# Design of Autonomous Delivery Using Time Optimal MPC Controller with Obstacle Avoidance

Chenhang Yuan (3038560631) Pinyun Hung (3038575698) Shixin Yan(3038581600) Yujie Zhang(3038574632) (Video link: https://youtu.be/qGjSPMXfcAU)

***Abstract- This paper seeks to design a time optimal controller for autonomous delivery to replace deliverer in a restaurant, considering real-world case including obstacle avoidance (e.g., pedestrian), proportionality between velocity (linear and angular) and load.***

## I. INTRODUCTION

Autonomous robots have been implemented in restaurants to do the work of delivering foods and drinks**.** Current delivery robots in the restaurants cannot address on time optimal purposes and velocity constraints due to load. Also, when counter dynamic obstacles such as people walking in the planned path, instead of detouring for the purpose of time optimal, the current delivery robot would stop and wait for the moving obstacles to be out of the way. The Model Predictive Control Algorithm allows us to plan for time optimal trajectory while having velocity constraints and dynamic obstacle constraints. In this paper, we implemented the MPC controller in simulations with the assumption of a perfect unicycle robot model with no noise and designated path without environmental disturbance. A real-world restaurant map is used to generate the environment with static obstacles such as walls and tables. Then, A-star algorithm is implemented to generate the reference trajectory for the MPC controller. To avoid dynamic obstacles, we implemented OBCA, optimization-based collision avoidance method developed by Xiaojing Zhang and his college [1]. Within OBCA, an MPC controller is designed with dynamic obstacle constraints and static obstacle constraints. The optimizer outputs the time-optimized trajectory and optimum input.

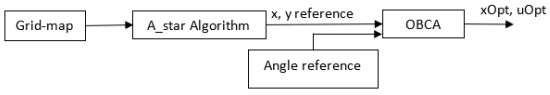


*Figure 1. Flow chart of obstacle avoidance method*

## II. ENVIROMENT

A map of the restaurant is generated using grid method where the static obstacles including the walls and tables are represented as 1, and the free space is represented as 0 in the grid map. Then with a start point and goal point setup, we used A-star algorithm to compute a shortest route as the reference path. As shown in figure one, the static obstacles are shown in light green, and the free space is shown in dark green. The start points shown as a yellow star is where the kitchen located, so the robot will pick up food at this start point. A red star represents the goal point which is the destination for the delivery robot. The route computed by A-star algorithm is shown in black line.

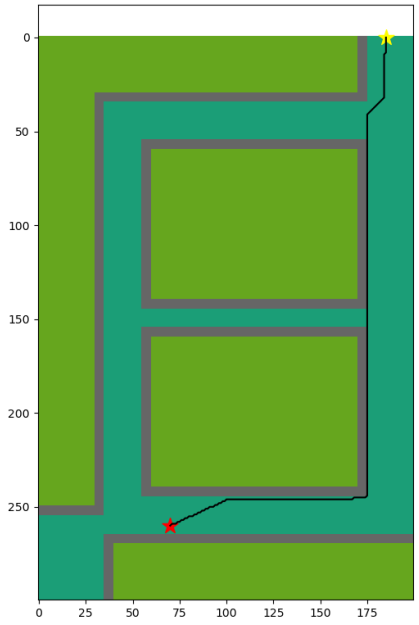


*Figure 2. Grid map with reference route*

## III. MODEL DYNAMICS

In this case, we consider the dynamic of the delivery cart as a vehicle, so the dynamic model is nonlinear continuous-time model shown below,

$$\dot{x} = v\cos(\psi)$$
$$\dot{y} = v\sin(\psi) \qquad (1)$$

where x and y are the global coordinates in grid map, v is the velocity of the robot and ψ is the angle of current velocity with respect to the global x-axis. The system is discretized using forward Euler discretization with time step $T_s$,

$$\begin{aligned} x(k+1) &= x(k) + T_s * v(k)\cos(\psi(k)) \\ y(k+1) &= y(k) + T_s * v(k)\sin(\psi(k)) \\ \psi(k+1) &= \psi(k) + Ts * \dot{\psi}(k) \end{aligned} \qquad (2)$$

The state vector $z = [x, y, \psi]^T$, and the input vector $u = [v, \dot{\psi}]^T$.

## IV. PATH PLANNING & OBSTACLE AVOIDANCE

The desired position vector is obtained from the reference tractor computed by A-star algorithm, and the third reference state $\psi$ is obtained from the angle of the path when the position state changes.

The finite time optimal control problem is set up as shown below. The cost function is in quadratic form including the transient costs and the terminal cost. In this paper, we assume that there are no disturbances and that the system will be subject to input and state constraints.

We need to find a set of control sequences that avoid obstacles with a restricted speed and angle, and the final state reaches the target state while optimizing the objective equation.

$$\min_{z,u,\lambda,\mu,T} \bar{z}_N{}^T P \bar{z}_N + \sum_{k=0}^{N-1} \bar{z}_k{}^T Q \bar{z}_k + u_k{}^T R_1 u_k + \frac{\bar{u}_k{}^T R_2 \bar{u}_k}{T_s{}^2} + qT$$

$$\begin{aligned} s.t. \quad & z_0 = z(0), z_N = z_F, z_{k+1} = f(z_k, u_k), \\ & z_L \le z_k \le z_U,\ u_L \le u_k \le u_U, 0 < T \le T_{max}, \\ & -g^T \mu_k^{(m)} + \left(A^{(m)} t(z_k) - b^{(m)}\right)^T \lambda_k^{(m)} > 0, \\ & G^T \mu_k^{(m)} + R(z_k)^T A^{(m)^T} \lambda_k^{(m)} = 0, \\ & ||A^{(m)^T} \lambda_k^{(m)}||_* \le 1, \lambda_k^{(m)}, \mu_k^{(m)} > 0, \\ & \text{for } k = 0, \dots, N, m = 1, \dots, M, \end{aligned} \qquad (3)$$

We add the optimization parameter T inside the cost equation, whose cost q is also calculated in the constraint. $T_{max}$ is calculated from the Manhattan distance of the divided grid. We require the trajectory to reach the target $z_F$ in a fixed number of steps N.

A* can find obstacle-free paths by gridding the location space. The advantage of the A* search algorithm is its simple logic and fast computation, so we use A* to provide an initial predicted trajectory for reducing the large number of repeated computations caused by complex maps.

We first grid the map that will be designed according to the true size and create nodes at each node. the A* algorithm will calculate the cost of each neighboring node from the starting point and the final cost is the sum of the costs of each step. The optimal route planning is the minimum of the open set list and A* only analyzes the nodes of the optimal trajectory. Figure 2 shows a simple trajectory map using A*, which handles the optimal path problem for complex maps well when only static obstacles are present.

$\lambda_k^{(m)}$ and $\mu_k^{(m)}$ are a pair of dyadic variables that we introduce as constraints for obstacle avoidance. m denotes one of the obstacles, $\lambda_k^{(m)}$ is the associated dyadic variable, and $A^{(m)}$ and $b^{(m)}$ are the inequality constraints for each obstacle. Since we mainly use rectangular obstacles, our constraints are the surfaces of the obstacles, and we mainly extract the positions of the four corners of the rectangular body.

Our system is nonlinear, and the trajectory calculated using MPC is dynamically feasible for one side of the case. As shown in Figs. 3, 4, the cart avoids dynamic obstacles during the motion.

## V. PERFORMANCE

In this section, we show different solutions optimizer generated using different strategies, including A*, Open Loop, and Closed loop.

### *A. Closed Loop MPC with reference tracking from A**

To tackle the problem of sensitive feasibility due to lack of initial set and terminal constraints, we tuned the Q and R matrices as well as optimum horizon N to fit feasibility. We put A*, Open Loop, and Closed Loop strategies into same map, with Open Loop and Closed Loop utilizing the states generated from A* for reference tracking in cost functions. OBCA, as presented in [1], is designed and utilized to avoid both dynamic and static obstacles, where we defined static obstacles as dining tables or walls in a restaurant in real cases.

For closed loop MPC, we developed a simulative LIDAR sensor with radius of 8 meters for detection of obstacle as marked with green circle. Blue dotted line is marked as the overall reference route plan generated from A*, orange dots are

marked as object car's past route, and purple dots marked as N+1 open loop trajectory planned within each closed loop. Spend Time is time spend by the car to achieve corresponding positions as shown in the pictures below. For simulation, there are two MPCs solving problems involving static obstacles only cases and both static and dynamic obstacles cases with same optimum horizon N of 5. For static obstacles cases, Q is set as 0.5 * $I_3$, R for input is set as 0.01 * $I_2$, and R for acceleration is set as 0.1 * $I_2$. In contrast for both static and dynamic obstacles cases, Q is set as 0.001 * $I_3$, R for input is set as 0.01 * $I_2$, R for acceleration is set as 1* $I_2$.

The car first starts with the first MPC solving static obstacles only while LIDAR is turned on for detection of dynamic obstacles. As LIDAR detects a dynamic obstacle within 8 meters around the car, it will switch to the second MPC solving with both static and dynamic obstacles. While switching to the second MPC, terminal set will be modified with respect to closest reference point from A*, LIDAR range, and optimum horizon. Furthermore, as dynamic obstacles move out of LIDAR's range, it will switch back to static obstacle avoidance MPC.

Also, Ts, which is sampling time, for closed loop MPC is not identical within each loop. During simulation, we detected that there is possibility of infeasibility with constant sampling time due to terminal constraints. As A* is not smoothened and to solve the possibility that terminal constraint might not be satisfied, we will adjust sampling time within MPC so that terminal constraint can be met and inherit the previous feasible sampling time to the next loop. As we switch from MPC solving static obstacles to MPC solving both static and dynamic obstacles, Ts will be imported from the first MPC to the second to satisfy terminal constraint.

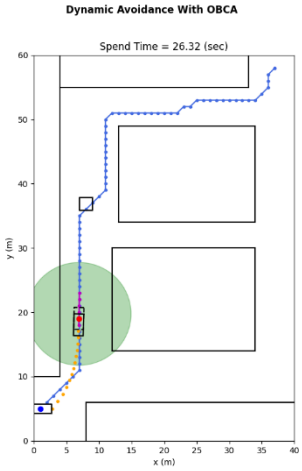


*Figure.3 Closed Loop MPC route with static obstacle avoidance when LIDAR does not detect a dynamic obstacle.*

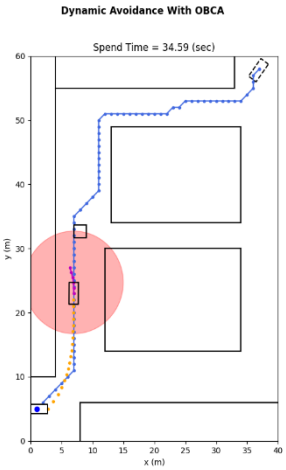


*Figure 4. Closed Loop MPC route with static obstacle avoidance when LIDAR detects a dynamic obstacle and is planning to avoid it.*

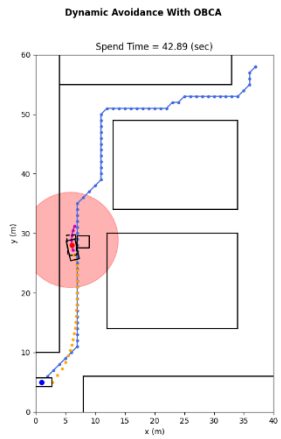


*Figure 5. Closed Loop MPC route with static obstacle avoidance when LIDAR detects a dynamic obstacle and has planned to avoid it.*

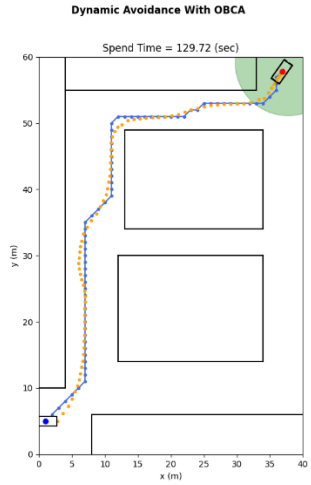


*Figure 6.Closed Loop MPC route with static obstacle avoidance when car achieves goal position.*

### *B. A*, Open Loop MPC, Closed Loop MPC states.*

The following figures show different states with respect to total simulation loops.

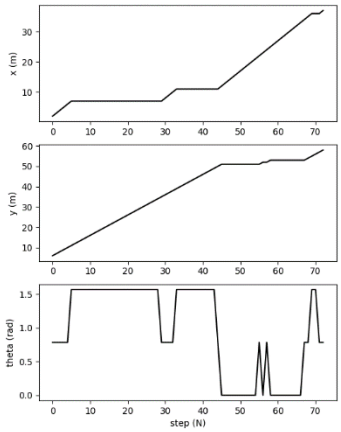


*Figure 7. A* Planning states in N steps*

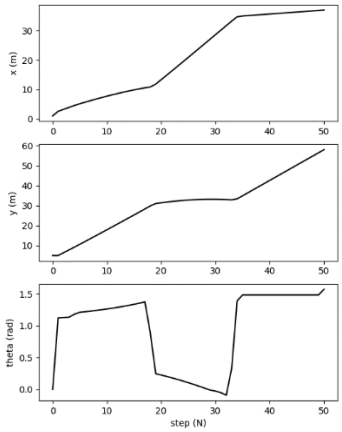


*Figure 8. Open Loop MPC states in N steps*

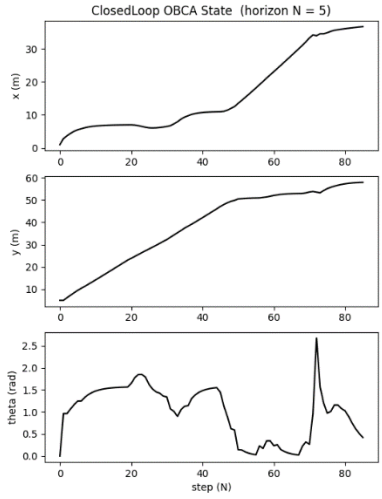


*Figure 9. Closed Loop MPC states in simulation with N = 5*

### *C. Open Loop MPC with static obstacle avoidance only vs A**

For the interest of removing reference tracking from A* in MPC, we simulated OBCA MPC with only defined initial states and terminal states. In this case, optimum horizon N is set at 5, Q is set as 0.5 * $I_3$, R for input is set as 0.01 * $I_2$, and R for acceleration is set as 0.1 * $I_2$.

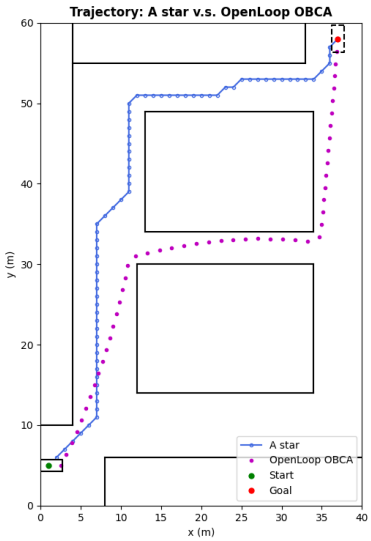


*Figure 10. Open Loop MPC with static obstacles avoidance only vs A* planning*

As shown in Figure 10, without reference tracking, OBCA MPC tends to avoid route on the top left corner.

## VI. CONCLUSION

As shown in the figures and explanations above, OBCA MPC controls successfully generate a closed loop simulation for our dining car, with given reference trajectory generated from A*, start position and terminal position as we defined. With reference trajectory given prior to closed loop simulation, dining car can achieve the terminal position along with tracking A* route and avoid both static and dynamic obstacles. As we expand our project, we are aiming to extend our map and involve more elements in our simulation, including more dynamic obstacles such as simulation of servers or guests with irregular movements, and more complex map layout.

Furthermore, we will expand our dynamic obstacles avoidance cases into more complex situations. As shown in Figure 11, OBCA MPC can avoid two dynamic obstacles situations when it can detect the second obstacle after it goes totally past the first one.

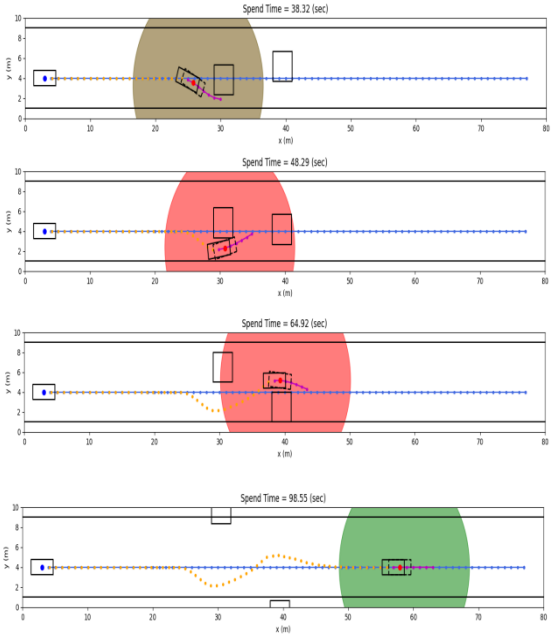


*Figure 11. OBCA MPC solving two dynamic obstacles, when LIDAR can only detect the second obstacle after it goes past the first one.*

The second situation is like the first one with tiny differences. If our dining car first avoids a static obstacle, as shown in figure 12, with first MPC solving static obstacle only and then detect the dynamic obstacle blocking its reference route. It can switch and solve dynamic obstacle cases.

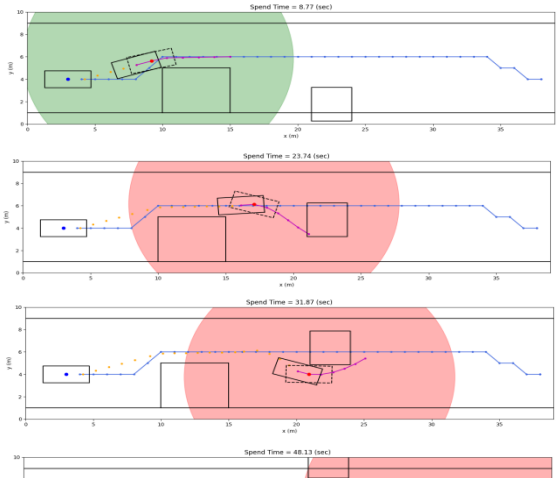

*Figure 12.OBCA MPC solving one big static obstacle and one dynamic obstacle, when LIDAR can only detect the second obstacle after it goes past the first one.*

## REFERENCE

[1] Zhang, X; Liniger, A; Borrelli, F. (2017).
"Optimization-Based Collision Avoidance".