Differential Drive Model Predictive Control Implementation in Python

Tanay P Kanduri

Abstract

Model Predictive Control (MPC) is a robust, optimization-based control approach suited for trajectory tracking in constrained environments. This document provides a comprehensive mathematical breakdown of MPC and a detailed Python-based implementation for controlling a differential drive robot. By predicting the trajectory of the robot over a set horizon, MPC optimizes control inputs to follow a sinusoidal reference trajectory accurately. The document also emphasizes efficient computation through lifted matrix representations and quadratic programming to meet real-time constraints.

1 Introduction

Model Predictive Control (MPC) is a control methodology that optimizes control actions based on predictions over a finite time horizon. Its widespread use in robotics, autonomous vehicle navigation, and industrial processes stems from its ability to handle multi-variable systems with constraints efficiently. Unlike conventional controllers that focus on immediate error correction, MPC's predictive capabilities allow it to take future errors into account and preemptively adjust control inputs. This characteristic is particularly beneficial in dynamic, non-linear systems such as differential drive robots, where maintaining precise control is essential despite operational constraints. This document aims to describe MPC's core mathematical principles and guide the implementation process for a differential drive robot, emphasizing tracking accuracy and control optimization.

2 Mathematical Formulation

2.1 System Model and State-Space Representation

We model the differential drive robot's dynamics using a state-space representation that captures the position and orientation of the robot over time. Define the state vector as:

$$\mathbf{x}_k = [x, y, \theta]^T$$

where:

- x and y: represent the Cartesian coordinates of the robot.
- θ: denotes the orientation angle with respect to a fixed reference frame.

The control inputs for the robot are the velocities of the left and right wheels, defined as:

$$\mathbf{u}_k = [v_{\text{left}}, v_{\text{right}}]^T.$$

Using the kinematic equations for a differential drive robot, the robot's linear velocity v and angular velocity ω can be derived from the wheel velocities:

$$v = \frac{v_{\text{left}} + v_{\text{right}}}{2},$$
$$\omega = \frac{v_{\text{right}} - v_{\text{left}}}{L},$$

where L is the distance between the wheels. The system dynamics are then defined by the following discrete-time state-space equations:

$$x_{k+1} = x_k + \Delta t \cdot v \cos(\theta_k),$$
$$y_{k+1} = y_k + \Delta t \cdot v \sin(\theta_k),$$
$$\theta_{k+1} = \theta_k + \Delta t \cdot \omega.$$

The system's evolution can be described compactly using a linearized state-space model around an operating point, allowing for the following form:

$$\mathbf{x}_{k+1} = A\mathbf{x}_k + B\mathbf{u}_k,$$

where:

- A: the state transition matrix, capturing the influence of the current state on the next state.
- B: the control matrix, representing the effect of control inputs on the state.

2.2 Objective Function for Trajectory Tracking

MPC operates by optimizing a cost function J that quantifies the tracking error over a prediction horizon f and penalizes sudden changes in control inputs over a control horizon v. Define J as follows:

$$J = \sum_{i=0}^{f} \|C\mathbf{x}_{k+i} - \mathbf{r}_{k+i}\|_{W_4}^2 + \sum_{j=0}^{v} \|\Delta\mathbf{u}_j\|_{W_3}^2$$

where:

- C: output matrix, mapping states to observed outputs.

- $\mathbf{r}_{k+i}$: reference trajectory the robot aims to follow.
- W_4: weight matrix for tracking accuracy, emphasizing positional adherence.
- W_3: weight matrix for control smoothness, discouraging excessive changes in control inputs.

The first term penalizes deviation from the reference trajectory, while the second term ensures that control inputs vary gradually, avoiding abrupt changes that could destabilize the robot.

2.3 Control Sequence Optimization

The goal of MPC is to find an optimal sequence of control inputs that minimizes J over the prediction horizon. The MPC optimization process can be formulated as a Quadratic Programming (QP) problem, which is solved at each time step to yield the optimal control actions $(v_{\text{left}}, v_{\text{right}})$ over the prediction horizon. Constraints on control inputs and state variables can be incorporated directly into the QP formulation to ensure safe and feasible operation.

2.4 Lifted Matrix Formulation for Computational Efficiency

For computational efficiency, MPC uses a lifted matrix approach, representing the optimization problem in terms of matrix operations that cover the entire prediction horizon. Define:

- **Observation Matrix** O: maps the initial state to predicted states over the horizon.
- **Control Matrix** M: captures the impact of each control input on future states.

This leads to a compact matrix form for state predictions:

$$\mathbf{X} = O\mathbf{x}_0 + M\mathbf{U},$$

where:

- $\mathbf{X}$: stacked vector of predicted states.
- $\mathbf{U}$: stacked vector of control inputs.

The cost function J can then be expressed in terms of O, M, W_3, and W_4, allowing the QP solver to efficiently minimize J across all predictions.

3 Dependencies

To implement MPC in Python, the following packages are required:

- **NumPy**: For efficient matrix and vector operations.
- **Matplotlib**: To visualize the reference trajectory and the actual trajectory followed by the robot.
- **CVXPY** or **OSQP**: Solvers for the QP problem generated by MPC.

4 Implementation Overview

4.1 Initialization of MPC

For MPC setup, initialize the following parameters:

- System matrices A, B, and C based on the linearized model.
- Prediction horizon f and control horizon v.
- Weight matrices W_3 and W_4 for balancing tracking accuracy and smooth control efforts.
- Initial state vector $\mathbf{x}_0$ and the desired reference trajectory.

4.2 Generating the Reference Trajectory

The reference trajectory consists of a sinusoidal path, defined as:

$$x_{\text{ref}}(t) = t,$$
$$y_{\text{ref}}(t) = A\sin(\omega t),$$

where A is the amplitude and ω is the frequency of the sine wave, presenting a challenging but smooth path for the robot to follow.

4.3 Control Computation and State Propagation

The MPC algorithm iteratively computes control inputs as follows:

1. Predict the robot's states over the prediction horizon using current state and control predictions.
2. Solve the QP to find optimal control inputs $(v_{\text{left}}, v_{\text{right}})$ that minimize J.
3. Apply the first control input in the sequence and update the state using the differential drive equations.

4.4 Visualization

Utilize Matplotlib to plot both the reference trajectory and the robot's actual path, allowing for a clear comparison and assessment of tracking accuracy and control performance.

5 Observations

5.1 Performance Evaluation

The following parameters influence MPC performance:

- **Tracking Accuracy**: Adjusting W_4 affects trajectory adherence. Higher values increase accuracy but may require higher control efforts.
- **Control Smoothness**: Modifying W_3 manages smoothness, balancing between sharp responsiveness and stable adjustments.
- **Prediction Horizon Impact**: Longer prediction horizons f generally improve tracking but increase computational demands.

5.2 Computational Efficiency

Efficient matrix handling, particularly with lifted matrices, allows MPC to operate within a reasonable computational scope. Optimization routines, such as quadratic programming, can enhance performance further, especially for systems with high-dimensional states and constraints.

6 Conclusion

MPC provides a structured, highly adaptable framework for trajectory tracking in robotic systems. This implementation for a differential drive robot demonstrates MPC's ability to manage complex paths, particularly those requiring fine adjustments for non-linear reference trajectories. By tuning weight matrices and prediction horizons, MPC can be adapted to various real-world scenarios. Overall, MPC's balance of foresight and optimization makes it a powerful tool in modern robotics.

7 References

- Haber, A. "Model Predictive Control (MPC) Tutorial 1: Unconstrained Formulation, Derivation, and Implementation in Python from Scratch." Available at `https://aleksandarhaber.com/model-predictive-control-mpc-tutorial-1-unconstr`