the occupied area on the planned path exceeds a certain threshold, it is judged to have a slope.

The second method is to obtain messages by subscribing to rostopic:octomap_full, which not only determines whether there is a slope on the path, but also detects the height value of the occupied grids on the path point.

From the experimental results, the method of using project_map to obtain whether the shadow is occupied cannot correctly obtain the message. At the same time, the second method can effectively obtain the height value of the occupied grid and give a ros message.

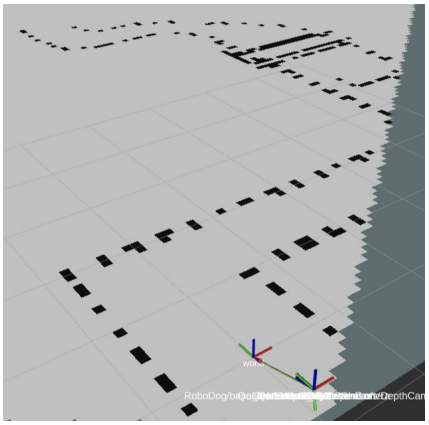

Figure 17: occupied grid

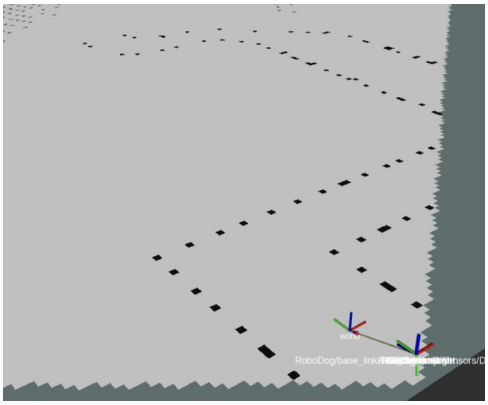

Figure 18: not occupied grid

6 Challenges and Remaining problems

6.1 Black-box Quadruped Gait Control

The complexity of ensuring coordinated leg movements, maintaining balance, and adapting to varying terrains without direct insight into the control mechanisms requires advanced problem-solving and iterative testing.

Bumps and Backs Up While Rotating:

While attempting to achieve smooth rotational movements, the quadruped robot frequently encounters problems with stability. During these maneuvers, the robot tends to bump into obstacles and occasionally backs up unintentionally. This behavior indicates a lack of precise control and coordination.

Unstable While Climbing:

Ensuring stability during climbing is another significant challenge for the quadruped robot. The robot often becomes unstable when navigating inclined surfaces or obstacles. This instability is may due to the complex interplay of forces and the need for precise weight distribution across all four legs.

6.2 Gloabl map unknow

When the global map is unknown, the robot must explore the local environment and set local goals. However, the local maps generated while the robot is in motion tend to be unstable. This instability complicates the navigation and decision-making processes, as the robot relies on these maps to understand its surroundings and plan its path effectively.

6.3 More flexible planning algorithm

The A* based planning provided by /move_base offers limited flexibility due to its few tunable parameters.

This restricts the ability to fine-tune the navigation and optimization processes, necessitating the development or integration of more adaptable planning algorithms to achieve better performance and customization in various environments.

7 Conclussion

Although we have conducted the above experiments, to ensure the stable operation of the quadruped robot, we chose very conservative parameters and methods for setting it up. Finally, we completed the navigation action up to the point before reaching the slope. The robot can detect the slope height and has the capability to climb over it. However, due to unknown power settings of the robot, it cannot stably achieve the climbing function. Therefore, we did not integrate the navigation, height detection, and climbing actions.

References

[1] depth information to occupancy grid http://wiki.ros.org/depth_ image_proc

[2] point cloud to occupancy grid http://wiki.ros.org/octomap/

[3] Learning Free Gait Transition for Quadruped Robots via Phase-Guided Controller, Yecheng Shao, Yongbin Jin, Xianwei Liu, Weiyan He, Hongtao Wang, Wei Yang, 2022

[4] https://www.youtube.com/watch?v=grGYAnFae7c