MPC Controller Design and Implementation for Optimal Trajectory with Lowest Time Consuming Based on TurtleBot.

Bowen Wang SID: 3035302336

Zheng Liu SID: 3034351831

Zack Hsu SID: 3035305820

Yuchen Zheng SID: 3035297552

Mingguang Zhou SID: 3035296681

Guanqun Huang SID:3035302674

Professor: Francesco Borrelli

Mechanical Engineering, University of California Berkeley

2019 ME C231A / EE C220B

Experiential Advanced Control Design I (Fall 2019)

MPC Controller Design and Implementation for Optimal Trajectory with Lowest Time Consuming Based on TurtleBot.

Bowen Wang, Zheng Lui, Zack Hsu, Yuchen Zheng, Mingguang Zhou, Guanqun Huang

Mechanical Engineering, University of California Berkeley

I. Abstract

The course project aims to provide better tracking accuracy when a two-wheeled robot endures inevitable noise and performs obstacle avoidance movement. Within this work, the team successfully presented two advanced algorithms of global A* search and local Model Predictive Control (MPC) to solve the optimal trajectory problem with TurtleBot: 1) applying A* to plan the path with obstacles in the way, 2) utilizing MPC to track the trajectory to the target. For the mechanical part, 1) machine operation parameters were measured experimentally, 2) numbers of tests are designed to analyze the system error of the TurtleBot fully. The nonlinear dynamic system was also calibrated to improve the accuracy of predictions. Furthermore, the project was extended to the comparison between simulation and actual robot path. By the end of the project, the whole system has been applied to the Turtlebot and perform an avoiding obstacle task successfully.

Video link: https://www.youtube.com/watch?v=aq2_gxfp9Lk

Keywords: Dynamic system, Optimal control, A* search, Model predictive control, Turtlebot, Matlab Simulink

II. Introduction

Due to their versatility and accessibility in narrow and congested areas, two-wheeled inverted pendulum (TWIP) robots with self-balancing capability have been popularly applied as a mobile platform for personal transporters, commuter vehicles, robotic wheelchairs, as well as mobile manipulators, and wheeled humanoids. In this paper, we present a motion planning system for a specific TWIP, Turtlebot. Motion planning includes global path planning and local trajectory tracking.

Currently, there exist several major algorithms for global planning. Algorithm Dijkstra solves the problem of positive weights only. Algorithm Bellman-Ford solves the problem of non-negative loops with arbitrary step sizes. Algorithm A* solves the problem of constrained trajectory and non-negative loops with any weights, which can reduce a large number of invalid calculations due to its heuristic search characteristics. As for local trajectory tracking, researchers have proposed many control methods for the trajectory tracking problem, such as PID method and LQR. Although those methods could control Turtlebot well in some respects, the tracking accuracy performance is terrible when the planned path has many small fluctuations and the Turtlebot needs to avoid hitting the obstacle. Therefore, we designed a type-int A* and nonlinear Model Prediction Controller to improve the tracking accuracy performance for the Turtlebot.

III. Experiments

A. Dynamic Model of Turtlebot

The schematic of a TWIP robot is shown in Fig. 1 with the parameters in Table 1.

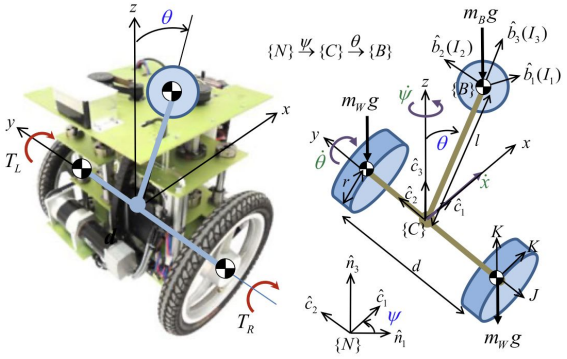

Fig. 1. Schematic of Robot Prototype

Table 1. Dynamic System Parameter of TurtleBot

Physical Properties	Symbol	Value
Mass of pendulum body [kg]	m_B	6.96
Mass of wheel [kg]	m_W	0.30
Length of pendulum [m]	l	0.208
Radius of wheel [m]	r	0.031
Distance of wheels [m]	d	0.257
Moment of Inertia of pendulum [kg-m^4]	I_1	NA
Moment of Inertia of pendulum [kg-m^4]	I_2	0
Moment of Inertia of pendulum [kg-m^4]	I_3	4.349x10^{-7}
Moment of Inertia of wheel [kg-m^4]	K	1.89x10^{-10}
Moment of Inertia of wheel [kg-m^4]	J	3.78x10^{-10}
Max linear acceleration [m/s^2]	a_{max}	0.5
Max angular acceleration [rad/s^2]	$a_{m\ max}$	3.5

Based on the dynamic model of TWIP in [1], the equation of motion could be obtained from time k to time k+1:

$$\begin{bmatrix} v_{k+1} \\ x_{k+1} \\ y_{k+1} \\ \theta_{k+1} \\ \omega_{k+1} \end{bmatrix} = \begin{bmatrix} v_k \\ x_k \\ y_k \\ \theta_k \\ \omega_k \end{bmatrix} + \begin{bmatrix} at_s \\ \left(\frac{1}{2}at_s{}^2 + v_k t_s\right) cos\mu_1 \\ \left(\frac{1}{2}at_s{}^2 + v_k t_s\right) sin\mu_1 \\ \theta_k + \psi \\ a_\omega t_s \end{bmatrix} \quad (1)$$

$$a = \frac{\left(\frac{(I_2 + m_B l^2)}{r} + m_B l\right)}{\mu_3}(T_L + T_R) \quad (2)$$

$$a_\omega = -\frac{d}{r\mu_2}(T_L - T_R) \quad (3)$$

Where the state vector $X = [v, x, y, \theta, \omega]$, v is the linear velocity of the Turtlebot, x and y are the position of the center of mass, θ is the angle between axle and x-axis, and ω is the angular velocity of the Turtlebot. The input vector $[T_L, T_R]$ represents the torque of left and right wheel motors, and t_S is the interval time valued 0.1s, ω is the angle velocity between axles of two states. The definition of other parameters can be found in appendix. Since the ROS system of the Turtlebot can support linear and angular velocity input, we only used $[T_L, T_R]$ in MPC computation and sent the optimal velocities to the Turtlebot after open-loop prediction.

B. Error Measurement

In order for the team to better understand the dynamic behavior of the TurtleBot, several experiments were conducted to measure the errors in both linear and circular movement. First goal of the experiments was to choose the proper range for linear and angular speeds with reasonable errors; secondly, the team would love to gain insight regarding the error effects on the actual traveling path and come up with a corresponding solution for possible issues. Two team members were directly involved in the experiments, and each of them was responsible for different tasks but consistent throughout the experiment session to eliminate possible measurement errors. All tests occurred in the same lab setting. Before the experiments, team assumed major error contributors could be acceleration, deceleration and friction.

1. Linear Movement Error Experiments

TurtleBot was initially tested under two speeds, 0.1 m/s and 0.2 m/s. For each speed, error measurements were taken under both 1 second and 2 seconds period (Table 1). By comparing the results, the team decided to use linear speed 0.1 m/s throughout the experiments because the error results were approximately in a linear relationship with time, which was more controllable.
In the final demo, TurtleBot was expected to run 1-2 meters in range. Team then experimented the Bot in 0.1 m/s under 5 seconds and 10 seconds (Table 2). By analyzing the table, two conclusions were made:

i. Acceleration and deceleration were major error contributors in short distance experiments but friction in normal travel was major in long distance

ii. TurtleBot has negligible errors on side deviation in linear movement.

As a result, the team concluded that in the range of 1 meter continuously straight line control, the distance error was 15%-20% or less and error for heading was negligible so it was set to be +/- 0.01 m.

2. Angular Movement Error Experiments

Team decided to follow the same linear speed for angular movement. Three error data points were recorded: heading error, linear angle error and linear distance error. A brief sketch is provided (Figure 1) to illustrate the physical meaning of the three errors. Based on performance requirements, angular speed 30 deg/s and 90 deg/s were tested and the robot was expected to travel 180 deg.

By analyzing test results (Table 3 & Table 4), it was obvious that the faster angular speed could result in higher error. To compensate for the effects of errors, the team decided to set high bound of tolerance on the planning path, especially . The exact figure will be determined later.

IV. Motion Planning

A. Global Planner A*

In the experiment, we partitioned the environment into 1000*1000 grid map and add the rectangle-shaped obstacle into the map. Considering the size of Turtlebot and movement error, we dilate the obstacle area to some extent. Therefore, we could regard Turtlebot as a particle.

Experiment result: the general version of the A* ran slowly on large grids which was measured 573.278ms from (0,0) to (999,999) in the 1000 * 1000 grid with C++. Based on the basic A*, we linearized the exact loss into discretized data with specific type int. Furthermore, we also modified the algorithm for finding the operating object with the smallest loss at each step. By using the heap priority structure, the running time decreased by 1.966ms under the same experimental conditions, meeting the MPC 10Hz frequency.

B. Local trajectory tracking

1. Open-loop prediction

In the open-loop prediction, to make the Turtlebot track the direction of the planned path, the MPC controller would find N points from the planned path as reference points based on the current position of the Turtlebot, where N is the prediction period. Then, we could build the optimal control problem using reference points:

$$\min_{X,U} (X_N - \bar{X}_N)^T P (X_N - \bar{X}_N) + \sum_{k=1}^{N-1} (X_k - \bar{X}_k)^T Q (X_k - \bar{X}_k) + \sum_{k=1}^{N-1} (U_k - U_{k-1})^T R (U_k - U_{k-1}) \quad (4)$$

$$X_0 = \bar{X}_0 \quad (5)$$

$$X_{k+1} = Dyn(X_k, U_{k+1}) \;\; k = 0 \cdots N \quad (6)$$

$$X_{lower} \le X_k \le X_{upper} \; k = 0 \cdots N \quad (7)$$

Where $\bar{X}_k$, $k = 1 \cdots N$, are reference points, U_k, $k = 0 \cdots N$, are input vectors, and U_0 is the input vector from previous open-loop prediction in order to limit the input rate. The constraints we used in open-loop prediction are the lower bound and upper bound of linear velocity, angular velocity and wheel motor torque. The parameters' value can be found in appendix.

2. Closed-loop simulation

In the closed-loop simulation, the MPC controller would get an optimal state vector from the open-loop prediction. Then, use the optimal linear velocity and angular velocity to control the Turtlebot. Next, depending on the real-time velocity feedback from ROS system, the MPC controller could calculate a new

current position, and send this position as well as the optimal input to the next open-loop prediction as X_0 and U_0. This iteration will terminate when the controller detects that the end point is not far. Then, the controller will enter a M-step optimal control to satisfy the terminal constraint $[0, \bar{x}, \bar{y}, \bar{\theta}, 0]$ instead of $[NaN, x, y, NaN, NaN]$. In every step, there is an N-scale open-loop prediction:

$$\min_{X,U} (X_N - \bar{X}_N)^T P_{new} (X_N - \bar{X}_N) + \sum_{k=1}^{N-1} (X_k - \bar{X}_N)^T Q_{new} (X_k - \bar{X}_N) + \sum_{k=1}^{N-1} (U_k - U_{k-1})^T R (U_k - U_{k-1}) \quad (8)$$

$$X_0 = \bar{X}_0 \quad (9)$$

$$X_{k+1} = Dyn(X_k, U_{k+1}) \;\; k = 0 \cdots N \quad (10)$$

$$X_{lower} \leq X_k \leq X_{upper} \; k = 0 \cdots N \quad (11)$$

We did not add the terminal constraint to the optimal control for two reasons. First, N is small compared with M, therefore, the Turtlebot could not reach the end point in N steps. Second, we only need the velocities to reduce to zero near the end point, and the M is large enough for the deceleration.

3. Reference points selection

The reference points have two functions in MPC controller, tracking the path and detecting the terminal point. Also, to avoid sharp turning and to keep the trajectory robust to some small fluctuations in the path, the reference points should distribute on a large scale. Here is the selection algorithm: First, the controller uses binary search to find the closest point to the current position from the planned path. Then, to avoid sharp turning in optimal control calculation, the first reference point will be picked after the closest point. Last, other reference points will be picked every several points starting from the first one. We set a constant distance between reference points in the selection algorithm, this value will depend on the Turtlebot's average velocity and the map scale, the detail can be found in appendix. Since the controller selects reference points in a relatively large scale, the MPC iteration will terminate and enter a deceleration MPC if the last reference point overtakes the end point.

V. Results and Discussions

The trajectory comparison between Matlab simulation and real Turtlebot based on the planned path is shown in Fig.2 as well as five obstacles. To get the trajectory of the Turtlebot, we assumed the real-time velocity feedback from the Turtlebot encoder as the ground truth, then calculated the positions every interval time 0.1s from the start point.

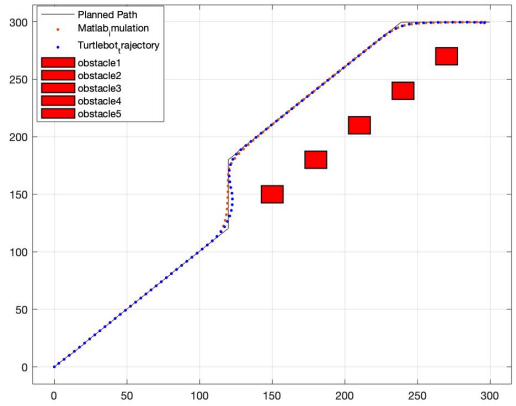

Fig. 2. Matlab simulation versus real Turtlebot trajectory.

As shown in the picture, red-points trajectory represents the Matlab simulation result with random noise added to velocities, and blue-points trajectory is the Turtlebot record result. Both of the two trajectories match the planned path pretty well at the straight part and have a small drift within 5 cm at the turning part. And the A* algorithm has already considered this error when generating the planned path.

VI. Conclusions

In this project, the team implements the whole motion planning system of a specific TWIP, Turtlebot. Global planner A* search gives out an optimal path to avoid obstacles and Model Predictive Control is used to track the path accurately. To apply to the Turtlebot, several tests are conducted to analyze the dynamic model and the motion error of the Turtlebot. The results show that both Matlab simulation and real Turtlebot trajectory have a good tracking accuracy. Further improvement will focus on the robust of the system to different motion scenarios such as obstacle avoiding at different maximum velocity or at different path condition. If possible, we will add a perception stack using lidar to build a real-time obstacle avoiding control system.

VII. References

[1] S. Kim and S. Kwon, "Nonlinear Optimal Control Design for Underactuated Two-Wheeled Inverted Pendulum Mobile Platform," in IEEE/ASME Transactions on Mechatronics, vol. 22, no. 6, pp. 2803-2808, Dec. 2017.
doi: 10.1109/TMECH.2017.2767085

[2] J. Petereit, T. Emter, C. W. Frey, T. Kopfstedt and A. Beutel, "Application of Hybrid A* to an Autonomous Mobile Robot for Path Planning in Unstructured Outdoor Environments," ROBOTIK 2012; 7th German Conference on Robotics, Munich, Germany, 2012, pp. 1-6.

APPENDIX

A. DYNAMIC MODEL OF TURTLEBOT

$$\mu_3 = \left(m_B + 2m_W + 2\frac{J}{r^2}\right)(I_2 + m_B l^2) - {m_B}^2 l^2$$

$$\mu_2 = I_3 + 2K + 2\left(m_W + \frac{J}{r^2}\right)d^2$$

$$\mu_1 = 90^o - \theta_k + \frac{\psi}{2}$$

$$\psi = \frac{1}{2}a_\omega {t_s}^2 + \omega_k t_s$$

B. OPEN-LOOP PREDICTION PARAMETERS

$$P = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 30 & 0 & 0 & 0 \\ 0 & 0 & 30 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, Q = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 100 & 0 & 0 & 0 \\ 0 & 0 & 100 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$R = \begin{bmatrix} 0.01 & 0 \\ 0 & 0.01 \end{bmatrix}, N = 5, t_s = 0.1s$$

C. DECELERATION MPC PARAMETERS

$$P_{new} = \begin{bmatrix} 10 & 0 & 0 & 0 & 0 \\ 0 & 30 & 0 & 0 & 0 \\ 0 & 0 & 30 & 0 & 0 \\ 0 & 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 0 & 10 \end{bmatrix}, \qquad Q_{new} = 2.0 * P$$

D. ERROR MEASUREMENT FIGURES AND TABLES

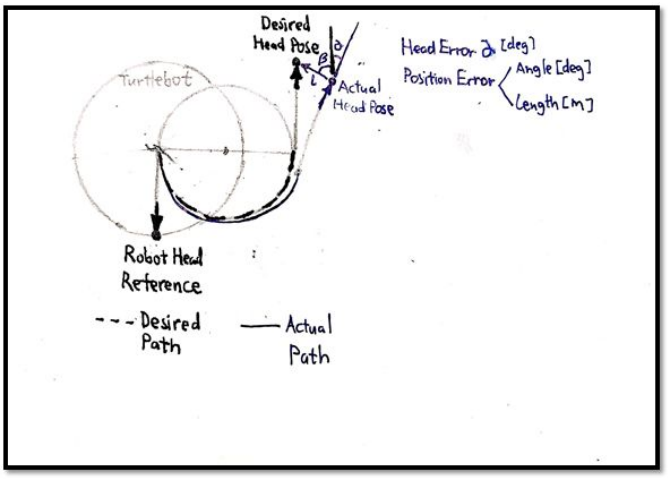

Figure 1: Angular Movement Experiment Sketch

Table 1: Errors Measurements 0.1 m/s VS 0.2 m/s

	Speed: 0.1 m/s	Speed: 0.2 m/s
Time: 1s	0.023 m (behind)	0.028 m (behind)
Time: 2s	0.047 m (behind)	0.078 m (behind)
Time: 3s	0.069 m (behind)	0.108 m (behind)

Table 2: Errors Measurements 0.1 m/s for 5s and 10s

Speed: 0.1 m/s	Time: 5 seconds	Time: 10 seconds
Test #1	0.097 m (behind) 0.009 m (right)	0.155 m (behind) 0.003 m (right)
Test #2	0.103 m (behind) 0.000 m (right)	0.159 m (behind) -0.025 m (right)
Test #3	0.099 m (behind) 0.000 m (right)	0.152 m (behind) 0.013 m (right)
Test #4	0.091 m (behind) 0.000 m (right)	0.148 m (behind) 0.009 m (right)
Test #5	0.100 m (behind) -0.005 m (right)	0.157 m (behind) 0.000 m (right)

Table 3: Errors Measurements under 30 deg/s

Angular Speed: 30 deg/s	Test #1	Test #2	Test #3
Heading Angle Error (alpha: deg)	24.62	22.62	28.07
Linear Angle Error (beta: deg)	27.82	24.62	24.62
Linear Distance Error Behind (m)	0.095	0.089	0.096

Table 4: Errors Measurements under 90 deg/s

Angular Speed: 30 deg/s	Test #1	Test #2	Test #3
Heading Angle Error (alpha: deg)	39.72	36.25	33.56
Linear Angle Error Beta (deg)	60	57.87	58.97
Linear Distance Error Behind (m)	0.103	0.104	0.107