Motion Planning for A Multi-Robot System

Hoang-Dung Bui, Erdem Murat, Amirreza Payandeh

I. Introduction

Multi-robot motion planning is an active research field in robotics. There are a lot of research which handle with moving obstacles, dynamic constraints, path optimizations. There is a significant progress in this field, however, most research still make simplification for the robots system such as: the robot can change their velocities instantly, the effect of internal-momentum is minimized, the information of environment is complete, and robots' dynamic is reliable. Those simplification limits the deployment of the research in simulation and real application.

In this project, we will deploy a state-of-the-art motion planner *Continuous Conflict-Based Search* [1] for multi-agent in gazebo environment. In the environment, we consider several factors such as: the error in robot's motion models, the sensor's imperfection, robot's acceleration and deceleration, and robot's inertial momentum. The goal is to build a motion planner's pipeline for multi-robot navigating successfully in the environment without collisions. We will start with two robots, and then add one more robot to check the robustness of the motion planner pipeline.

II. Related Work

For this work, we found 5 related papers that take an approach to multi-robot navigation that is similar in some ways to ours. For example, a paper [2] used conflict-based search (CBS) and enhanced it for multi-agent path finding. They did this by first merging each node of the tree, and restarting the search after each merge. Then, they prioritized conflicts and made more informed choices on splitting conflicts. One paper[3] expands CBS by providing a framework operates on a search forest rather than a search tree, and creates the forest on demand, avoiding a factorial explosion of all possible tasks assignments. Another paper [4] expands on CBS with A*. One paper [5] focuses on reducing dynamic obstacle computation by building on the observation that while the number of safe timestamps may be unbounded, the number of safe time intervals may be finite and small. Another paper [6] reviews the systems from multi-agent path finding and surveys different categories and state-of-the-art approaches to improve computational optimality.

III. Project Framework

Deploying a research into a simulation application is a sophisticated work and time-consuming to integrate all the elements. The main components of the projects as shown in Fig. 1.

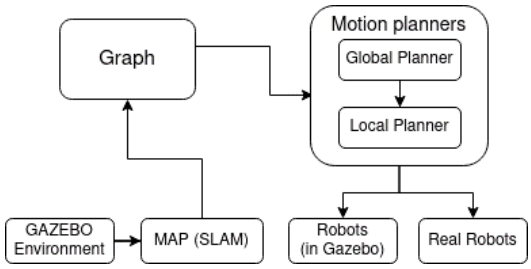

Fig. 1: Motion Planner's Structure

As being putted into a Gazebo environment, the robots with their sensors will perceive the environment by localization and mapping (SLAM). As a result, we have an occupancy-grid map of the environment. We will generate a graph $G = (V, E)$ for the motion planners. The graph reduces significantly the state's space of the motion planners comparing to the occupancy-grid map. To handle the robot's dynamics and sensor's errors, we setup a motion planner's framework which consists of two lever planners. The first one is called *Global Planner* which determines paths for the robots as inputting the graph G, and robots start and goal positions. This *Global Planner* is shared among all robots. As completing the task, the *Global Planner* will send paths to local planners which directly control the robot's models or real robots. The local planners divide the paths into multiple segments, and control the robot follow the currents ones. The passed segments will be ignored. Each robot has its own local planner. The *local planner*'s output are the velocities of the robots (angular and linear ones), which are sent to the on-board local controllers on the robots.

Based on the project framework, we divided the tasks for members as following:

- *Erdem*: Working Gazebo environment, occupancy-grid map, and graph's generation, and take major charge of writing this report.
- *Amirreza*: Working on occupancy-grid map, graph's generation, and write the description about his part in the report.
- *Dzung*: Working on Gazebo environment setup, the motion planner's framework (global planner and local planner), controlling the robots, and integrating all the components into a system. He also writes about what he has implemented.

IV. Environment Setup

In this project, we used Gazebo simulator to simulate a environment and robots. We create a open-space with wall-

closed boundary as shown in Fig. 2. Inside the space, there are 9 columns being setup in a square-shape as obstacles. Laser sensors were equipped on the robots and can detect the obstacles, the walls, and measure the distances between the robots and the objects.

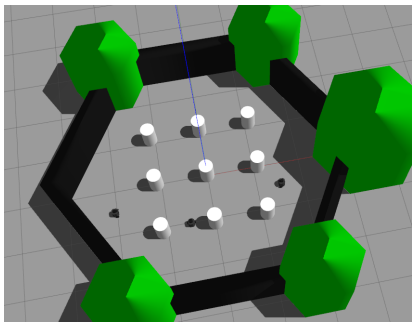

Fig. 2: Gazebo Environment for robots

The used robot's models are *turtlebot3* which is a mobile robot with differential drive. The robot can move forward, backward, and rotating by control the velocities' difference among their wheels. The robots are equipped with laser sensors which scan the environment frequently.

The number of robots in the environments can be changed as we set up the robot's description file. The robot's controllers are exactly the same except their namespaces. The initial robots poses were changed manually each time we made difference configuration tests. The most challenge's configuration for multi-robot motion planner is the robots are opposite and exchange their position. The environment starts with two robots, then extend the work for three robots.

The communication between gazebo with the motion planners was done by *ros messages*. The environment received the velocities messages from the motion planners via the topics *robot_name/cmd_vel*. The environment sent back the laser-sensor data and their odometry information to the motion planners by the topics */robot_name/scan /robot_name/odom*. The laser and odometry data were processed by the motion planner and can be displayed by *Rviz* as shown in Fig. 3.

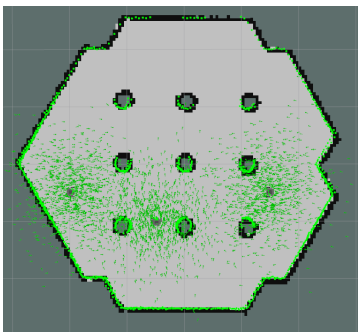

Fig. 3: Outlook of environment from robots' laser sensors

As mentioned earlier, we also consider the uncertainty of the robot location and the motion model's, we use adaptive Monte Carlo localization (*acml*) approach to estimate the position the robots. The green arrow in the *Rviz* displaying the particles in *acml*. We set the number of particles 100.

To run the simulation, we call the *launch* file in the *cs685* package (submitted) by the command:

- *roslaunch cs685 cs685_robots_gazebo.launch*

V. Occupancy-Grid Map and Graph Extraction

In this section, we discuss how to generate a occupancy-grid map from Gazebo environment. This *map* is the global frame which all others will rely on.

A. Occupancy-Grid Map

To create the map, we will add a robot into the Gazebo environment and move it around. As the robot moves, its laser sensor scans the environment and get the distance from the sensor (and robot) to the objects in the environment. Then, both the laser scan and odometry data were sent back to the *SLAM* node which then process and generate the map.

To perform the tasks, we open 4 terminals and exports the *environment* variable $TURTLEBOT3\_MODEL$ to *burger* (we use the burger model of robot turtlebot3). In each terminal, typing the following commands.

- In all terminals: *export TURTLEBOT3_MODEL=burger*
- *terminal 1*: *roslaunch turtlebot3_gazebo turtlebot3_world.launch*
- *terminal 2*: *roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch*
- *terminal 3*: *roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=gmapping*
- *terminal 4*: *rosrun map_server map_saver -f /map*

The command in terminal 1 will trigger the Gazebo environment with one robot. The command in terminal 2 allowed us to move the robot around the environment using a keyboard. The terminal 3's command starts the *slam* package which build the occupancy-grid map by the laser and odometry data. After the map is done, the command in terminal 4 will save it into a *pgm* file which is shown in Fig. 4. All the commands should be implemented following this order. Especially, the last command is only triggered as we already discover all the maps.

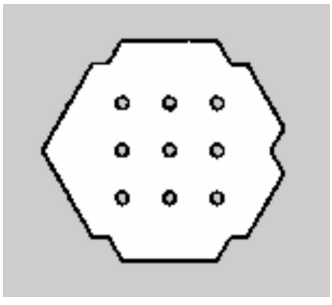

Fig. 4: Occupancy-Grid Map from the SLAM

As completing the map, we will generate a graph $G = (V, E)$ for the motion planners.

B. Graph Extraction

Using the occupancy-grid map as the input to the motion planner, the state-space will be significantly large. Searching becomes so expensive for the motion planner, especially, if adding more robot into the system (the computation time increases exponentially). To improve the efficiency of the searching, we convert the occupancy-grid map into a graph and use it as the input to the motion planner. To do this, we wrote Python code that used PRM (probabilistic roadmap) method to generate a graph which represents the environment structure.

Within a given area, PRM generates a limited number of random points. The PRM algorithm clusters nodes into connected components after they are generated. The nodes are connected if the connection's line do not intersect with any obstacles and the line's length is smaller than a threshold d_{max} (d_{max} is set to five in this project). To increase the clearance between the sample points and obstacles, we flatten the obstacles and wall by six pixels (the circle around the obstacles and the blue polygon offset from the boundary). The number of sample is set to four thousands. The result's graph from the algorithm is shown in Fig. 5.

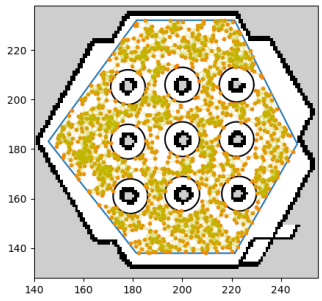

Fig. 5: Graph from the Occupancy-Grid Map

This graph will be saved in a file *graph.xml*, which is one input to the CCBS - global planner.

VI. Motion Planner Framework

In this section, we will discuss how to build a motion planner framework which is able to determine collision-free paths and control the robot strictly follow the paths under several constraints. As mentioned in section III, the motion planners consists two levels: global one and local one.

In the *global planner*, we use the planner *Continuous Conflict-Based Search* - a state-of-the-art motin planner [1]. The planner receives a graph, robot's starts and goals, then output the collision-free paths with timestamp for all robots. In the paths, beside the set of waypoints which the robots must follow, there is also a timestamp, which indicates how long the agent has to perform the move. If the agent can not achieve the points at the timestamp, the collision still can happen. The global planner is the block *CCBS - Global MP* in Fig. 6.

The *global planner* exchanges data with the *Main node* by ros service. As the *Main node* receives the paths from the

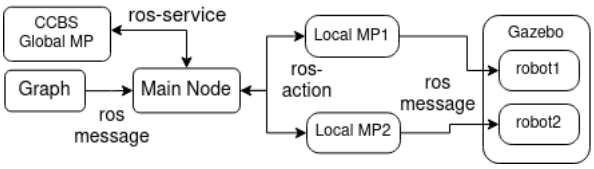

Fig. 6: Motion Planner's Pipeline

global planner, it sends them out to the *Local MP* nodes. As we have more robots in the simulation, we need to add more *local planner* in this structure. The structure in Fig. 6 works for two robots.

The *local planner* is based on Dynamic Window Approach (DWA) [7] to avoid collision. This planner will divide the received path into multiple-segments, and only consider the current segment which the robot is currently in. The previous segments will be ignored. As the robot completed 1 segment, the local planner will move to the next one. In each segment, the planner will calculate the velocities (angular and linear) for the robot and send them to the wheels by a ros message with topic */robot_name/cmd_vel*. After moving each step, the robot will re-estimate its location by *acml* package. As the estimated pose closes to the goals under a threshold, the robot will be considered being at its goal.

The communication among the nodes are described as following:

- Between *Graph* and *Main Node*: As *Graph* node has the graph, it will send it by a ros message to *Main Node*.
- Between *Main Node* and *CCBS - Global MP*: They exchange data by *ros - service*. As *Main Node* have a graph, starts and goals for robots, it will send a *request* to the node *CCBS - Global MP*. As completing the *paths*, the node will response to *Main Node* by a set of paths.
- The exchanged data between *Main Node* and *CCBS-Global Planner* node consists of multiple paths, which a new structure in *nav_msgs* package, thus we define a new service message as shown in Fig. 7. The request

```
# request
geometry_msgs/PoseStamped[] starts
geometry_msgs/PoseStamped[] goals
---
# response
nav_msgs/Path[] paths
```

Fig. 7: Service message

  contains of two vectors of *geometry_msgs/PoseStamped* which consists of the starts and goals. The reponse part contains a vector of *nav_msgs/path* - consists the determined paths for the robots.
- Between *Main Node* and the *Local Planners*: They exchange data by *ros action*. As *Main Node* receives a set of paths, it will separate the set into the single path and send it to the corresponding robots. Then *Main Node* monitors the moving process of the robots and

receives the feedback from the *Local MP* nodes during the process. As the robots reach their goals, all *Local MP* will send a result back *Main Node*.

- To use *ros action* to communicate between *Main Node* and *Local Planner* nodes, we setup a new action message which is new for our own purpose. The structure of the action message as shown in Fig. 8. The goal part

```
#goal definition
nav_msgs/Path path
---
#result definition
---
#feedback
geometry_msgs/PoseStamped current_pose
```

Fig. 8: Action message

is the path which the robot need to follow. As robot reach goal, we do not need to send any data back, just notify that the robot has reached goal. The action server feedback to the action client by the current robot pose.

- Between *Local Planner* nodes and their robot's models: the *Local Planner* nodes send the velocities to the robot by *ros message topic*.

VII. Implementation

There are prerequisite packages, which are needed to be installed to run our simulation packages:

1) Install ROS *noetic* (with Rviz and Gazebo)
2) Install the ROS navigation stack package: *https://github.com/ros-planning/navigation*
3) Install Shapely *pip install shapely*
4) Install CCBS *https://github.com/PathPlanning/Continuous-CBS*

A. Running our project

Due to the complex of CCBS package, we don't have enough time to integrate all the mentioned components into a seamless ROS system. The *simplex* package in CCBS used a definition circle to define some subclass within it. That is an error in ROS, which maintain a strict definition hierarchy. Therefore, the system works as following:

- Generate the occupancy-grid map and save it into a *map.pgm* file (following section V-A)
- Run the python script (python3 graph_xml.py) to generate a graph (from the map.pgm file), and save the graph into a *graph.xml* file. The graph should contain at least 4000 nodes.
- Define the number agents and its goals and starts in a file: *graph_task.xml*
- Run CCBS (after compiling it successfully) by the command: *./CCBS address_to_graph/graph.xml address_to_graph_task/graph_task.xml*. The output is a *graph_task_log.xml* file, which consist the paths for robots. Copy that file into the *src* directory in the submitted cs685 package.
- Run the cs685 package which is a ROS package and submitted with this report. To run this package, we need four terminals:
  - *terminal 1*: run the command *roslaunch cs685 cs685_robots_gazebo.launch*
  - *terminal 2*: run the command *roslaunch cs685 cs685_3turtlebot_navigation.launch*
  - *terminal 3*: run the command *rosrun cs685 reading_paths*
  - *terminal 4*: run the command *roslaunch cs685 move_control_client*

  Terminal 1 runs the Gazebo simulation, terminal 2 runs the *local planners*, terminal 3 reads the file *graph_task_log.xml*, extract the paths to be ready. And terminal 4 calls *Main Node*, which get the *paths* from the *reading_paths* nodes, separate them and send the single paths to each *local planner*. The node *move_control_client* will terminate as it receives the results from all *local planners*.

VIII. Results

Although the ROS Integration work is not done, the system still perform in some ways, and the results are quite positive. We tested with two agents and then extended to three agents.

A. 2 Agents

We have test in four robot configurations. In configuration A (scene A) we set up a simple scenario as the robot paths does not intersect. In scene B, the starts and goals for each agents are far, and the paths intersect at 1 point.

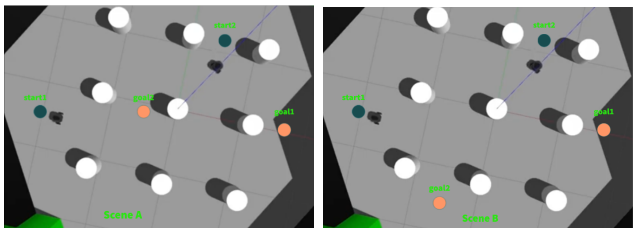

Fig. 9: Configuration A and B

In scene C (Fig. 10), we set the goal of agent 1 is close to the agent 2, so the motion planners must find paths which move agent 2 out of is initial position as soon as possible. The most challenge scene is scene D, where the robots at the opposite positions, and their goals are at the opposite position. It requires the motion planner being robust to ask at least one robot make a turn soon to let other robot reaching the goal.

In all the scenes, the motion planner works efficient, and determines the paths for the robots to go safely. The local planners also perform well, and regulate the robots' velocities to reach the goals without collision and proper time stamps. The implementation of two robots configuration can be seen full in this `https://youtu.be/3cM25l3SEEg`.

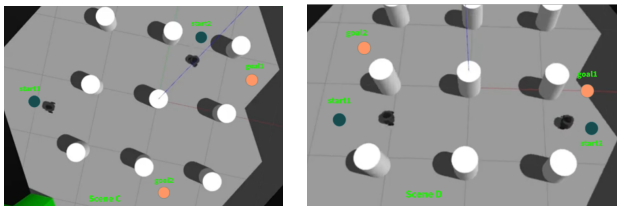

Fig. 10: Configuration C and D

B. 3 Agents

In this part, we add one more robot into the environment. This will make the simulation more complicated in motion planning and controlling. The robot configuration is shown in Fig. 11.

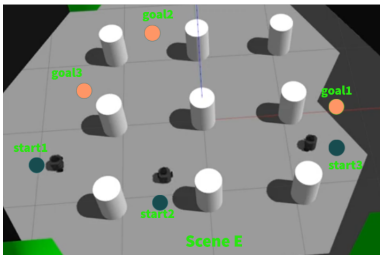

Fig. 11: Motion Planners with 3 robots

All robot paths intersect each other at some points, and their starts are at some side of the environments, and the targets are one the other side. The goal 1 is close to start 3, and vice versa, and that makes challenge for the motion planners to complete the tasks. And the motion planners proved their robustness by planning and controlling the robot reaching their goals safely. The video-clip can be seen in this link `https://youtu.be/q7LwcfWmWbQ`.

IX. CONCLUSIONS

In this project, we have built a motion planner framework which works for a multi-robot system to find collision-free paths for multi-robot. The motion planner framework contains a *global planner* and multiple *local planners* can handle the robot dynamics, tolerate the sensor's errors and the uncertainty of the robot motion's models. As shown in the results, the motion planner framework is able to work well with two to three robots with multiple scenarios. There is still some parts which are not completed in this project such as integrating all the components into a ROS seamless system or improve the clearance from the robots to the obstacles.

In a review, we can say that we have completed all the initial goals which are defined at the beginning of the projects: (1) build a sensor-based motion planner for two robots in a Gazebo simulation, (2) the motion planner can determine efficient the paths for the dual robots in simple scenarios. More from that, we can extend to three robots which works in complex environment with multiple obstacles. We also tested the motion planner in multiple scenario, and it still provided a positive result.

X. FUTURE WORK

From this project, there is plenty of space to improve in the future. We should spend more time on engineering work to build a seamless ROS multi-robot system. We should improve the *local planner* to increase the clearance to the obstacles and other robots. Improving the velocities control of the *local planner* is also a interesting aspect. One more interesting task is applying this motion planner framework to the real robots. Moreover, to get a good publication from this, we need to increase significantly the number of robots and add more constraints such as communication ones into the system.

REFERENCES

[1] Anton Andreychuk, Konstantin Yakovlev, Pavel Surynek, Dor Atzmon, and Roni Stern. Multi-agent pathfinding with continuous time. *Artificial Intelligence*, 305:103662, 2022.

[2] Eli Boyarski, Ariel Felner, Roni Stern, Guni Sharon, Oded Betzalel, David Tolpin, and S. E. Shimony. Icbs: The improved conflict-based search algorithm for multi-agent pathfinding. In *Symposium on Combinatorial Search*, 2015.

[3] Wolfgang Hönig, Scott Kiesel, Andrew Tinka, Joseph W. Durham, and Nora Ayanian. Conflict-based search with optimal task assignment. AAMAS '18, page 757–765, Richland, SC, 2018. International Foundation for Autonomous Agents and Multiagent Systems.

[4] Yang Li, Jun Wang, and Hualiang Zhang. Research and optimization of conflict search algorithm for multi-agent path planning based on incremental heuristic. *Journal of physics. Conference series*, 2024(1):12048–, 2021.

[5] M Phillips and M Likhachev. Sipp: Safe interval path planning for dynamic environments. In *2011 IEEE International Conference on Robotics and Automation*, pages 5628–5635. IEEE, 2011.

[6] Hang Ma. Graph-based multi-robot path finding and planning. *Current Robotics Reports*, 3(3):77–84, jun 2022.

[7] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.