Robotics Assignment: Path Smoothing and Trajectory Control in 2D Space

Objective

The objective of this project was to design a modular MATLAB–ROS2 pipeline that allows a differential drive robot (TurtleBot3) to follow a smooth 2D trajectory through a set of waypoints.
The implementation includes:

1. Path smoothing using spline interpolation.
2. Time-parameterized trajectory generation.
3. A feedback controller for trajectory tracking.
   The system is extendable to incorporate obstacle avoidance and higher-level planning.

1. Path Smoothing

Algorithm

A set of discrete waypoints

$$\text{waypoints} = \{(x_0, y_0), (x_1, y_1), \dots, (x_N, y_N)\}$$

is converted into a continuous parametric curve $x(s), y(s)$as a function of arc length s.

The function make_geom_from_waypoints.m performs this using MATLAB's cubic spline interpolation:

$$x(s) = \text{spline}(S, x_i), y(s) = \text{spline}(S, y_i)$$

where $S$is the cumulative arc length between consecutive waypoints:

$$S_i = \sum_{k=1}^{i} \quad \sqrt{(x_k - x_{k-1})^2 + (y_k - y_{k-1})^2}$$

Derivatives are computed for curvature evaluation:

$$\kappa(s) = \frac{x'(s)y''(s) - y'(s)x''(s)}{(x'(s)^2 + y'(s)^2)^{3/2}}$$

This produces a smooth geometric path represented by piecewise polynomial structures (ppx, ppy) stored inside a geometry structure geom.

Result

The smoothed path eliminates discontinuities in heading and curvature, allowing physically feasible motion for the robot.

2. Trajectory Generation

Method

Using the smoothed path, a time-parameterized trajectory is created in fcn_traj_dd.m and poly_traj_coeffs_vdes.m.

For each segment between two waypoints, we generate a cubic polynomial:

$$s(t) = a_0 + a_1(t - t_0) + a_2(t - t_0)^2 + a_3(t - t_0)^3$$

subject to:

$$\begin{cases} s(t_0) = S_0, \\ s(t_f) = S_f, \\ \dot{s}(t_0) = v_{des}, \\ \dot{s}(t_f) = v_{des} \end{cases}$$

Solving:

$$A\mathrm{a} = \mathrm{b}$$

yields the polynomial coefficients for each segment.
The time vector is uniformly sampled at interval dt, producing continuous position, velocity, and curvature profiles:

$$v_{ff} = v_{des}, \omega_{ff} = \kappa(s)\, v_{ff}$$

Outputs

Each trajectory point contains:

$$\{t, s, x, y, \theta, v_{ff}, \omega_{ff}\}$$

The nominal speed $v_{des} = L/T$ ensures consistent time scaling across the full path.

3. Trajectory Tracking Controller

Controller Logic

Implemented in fcn_controller_dd.m, the controller compares the robot's current pose

$$X = [x, y, \theta]^T$$

with the closest reference point on the trajectory.

The cross-track error in the robot frame is:

$$e_y = -\sin(\theta)(x_r - x) + \cos(\theta)(y_r - y)$$

and the heading error is:

$$e_\theta = \theta_r - \theta$$

A proportional feedback controller is used:

$$\begin{aligned} v_{cmd} &= v_{ff}, \\ \omega_{cmd} &= \omega_{ff} + k_y e_y + k_\theta e_\theta \end{aligned}$$

with tuned gains $k_y = 1.5, k_\theta = 2.0$.
The commands are saturated within physical limits:

$$v_{cmd} \in [-0.8, 0.8], \omega_{cmd} \in [-2.0, 2.0]$$

Simulation

The function dyn_diffdrive.m integrates the kinematic model:

$$\dot{x} = v\cos\ \theta, \dot{y} = v\sin\ \theta, \dot{\theta} = \omega$$

Simulation in main.m (non-ROS) and main_ros_*.m (ROS2 TurtleBot3) confirms accurate tracking with low cross-track and heading errors.

4. System Architecture and Modularity

The project is structured into reusable modules:

Module	Purpose
make_geom_from_waypoints.m	Converts discrete waypoints to a smooth spline-based geometry
eval_path.m	Evaluates x(s), y(s), derivatives, and curvature
poly_traj_coeffs_vdes.m	Computes cubic polynomial coefficients for time-based interpolation
fcn_traj_dd.m	Combines geometry and timing to create full trajectory
fcn_controller_dd.m	Feedback controller for differential drive tracking
dyn_diffdrive.m	Kinematic model used in MATLAB simulation
main.m	Offline simulation and plotting
main_ros_*.m	ROS2 interface for real or simulated TurtleBot3

Each module has independent inputs and outputs, promoting reusability.
This modular design allows easy replacement of any block—for instance, substituting a trapezoidal velocity profile or a model predictive controller without altering other functions.

5. ROS2 Integration

The ROS2 scripts (main_ros_square.m, main_ros_crcl.m, etc.) publish linear and angular velocity commands (/cmd_vel) and subscribe to odometry (/odom).
They allow real-time visualization of robot motion and data logging for comparison with the reference trajectory.

The same controller function is reused directly in ROS, confirming code modularity and testability.

6. Extension for Obstacle Avoidance

Add a separate function (for example plan_avoidance.m) that pre-processes the waypoint list. It checks each straight line segment between consecutive waypoints against known obstacles. If a segment is too close, it shifts affected waypoints outward by an offset and returns the modified list, which you then pass into this pipeline exactly like the original waypoints.

Rule:

Let $D_{\max}$be the maximum robot dimension. Set the safety offset

$$\Delta = 1.5\, D_{\max}.$$

For each segment $P_i\bar{P}_{i+1}$, compute the minimum distance to each obstacle. If

$$\text{dist}(P_i\bar{P}_{i+1},\text{obstacle}) < \Delta,$$

adjust the segment locally by moving the relevant waypoint(s) along the outward normal by Δ. The resulting modified waypoints are then fed into make_geom_from_waypoints.m with no other code changes.

7. Results Summary

Metric	Observation
Cross-track error e_y	Small oscillations (< few cm) around zero
Heading error e_θ	Smooth convergence to trajectory
Velocity commands	Within limits and stable
Path smoothness	Continuous curvature with no sharp turns
ROS performance	Stable real-time tracking on TurtleBot3 simulation

Figures generated include 2D path plots, curvature, velocity profiles, and wheel angular speed plots, verifying consistent behavior across multiple trajectory shapes (square, circle, squiggly, etc.).

8. AI Tools and Workflow

AI-powered tools such as ChatGPT were used for:

- Structuring MATLAB functions.
- Refining controller tuning and debugging errors.
- Generating documentation and performance plots efficiently.

This approach accelerated development while maintaining clear understanding of every component.

9. Conclusion

The implemented system demonstrates complete navigation functionality for a differential drive robot in MATLAB and ROS2.
The modular design separates geometry, trajectory generation, control, and visualization, making the system scalable, maintainable, and easily extendable for obstacle avoidance or real-world deployment.