Homework 1 intelligent robotics

Mattia Fanan, Luca Davì, Hassan Habib

December 22, 2021

1 Problem analysis

The problem that we had to deal with was simple, create something that enable to move the robot from the current location to another and, when the latter has been reached, retrieve the position of non-static obstacles. The target location has to be given as command line argument. The code had to implement an action client/server infrastructure to solve the problem.

2 Ideal structure

The ideal solution should should follow the structure reported in the image 1. The idea is that to impose a strong modularity in such a way that we can reuse and update the code with more freedom. The client and server parts are imposed by the assignment, the robot part is what we have to interface with and the last three was left to our interpretation and we felt that each of the components should be an entity by its own. E.g. from the same set of obstacles points we can infer multiple sets of features.

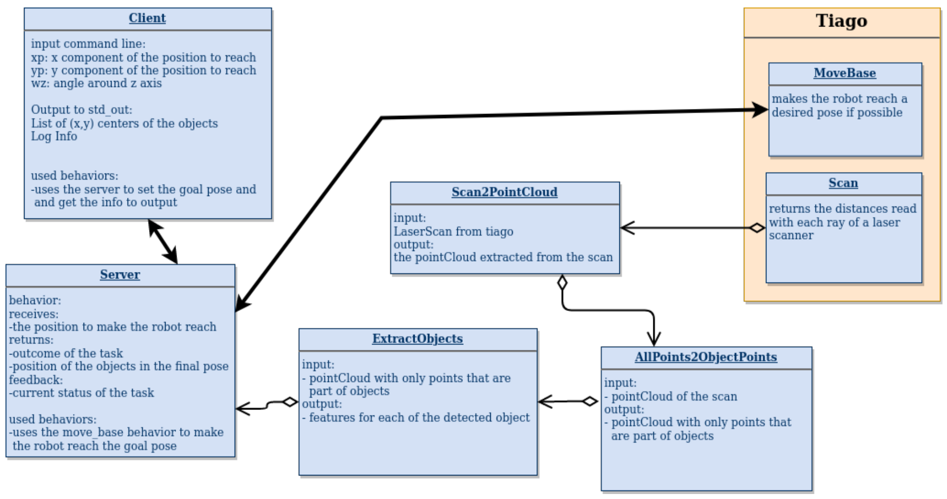
Client
input command line:
xp: x component of the position to reach
yp: y component of the position to reach
wz: angle around z axis
Output to std_out:
List of (x,y) centers of the objects
Log Info
used behaviors:
-uses the server to set the goal pose and and get the info to output
Tiago
MoveBase
makes the robot reach a desired pose if possible
Scan
returns the distances read with each ray of a laser scanner
Scan2PointCloud
input:
LaserScan from tiago
output:
the pointCloud extracted from the scan
Server
behavior:
receives:
-the position to make the robot reach
returns:
-outcome of the task
-position of the objects in the final pose
feedback:
-current status of the task
used behaviors:
-uses the move_base behavior to make the robot reach the goal pose
ExtractObjects
input:
- pointCloud with only points that are part of objects
output:
- features for each of the detected object
AllPoints2ObjectPoints
input:
- pointCloud of the scan
output:
- pointCloud with only points that are part of objects

Figure 1: Ideal structure of the system.

3 Practical structure

Ideally the plan was to realize the ideal structure that keep the points with respect the robot base link frame of coordinates and then to re-reference them with respect to the world reference frame when needed, however we would have needed a lot of additional messages to ensure the right path for the information so in practice in the Scan2PointCloud part we already output the points with respect to the WRF (map frame, the one at the starting position of the robot). Then also the extraction of the obstacles points and their features was merged to simplify a bit.

4 Implementation

4.1 Action-Client

This node implement a simple action-client that require 3 arguments that has to be given by command line. Those are the elements needed to describe the motion on a plane and they are: two coordinates that will describe the position xp, yp and the orientation wz. The action-client will request to move the robot in the given location by sending a request to the action-server (navigation_server) that we made. After that, it will wait a result from that before continuing. When a result arrives, if the goal state of the action-server was set as SUCCEEDED, the action-client create a pointer to the result of the action-server and will print all the estimated coordinates of the detected objects center.

4.2 Action-Server

This node implement a simple action-server that will work as interface with our client and the other nodes needed to solve the problem. Every time a request arrives, the callback uses a simple action-client to send a request to the move_base action-server with the data sent by our action-client. If the request end up by having a SUCCEEDED, our action-server will retrieve the processed data from the obstacles_centers node using, in a first time a shared pointers to those data published in the topic, and then "saving" those in local so to work with them without changing the data published. In the end we use those data to set the result of the action-server. Feedback are set and published with information on what is happening or happened.

4.3 ObstaclesPositionNode

This part in the ideal structure could have come out both as a service or as a node depending on the computational complexity of the two components. As we will see later for the obstacle points detection the complexity is linear, for the feature extraction is in the worst case quadratic but in the very majority of the cases is not and we also deal with data of small sizes (666 points) so it's ok to keep it as a node that continuously publish the features.

4.3.1 Obstacle points detection

To solve this task in the most general way possible we exploited the fact that given a closed surface considered the space respectively on the inside and on the outside must have respectively some convex, concave parts. E.g. a sphere from the inside is convex since from every two points all the points between them is inside the sphere and from the outside is concave since from any two points on the surface the points between them are also inside the sphere. To have such information we used the gradient to the surface in the Cartesian space. Between two objects there is a big jump on the magnitude of the gradient so we are able to separate the objects between the peeks of the gradient magnitude. Then we can watch the angle of the gradient that is more or less constant in flat surfaces and always decreases or increases along concave or convex surfaces (the map between the two depends on what is the given positive sense of the angle in our case the obstacles were always decreasing angles). Actually we checked the derivative of the angle gradient for simplicity. Then to discard the angles of the room that sometime get false positives we set a lower bound for the number of points in the object to be accepted. Notice how all this operations are linear in the size of the problem (find_peaks simply checks for points with both neighbors of lower value [2] so should be linear).

4.3.2 Feature extraction

To solve this task we use the approximated least squares circle from [3] that needs to solve a linear system of size square the number of points in the object, then being not common that an object is of size comparable with the whole scan this is most of the times linear and even if not with 666 points is not a big problem.

4.4 ScannerCloudPointNode

This node converts the data from the Laser scan from the topic /scan to the PointCloud2 points that are the published in the topic /intro_navigation/scan_cloud. This part was keep aside from ObstaclesPositionNode since both it was planned to do so and using the python libraries is not possible to perform the high precision projection of the points with $.transformLaserScanToPointCloud()$ instead of $.projectLaser()$ as written in [1].

5 Future improvements

- Due to some problems related to the inability of one group member to work well because the lack of proper hardware/software, and due to other deadlines, we were able only to complete the mandatory part plus the feedback.
- Improvement of diagnostic messages (to be more precise and situation based).
- Reach the originally planned modularity
- Make possible to work with all the points with respect to the robot frame and then re-reference them when needed
- Try a better find peaks in case of noisy data even if the problem hasn't manifested itself here.
- Try to remove the lower bound on the points to be accepted as an object since in the long distance the points on the objects become less and less.
- To integrate we use as integration step the scan angle increment so we need to pass it through message instead of setting it hard-coded as it is.

6 Work shares

- Luca Davì made the Client-Server infrastructure and the integration of the other part with the first.
- Mattia Fanan made the three obstacle detection related nodes.
- Hassan Habib had hardware and software problems.

References

[1] *ROS laser_geometry.* `https://wiki.ros.org/laser_geometry`. Accessed: 2021-12-22.

[2] *SciPy find_peaks.* `https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.find_peaks.html`. Accessed: 2021-12-22.

[3] *scipy-cookbook Least squares circle algebraic approximation.* `https://scipy-cookbook.readthedocs.io/items/Least_Squares_Circle.html`. Accessed: 2021-12-22.