Documentation

Robotics Challenge

submitted by

Steffen Schubert
ID Number L1TY86LWJ9
Curso: Robotica y Vision Artificial

Departamento del Deporte e Informática

submitted on April 4, 2025

Contents

1 Dynamic Robot Controller

For the dynamic robot controller we had a few different challenges:

- Navigating towards a goal
- Managing linear and angular velocities
- Avoiding obstacles

1.1 Navigating towards a goal

To drive the robot towards a goal the basic principle is turning the robot towards the target direction and then applying a linear velocity if the robot is facing towards the target direction. While there is always a command for an angular velocity, linear velocity is only applied in the given condition.

1.2 Managing linear and angular velocities

For the angular velocity two different velocities are considered; when the robot is facing away from the goal by a big angle, the max angular velocity is applied; when the robot is facing close to the target direction, a slow angular velocity is applied. To control this behavior we introduced a variable that is the *allowed angle*. The allowed angle is set to 10° and controls the window of angles of the robot, with respect to the target direction, in which linear velocity may be applied. When the angle of the robot is below 1.5 times the allowed angle, angular speed is reduced. This allows the robot to slowly keep making corrections and also slowing down the angular velocity even before the allowed angle is reached, so linear velocity is never applied while the fast angular velocity is active.

Idea: Angular = 0 for very small angles

For the linear velocity the following is considered: when the robot is facing away from the target direction, no linear velocity is applied; when it is facing closely towards the target direction, the max linear velocity is applied and multiplied by a speed factor between 0 and 1. The speed factor depends on the following:

- Distance to the nearest obstacle
- Whether or not there is a big difference between the current intended direction and the direction of the next subgoal.

If the distance to the nearest obstacle (min_dist) is below the radius of the sphere of influence variable (r_soi), the speed factor is calculated in the following way:

$$(r\_soi - min\_dist) * 0.5 + 0.5$$

This results in a value between 0.5 and 1, dampening the speed when the robot approaches an obstacle. If there is a big difference in angle between the current intended direction and the next subgoal (45°), the speed factor is set to 0.33. This leads to the robot slowing down in situations when he has to move into a direction that is way off the intended path.

1.3 Dynamic Collision Avoidance

Collision Detection

Collision detection is done using the *min_dist* and *r_soi* again. Once the *min_dist* is below the *r_soi*, collision detection is started. Collision detection is maintained until the *min_dist* is greater than $r\_soi + 0.3$. This is done in order to not keep starting and stopping the collision detection while the robot is on the edge of the sphere of influence.

Collision Avoidance

Idea: consider also 180 deg point

For the collision avoidance, a potential field approach was used. We hereby consider only the nearest point towards the robot. From the angle and the distance of the laser measurement, we calculate the 2d point in space of the obstacle and use the inverse as a repellant force. Using the formula from the lecture about potential field forces we calculate an initial potential field force:

$$m = \frac{r\_soi - min\_dist}{min\_dist}$$

We then combine this force with the direction of the current subgoal using weights.
Through trial and error we reached a final weight of 4.5 for the weight of the direction of the subgoal and 1 as a weight for the potential field forces. Since the resulting direction is normalized, we only consider the relation between the two weights. Choosing a low value for the weight of the subgoal direction leads to potential field forces overwhelming the target direction and the robot not driving closer towards the goal. Increasing the weight leads to the robot following the intended path more closely, but also increasing the risk of a collision, as the distance towards obstacles is generally a bit lower.

1.4 Problems Encountered

Overcorrecting: Angular speed

Before introducing the slowed down angular speed, we tried to just apply the positive or negative max angular velocity, this leads to the robot always overshooting the desired direction, often times correcting from one side to the other back and forth. Therefore, the reduced speed was introduced when the robot is facing somewhat towards the current target direction.

Overcorrecting: Linear speed

Especially in tight corridors, where the potential field forces are really strong, the robot would often times be forced to drive perpendicular to the wall in order to get away from it. For this purpose the angle between the direction of next subgoal and the current intended direction was considered, which can be seen by the speed factor being set to 0.33, when this angle is large. This leads to the robot slowing down in such situations and increasing the likelihood of getting

onto the path where it can drive through the corridor. Increasing the weight of the goal direction in the potential field forces also helped in these situations, but always had the risk of increasing the potential for a collision.

(Almost) Running into walls

When linear speed was set to the max linear speed regardless of distance to obstacles, sometimes the robot would get scarily close to walls, before the potential field forces overwhelmed the forces for the target direction. For these cases the dampening speed factor for when the robot is within the sphere of influence of an obstacle was introduced.

Moving on the Edge of the Sphere of Influence

When the robot was moving along the edge of the sphere of influence, sometimes it happened, that it would start the collision detection and therefore collision avoidance which made it turn around to move away from the obstacle. Then the collision detection would stop making it turn around again to move towards the target. This kept happening, with the majority of the time spent turning around. For this issue the buffer of the collision detection was introduced as described above.

Problems with Provided Material

While we tried to keep code changes within the marked *TODO* sections, sometimes the circumstances would force us to change code beyond that. In the case of the robot controller, the controller would just permanently stop the agent, whenever the goal was reached. While this is not a problem when there is only one goal, in the final challenge there are several goals, leading to the robot stopping after reaching the first goal. A slight modification was done to circumvent this problem.

2 Path Planner

For the global path planning we implemented the A^* algorithm. We introduced a new class to represent the nodes in our graph of possible cells. These objects then kept track of their g and h values resulting in an f value for the algorithm to use. The nodes were stored in a hash map in order to keep track of which cells already have an associated object created and which not. In this way, we only needed to create those nodes that are actually necessary for the algorithm, instead of creating the entire graph from the collision map.

2.1 Problems Encountered

Hardware of the Lab

Executing our code on the PCs in the lab yielded very different results. The smaller PCs struggled a lot with running gazebo, rvis and the algorithm at the same time. This lead to the first run of the algorithm often taking upwards of 20 seconds to execute. When running the programs on other machines in the lab (one of the larger ones), the algorithm executed within a very short timeframe of receiving the goal.

Problems with Provided Material

Again, while we tried to keep code changes within the marked *TODO* sections, sometimes the circumstances would force us to change code beyond that. In the case of the path planner, when creating the obstacle map it had a threshold that was a bit low, which resulted in multiple of the navigation goals being marked as invalid cells, which then stopped the path planning.

3 Simulation Results

When we ran the evaluation simulation on the better PCs in the lab we could see that the robot would reach all four goals with no penalties:

```
Goals reached: 1
Elapsed time: 198.106000
Traveled Distance: 19.214963
Min distance to obstacles: 0.173652
Avg distance to obstacles: 0.350000
Collision penalties: 0
Linear vel penalties: 0
Angular vel penalties: 0
```

Listing 3.1: metrics.txt

When running on the slower PCs, the robot would not reach all four goals which could be due to the long times that the A^* took to execute, but it also seemed as though the robot wasn't controlled as smoothly as on the better PCs.

3.1 Problems Encountered

Problems with Provided Material

Originally we would see a lot of penalties for the angular velocities, which was an error within the *evaluation.py* script, where the *max_lin_vel* ros parameter was also used to set the maximum angular velocity. When we set the correct parameter, no penalties were applied.