TEK4030, Trajectory tracking with mobile robots

Nadia Garson Wangberg, Ludvig Løken Sundøen, Jeya Lakshmi Subramanian

November 27, 2020

1 Abstract

In this assignment trajectory tracking was implemented in Python with ROS for a simulated mobile robot. The system consists of three distinct subsystems, path planning, timing law and trajectory tracking. The result can be seen in figure 6.

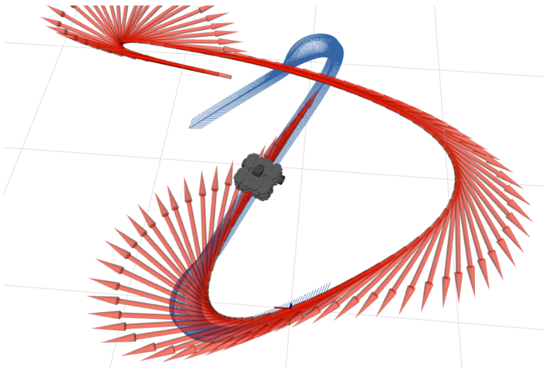

Figure 1: Trajectory of TurtleBot 3 Waffle Pi in Rviz. Blue is the planned trajectory, while red the path it took

2 System overview

The mobile robot was initially simulated using Gazebo, but the sim was later changed to *turtlebot3_fake* due to being more convenient. *turtlebot3_fake* provided an rviz config file that made it easy to visualize both the robot, its odometry and the path from our trajectory planner. The goal pose of the trajectory could also be set interactively in rviz with the "2D Nav Goal" tool, that let the user publish a message of type *geometry_msgs/PoseStamped* to the topic */move_base_simple/goal*. Figure 2 is a diagram from *rqt_graph* showing how the different nodes (circles) communicate with each other on different topics (squares).

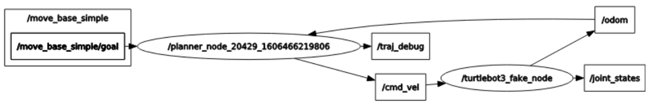

Figure 2: rqt_graph

Our node, the */planner_node*, subscribes to both the trajectory goal pose */move_base_simple/goal* and the mobile robots ground truth odometry */odom*. When the node receives a goal pose it plans a trajectory from its current pose to the goal pose and publishes it to */traj_debug*, which lets the user visualise the trajectory in rviz. Within the */planner_node* the timing law and trajectory tracking algorithm is implemented which uses the trajectory together with the robots pose to send messages on */cmd_vel*. These messages commands the simulated mobile robot to follow the trajectory and reach the end goal in a specified amount of time.

3 Path planning via Cartesian polynomials

Path planning via Cubic Cartesian polynomials was implemented according to the theory in chapter 11.5.3 in the book (See bibliography). The path's final (x_f, y_f, θ_f) and initial (x_i, y_i, θ_i) poses were given as input. In addition the s_{arr} output from the timing law and the parameters k_i and k_f was input. k_i and k_f are tuning parameters that affected the shape of the path.

The parameters $\alpha_x, \alpha_y, \beta_x$ and β_y were used in equation 1 to generate the path array of poses $(x(s), y(s), th(s))$. α and β are given by equation 2.

$$\begin{bmatrix} x(s) \\ y(s) \end{bmatrix} = \begin{bmatrix} s^3 - (s-1)^3 x_i + \alpha_x s^2(s-1) + \beta_x s(s-1)^2 \\ s^3 - (s-1)^3 y_i + \alpha_y s^2(s-1) + \beta_y s(s-1)^2 \end{bmatrix} \tag{1}$$

$$\begin{bmatrix} \alpha_x \\ \alpha_y \end{bmatrix} = \begin{bmatrix} k_f \cos\theta_f - 3x_f \\ k_f \sin\theta_f - 3y_f \end{bmatrix} \tag{2a}$$

$$\begin{bmatrix} \beta_x \\ \beta_y \end{bmatrix} = \begin{bmatrix} k_i \cos\theta_i - 3x_i \\ k_i \sin\theta_i - 3y_i \end{bmatrix} \tag{2b}$$

4 Timing law for trajectory planning

When a path is determined, we need to choose a timing law $s(t)$ such that the robot reaches the end point in the desired amount of time $T = t_f - t_i$ .In in this part we need to take into consideration the physical limitations of the turning rate and max velocity of the agent. We used the normalized time variable $\tau = t/T$ instead of using t directly. The desired velocity and turning rate were given according to theory from chapter 11.5.4 in the book and are given by equation 3.

$$v_d(t) = \tilde{v}(s) \frac{ds}{d\tau} \frac{1}{T} \tag{3a}$$

$$\omega_d(t) = \tilde{\omega}(s) \frac{ds}{d\tau} \frac{1}{T} \tag{3b}$$

From this timing law the output is $v_d(t)$ and $w_d(t)$. These can however not be sent directly to the robot, as feedback of the robot pose is necessary. We therefor need a Trajectory tracking algorithm which takes feedback into account to better deal with odometry and modelling imperfections.

5 Trajectory tracking

It is not sufficient to rely on that the vehicle will follow the path perfectly. A small error in the tracking will further propagate at each trajectory time step. To prevent this a trajectory tracking controller with feedback was implemented. The error was transformed from the world frame to the body frame and used in the error feedback according to equation 5.

Finding the characteristic polynomial of the system 5 and setting it equal to the mass-damper spring oscillator's $(s^2 + 2\zeta\omega_0 + \omega_0^2)$, gives us equation 4. ζ and ω_0 are tuning parameters. ζ is the relative damping ratio which can be tuned to a value above 0. If $\zeta < 1$ the system will be under-damped, oscillatory and less stable. $\zeta = 1$ gives a critically damped system, which is usually preferred. $\zeta > 1$ gives an over-damped system, which could be slow at converging. ω_0 is the undamped natural frequency which can also be tuned. We used $\omega_0 = 1$ and $\zeta = 1$ for our simulation.

Approximate linearization was done in order to get equation 6. This is the final linear velocity and rotational velocity which is directly sent to the simulated robot.

$$k_1 = k_3 = 2\zeta\omega_0 \tag{4a}$$

$$k_2 = \frac{\omega_0^2 - \omega_d(t)^2}{v_d(t)} \tag{4b}$$

$$u_1 = -k_1 e_1 \tag{5a}$$

$$u_2 = -k_2 e_2 - k_3 e_3 \tag{5b}$$

$$v(t) = v_d(t) \cos e_3 - u_1 \tag{6a}$$

$$\omega(t) = \omega_d(t) - u_2 \tag{6b}$$

6 Results

As can be seen in the figure the mobile robot managed to follow the trajectories quite well. Tuning k_i and k_f changed the aggressiveness of the turning of the vehicle. We finally set $k_i = k_f = 10$.

We noticed that some paths that should have become straight got 360^o turns. We implemented a small trick from Thor Inge Fossen called smallest signed angle. This removed these loops to create smoother and more efficient paths. ssa was also used when finding the theta error $e_3 = ssa(\theta_d - \theta)$.

The robot sometimes fell out of the path but quickly recovered due to the feedback control mechanism. It was able to go to most of the endpoints that were tested, even those who required sharp turns and sophisticated maneuvers.

7 Further work

As the simulation used ground truth odometry, our results were probably a lot better than could be expected in real life. Simulating noisy odometry could be a good way to get a better understanding on how the algorithm would perform in the real world. From our results the linear tracker performed well, but in real life a non linear one might be better. This algorithm does not take obstacles into account, so in most real scenarios would not be sufficient.

Trajectory tracking which takes time aspects into account is not always what one is looking for. In such scenarios simpler algorithms such as LOS and ILOS may be sufficient. Implementing a planning algorithm that minimizes the energy usage could also in many cases be more interesting.

References

[1] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, Giuseppe Oriolo. *Robotics, Modelling, Planning and Control.* 2009. Springer.