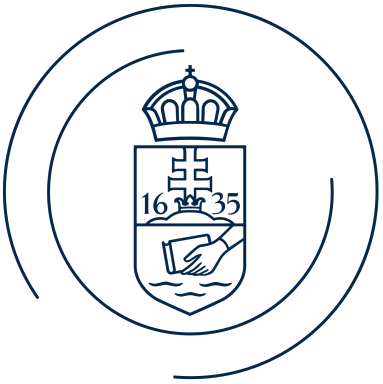
Report



ELTE
EÖTVÖS LORÁND
UNIVERSITY

Guide Robot

Supervised by:

Istenes Zoltan

Submitted by:

Jomana Ashraf

Rahaf Abu-Hara

Contents

List of Figures

1. Objective

The objective of this project is to make an interactive guide robot in the university which helps to guide people to their required place all over the university. It is an interactive robot which autonomously navigates to the required place by using path planning and localization algorithms to achieve the goal precisely.

1.1 Overview

The work is implemented in both simulation and real robot with the same localization approach but the planning algorithm is different in simulation than that is used in the real robot. The simulation is using the RRT with Dubins but the real robot uses the move_base package which is implemented from ROS. In the localization, the robot_localization package is used in order to fuse the data obtained from odometry, IMU, and GPS to give a new filtered odometry that can be used by the planning algorithm to give the current position of the robot.

2. Simulation

The robot used is the scout-mini which is equipped with a velodyne lidar that creates a 360-degree 3D map of the environment. Using the information from the lidar, the robot will use the **octomap** package to give the robot the free cells to navigate through it and also the cells which had the obstacles to prevent it. The planning algorithm used in the simulation is **RRT with Dubins** which is implemented by us and it give the path and the movement of the Robot is controlled using **tracking PID** package.

2.1 Methodology

As shown in fig.1, the odom and the lidar readings are published from Gazebo; where the lidar topic is "/velodyne_points" which is received by the octomap. The octomap then published the map to a topic called projected map to visualize the environment around the robot. Then our main node,**global_planner** node,is subscribing to this projected map to get the seen part by the lidar from the environment and determine which cells are free and which are obstacles. Furthermore, the position of the robot is being published by the odom topic and subscribed by the node also which will use this data. The global planner node uses the "RRT Dubins" node to compute the path required to reach the goal and then send it to the interpolator which convert it to a trajectory that can be subscribed by the controller which then sends the velocities to the motors of the robot to move to the goal in the specified path.

2.1.1 Octomap

The OctoMap library implements a 3D occupancy grid mapping approach, providing data structures and mapping algorithms in C++. The map implementation is based on an octree.

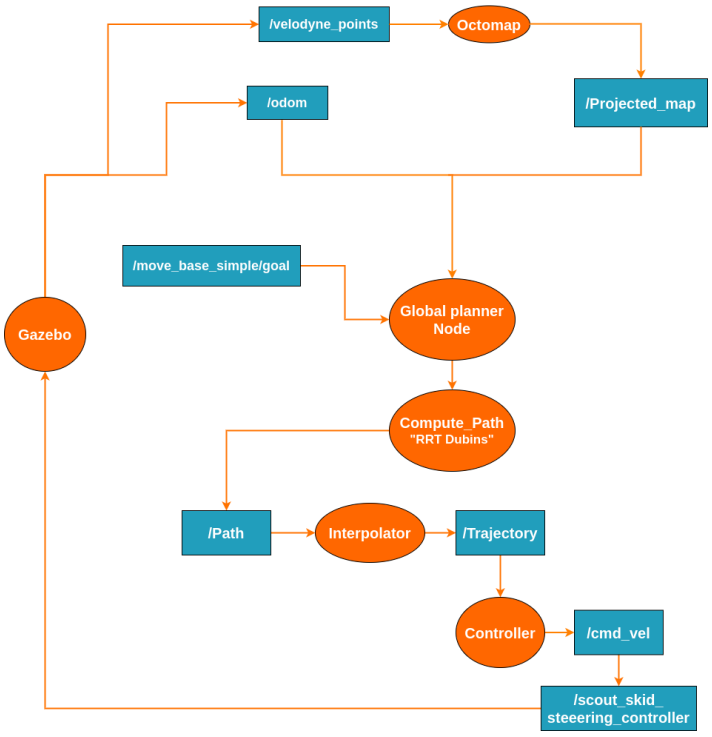


Figure 1: shows the main architecture of the projects with all the nodes and topics used

2.1.2 Global Planner Node

One of the fundamental challenges in mobile robotics is computing a sequence of valid configurations that moves the mobile robot from a start configuration to a goal configuration while avoiding collisions with known static objects using local information from its onboard sensors. The Rapidly-exploring Random Tree algorithm(RRT) algorithm is unusable for path planning and combine it with Dubins Curve algorithm.

2.1.2.1 Dubin's curves

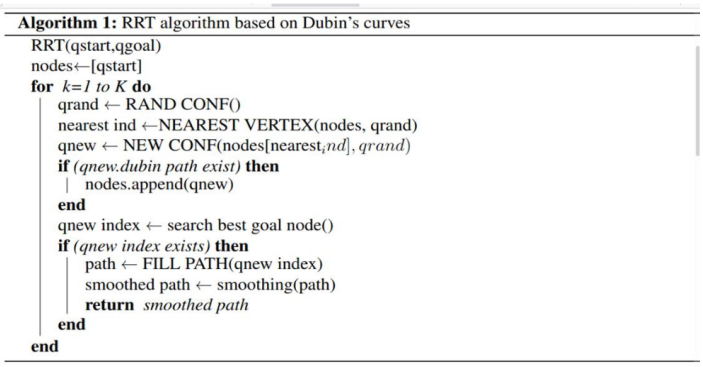
The optimal path between any two configurations in space can be expressed as a combination of no more than three motion primitives S, L, and R [1]. The S means that the robot moves straight ahead. R and L primitives stand for turning as sharply as possible to the right and left, respectively. Using these primitives, every possible path between two states in configuration space can be designed as a sequence of three symbols that corresponds to the order in which primitives are applied. These sequences are called Dubin's curves. An optimal path will always be the shortest curve of the six Dubin's curve types: RSR, RSL, LSR, LSL, RLR, and LRL.

2.1.2.2 Basic RRT

The RRT algorithm is commonly used for path planning problems in mobile robotics [2]. The algorithm grows a tree rooted at the starting configuration by using random samples from the search space. As each sample is drawn, a connection is attempted between it and the nearest state in the tree. If the connection is feasible a new state will be added to the tree. Then the generated states will be used to form a path.

2.1.2.3 RRT with Dubins

The main difference between basic RRT and RRT Dubins used in this project is in the way how the newly generated point is connected with the nearest vertex on the graph. In the basic RRT algorithm, two states in the tree are connected by a straight line. Since that is not possible in the case of a mobile robot with differential constraints, the algorithm is modified to connect states using Dubin's curves. So, if a feasible Dubin's curve exists then it will be added as edges to the graph. Algorithm execution and its main steps are illustrated by the pseudocode given in fig.2, which is explained in more detail in the next subsection.



Algorithm 1: RRT algorithm based on Dubin's curves

```
RRT(qstart,qgoal)
nodes←[qstart]
for k=1 to K do
    qrand ← RAND CONF()
    nearest ind ←NEAREST VERTEX(nodes, qrand)
    qnew ← NEW CONF(nodes[nearest,nd], qrand)
    if (qnew.dubin path exist) then
        nodes.append(qnew)
    end
    qnew index ← search best goal node()
    if (qnew index exists) then
        path ← FILL PATH(qnew index)
        smoothed path ← smoothing(path)
        return smoothed path
    end
end
```

Figure 2: shows the Pseudocode of the RRT algorithm based on Dubin's curves.

Functions Explanation

1. **RAND CONF Function**
   This function generates a random configuration 'qrand' inside the free space of the grid map. First, it creates a random value between 0 to 1, if this value is smaller than 'P', which is a small probability of sampling the 'qgoal' as the 'qrand', the random tree grows toward the target point, the higher this probability, the more greedily the tree grows and biases towards the goal. On the contrary, if it is bigger, the random tree is facing one Growing in a random direction.

2. **NEAREST VERTEX Function**
   After generating 'qrand', 'NEAREST_VERTEX' function takes it as an input, then it iterates over the 'nodes' list to compute the Euclidean distance 'dlist' between each vertex and 'qrand', after that it checks and returns the index of the minimum distance 'minind'.

3. **NEW CONF Function**
   This function connects 'qnear' and 'qrand' with a Dubin's curve using the 'shortest_path' function from the dubins library, which computes the shortest dubins between two configurations with a constrained turning radius. Using the 'sample_many' function, the generated path is sampled at a finite step size which is assigned to 0.1, the smaller the step size the longer the returned path. Then 'qnew' is found after moving an incremental distance 'q' from the parent 'qnear'

in the direction of 'qrand' following the generated curve. Longer distance "Δ q" will help to reach the goal faster especially in the open space like in this project, therefore it is assigned to 40. However, the length of the path connecting the 'qrand' node and the 'qnear' node is limited to ensure that 'qnew' stays in the space configuration. If the generated edge is inside the grid map, a new object of the Node class is created to save the node. its parent and the edge.

4. **Check collision Function**
   This function checks each point of the edge between 'qnear' and 'qnew' if it is valid or not. Moreover, it checks if the cells at distance 'dis' from the (x, y) position are also free or not. The function returns True if all checked cells are free and False otherwise.

5. **Search best goal node Function** This function checks if the distance and the absolute difference of the angles between 'qnew' and the 'qgoal' are less than a threshold, if True it will return the 'qnew'. The distance threshold 'goal_xy_th' and the angle threshold 'goal_yaw_th' are set to 0.5 and 0.4 rad respectively.

6. **FILL PATH Function**
   After the tree is reached the target, it's the time to compute the path, this will be done by moving from 'goal' (last node in the nodes_list) to the 'qstart'; because there are multiple paths from 'qstart' to 'goal', but only one path from 'goal' to 'qstart'. This path is constructed by finding the parent of the goal node and adding the edge between the goal node and its parent to the path, then adding the edge between the goal's parent and its parent, repeating this until reaching the 'qstart', finally it returns the constructed path after adding 'qstart' to it.

7. **Smoothing Function**
   The smoothing algorithm is used for obtaining a shorter and less noisy path. Using a greedy approach, first, a path between 'goal' and 'qstart' is constructed using a Dubin's path, if it's not valid another path will be constructed from a closer position until it connects. Once the 'goal' is connected, this process will be repeated with its directly connected position.

2.1.3 Controller

The tracking PID controller package from ROS is used to send the desired velocities to the robot. This package is composed of two nodes which are the interpolator and the controller. As mentioned before, the global planner sends the path over the /nav_msgs/path topic which the interpolator subscribed to and then sends it to the controller. This package offers a tunable PID control loop to accurately follow a trajectory as shown in fig.3 by using the command "rosrun rqt_reconfigure rqt_reconfigure". The tuning is done by trial and error to make the robot eventually move along and to turn with accurate rotations with the specified path. To tune those parameters and

know the effect of each parameter, we use Table. (1) to know the influence of the parameters on the robot behavior.

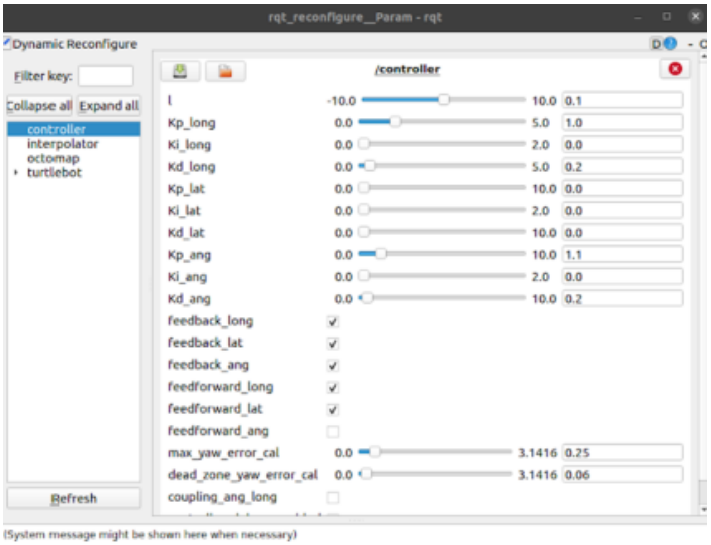


Figure 3: shows the rqt for tuning the controller parameters

3. Real Robot

3.1 Localization Algorithm

The robot_localization is a collection of state estimation nodes, which is an implementation of a nonlinear state estimator for robots. It contains two state estimation nodes; extended kalman filter and unscented kalman filter. In our case, the ekf is the one that is used in order to estimate the robot position. This package has no limit for the number of sensors used to fuse but the only restriction is about the message type that it supported which are:

- nav_msgs/Odometry
- sensor_msgs/Imu
- geometry_msgs/PoseWithCovarianceStamped
- geometry_msgs/TwistWithCovarianceStamped

In order to use the package, there are five steps required to be configured to run the localization perfectly.

3.1.1 Parameters for the Localization Package

3.1.1.1 Specify the coordinate frames

The frames that is used by the robot should be defined where the 4 important frames are map_frame, odom_frame, world_frame, and base_link_frame.

3.1.1.2 Input data

For each sensor, the message type is required to be defined. The index for each parameter name is 0-based (e.g., odom0, odom1, etc.) and must be defined sequentially and then the values for each parameter are the topic name for that sensor.

3.1.1.3 Data from each sensor

For each of the sensor messages defined above, the variables should be defined to true or false in order to get the true variables fused into the final state estimate.The order of the boolean values are X,Y,Z,roll,pitch,yaw,$X^{\cdot}$,$Y^{\cdot}$,$Z^{\cdot}$,$roll^{\cdot}$,$pitch^{\cdot}$,$yaw^{\cdot}$,$X^{\cdot\cdot}$,$Y^{\cdot\cdot}$,$Z^{\cdot\cdot}$. As shown in fig.9, the $X^{\cdot}$,$Y^{\cdot}$, and $yaw^{\cdot}$ are used for fusing only.

```
odom0_config: [false, false, false,
               false, false, false,
               true, true, false,
               false, false, true,
               false, false, false]
```

Figure 4: shows the data used for fusing from odom0

3.1.1.4 Initial State Covariance

The estimate covariance,commonly denoted P, is a 15x15 matrix which defines the error in the current state estimate which allows to set the initial value for the matrix, which will affect how quickly the filter converges.

3.1.1.5 Process noise Covariance

The process noise covariance, commonly denoted Q, is also a 15x15 matrix where the diagonals are the variances for the state vector. It models uncertainty in the prediction stage of the filtering algorithms and it should be well tuned as it improve the results produced by the filter.

3.1.2 How to use the GPS data?

As known that the GPS message data is "sensor_msgs/NavSatFix" which is not supported by the robot_localization package. So, the "**navsat_transform_node**" is used to convert this type of message to a "nav_msgs/Odometry" which can be used as an odometry but given from the GPS. It produces an odometry message in coordinates that are consistent with your robot's world frame. This value can be directly fused into the state estimate.

Then after obtaining the "odometry/gps" message; it can be used in the parameter file which is explained in section 3.1.1 So, the localization_package will fuse the odometry, IMU, and GPS data and will produce a new odometry message which is called "odometry/filtered" that give the current estimate of the robot location without any noisy data taken into account.

4. Results

4.1 Simulation

4.1.1 Simulation Environment

In the below figure we can see the created environment in Gazebo, it consists of moving people in which we can consider them as dynamic obstacles and buildings as static obstacles.

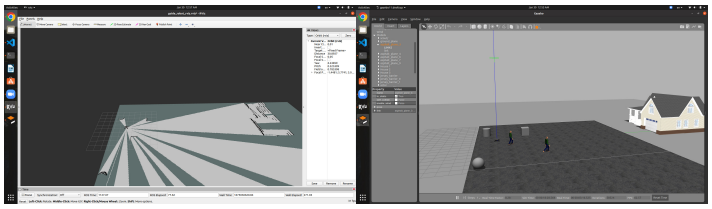


Figure 5: shows the created environment in Gazebo.

4.1.2 Planning using RRT-dubins

As shown in Fig.6 the algorithm has planned successfully a smooth valid path (the green path) towards the goal.

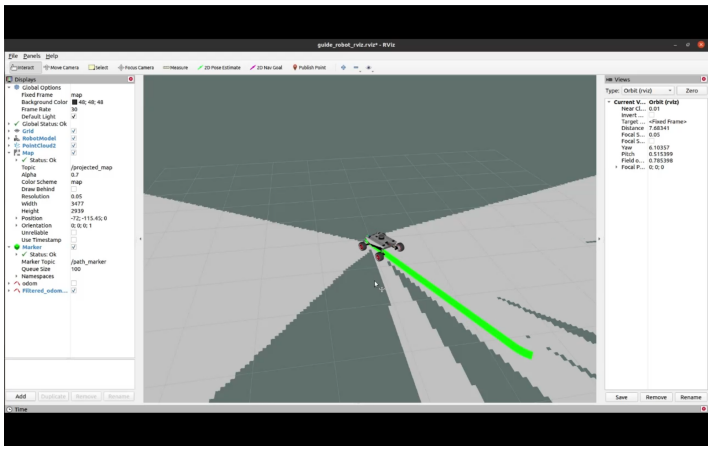


Figure 6: shows the planned path using RRT-Dubins algorithm.

4.1.3 Localization using IMU, GPS, and Odometry Data

One of the main objectives of this project is to localize the scout mini precisely by fusing the data coming from the IMU, GPS, and the Odometry. As there is no noise on the odom data in the simulation, we have created a noisy odometry and sent it to the EKF node to test it. It can be deduced from the following figure that the node filtered the noisy data (red arrows) perfectly after the data fusion.

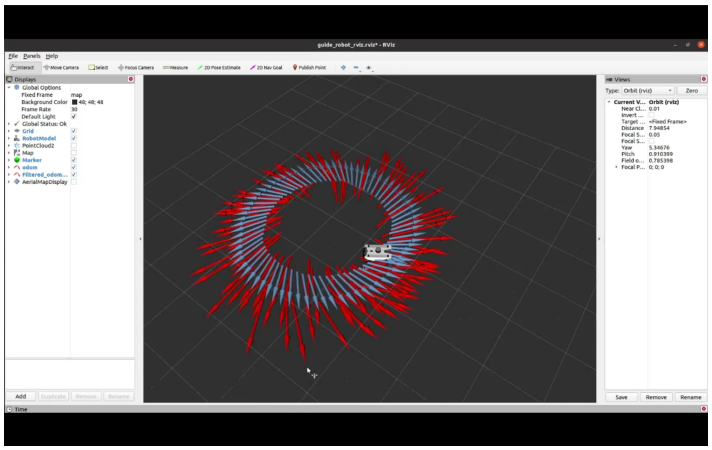


Figure 7: shows the filtered data (blue arrows) after sending the noisy data (red arrows) to the EKF node.

4.2 Real Robot

4.2.1 Testing the fusion of IMU and Odometry data

We have transferred the work step by step to the real robot. The first step was to test fusing only the IMU and the Odometry data and it works well as shown in Fig.8, the red arrows are the noisy odom which is received from the real robot and the golden arrows are the filtered odom.

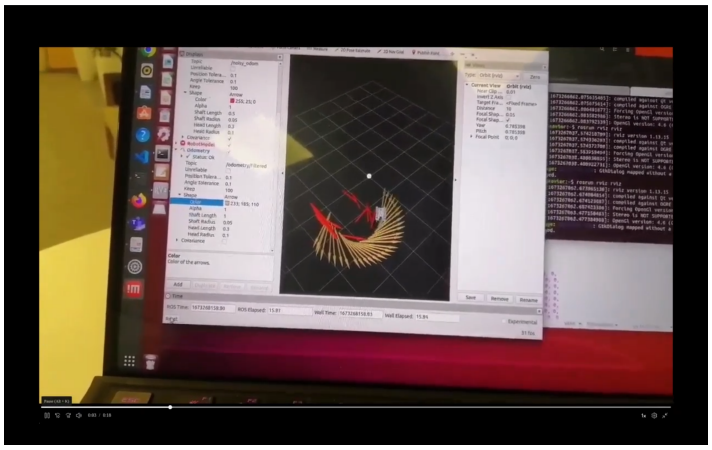


Figure 8: shows the filtered data (golden arrows) after sending the recieved odom data from the real robot (red arrows) to the EKF node.

4.2.2 Testing the fusion of IMU, GPS, and Odometry data

In this step we have tested the fusion of the three sensors together a after specifying a goal to the robot around the CLC building. It is worthy to be noted that we have sent the filtered odometry to the move-base package for the navigation. As demonstrated in Fig.9 the noisy odometry (orange line) has been filtered nicely (blue line) after fusing the three sensors' data by the EKF-node.

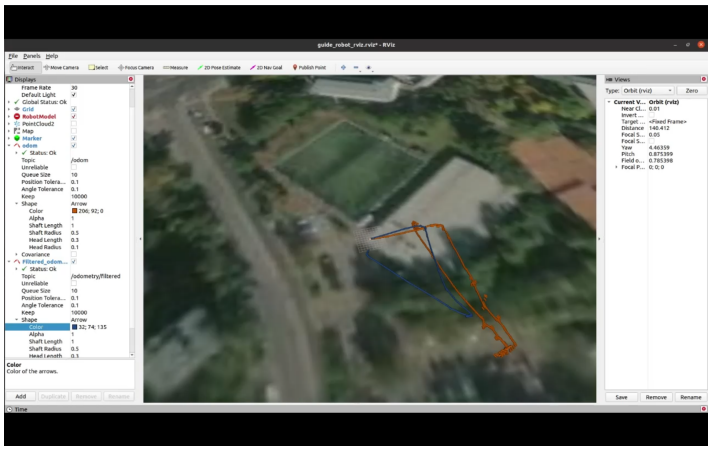


Figure 9: shows the filtered data (blue line) after fusing the GPS, IMU, and Odometry data.

5. Future Work

We are planning to implement the Dynamic window approach to use it as a local planner to avoid the dynamic obstacles efficiently. Then to merge it with the RRT-Dubins algorithm for the autonomous navigation part. Furthermore we will implement a face recognition algorithm to recognize and keep tracking the person that the scout robot is guiding, this will be helpful in the event of loosing the person in the crowd while guiding him/her.