International Journal of Advanced Robotic Syst

MLP-PSO: A Lightweight Optimization Algorithm for the MPC Local Planner

Journal:	*International Journal of Advanced Robotic Systems*
Manuscript ID	ARX-24-0078
Manuscript Type:	Research Article
Keywords:	Optimization Algorithm, MPC local planner, Motion Planning, Multi-layer Perceptron, Particle Swarm Optimization
ARX Research Topics:	AUTONOMOUS CONTROL < ROBOT MANIPULATION AND CONTROL, WHEELED MOBILE ROBOTS < MOBILE ROBOTS AND MULTI-ROBOT SYSTEMS, PATH PLANNING AND NAVIGATION < MOBILE ROBOTS AND MULTI-ROBOT SYSTEMS, MOBILITY AND MOTION PLANNING < SERVICE ROBOTICS
Abstract:	The model predictive trajectory (MPC) planner is a popular and effective robot local motion planner. However, it is challenging to satisfy real-time requirements and implement them on embedded platforms due to its high solving complexity and reliance on optimization solvers. This letter reports a lightweight and efficient two-stage solving algorithm for the MPC planner. Firstly, the general form of the MPC local planning problem was specified and simplified by the motion primitives. Then, a two-stage solving method of multi-layer perceptron (MLP) pre-solving and particle swarm optimization (PSO) re-optimizing is developed after splitting the cost function into two pieces. An MLP neural network was designed and trained offline to learn the solution of the MPC local planner without considering obstacles after selecting the inputs and outputs. Next, to accomplish obstacle avoidance, the PSO algorithm re-optimizes the trajectory based on the outputs of the neural network. The experiment results demonstrate that the MLP-PSO algorithm can quickly and accurately solve local planning problems, guiding robots to complete global paths with the same efficiency as expert solvers. The average solving time has been reduced by over 90%, enabling the robot to increase its control frequency or adopt higher-quality complex motion primitives. The MLP-PSO algorithm also can be used for various robots and motion primitives, with a wide range of application potential.

MLP-PSO: A Lightweight Optimization Algorithm for the MPC Local Planner

Journal Title
XX(X):1–10

DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Abstract

The model predictive trajectory (MPC) planner is a popular and effective robot local motion planner. However, it is challenging to satisfy real-time requirements and implement them on embedded platforms due to its high solving complexity and reliance on optimization solvers. This letter reports a lightweight and efficient two-stage solving algorithm for the MPC planner. Firstly, the general form of the MPC local planning problem was specified and simplified by the motion primitives. Then, a two-stage solving method of multi-layer perceptron (MLP) pre-solving and particle swarm optimization (PSO) re-optimizing is developed after splitting the cost function into two pieces. An MLP neural network was designed and trained offline to learn the solution of the MPC local planner without considering obstacles after selecting the inputs and outputs. Next, to accomplish obstacle avoidance, the PSO algorithm re-optimizes the trajectory based on the outputs of the neural network. The experiment results demonstrate that the MLP-PSO algorithm can quickly and accurately solve local planning problems, guiding robots to complete global paths with the same efficiency as expert solvers. The average solving time has been reduced by over 90%, enabling the robot to increase its control frequency or adopt higher-quality complex motion primitives. The MLP-PSO algorithm also can be used for various robots and motion primitives, with a wide range of application potential.

Keywords

Optimization Algorithm, MPC local planner, Motion Planning, Multi-layer Perceptron, Particle Swarm Optimization

Introduction

Motion planning is an important part of the robot system, bridging the gap between task execution and attitude control, which can be divided into global and local motion planning Howard et al. (2014). Local motion planning is responsible for guiding the robot to follow the global path efficiently and stably in shifting environments with a detailed kino-dynamics model.

Optimal-based methods, one type of commonly used method, transform the problem into a high-dimensional constrained optimization problem, such as the time elastic band (TEB) Rsmann et al. (2017), MPC Howard et al. (2014), polynomial curve method Yang et al. (2021), Bézier curve method Yang and Sukkarieh (2010), etc. In recent years, MPC local planner has been studied intensively and applied successfully on robots such as unicycles Ulrich and Borenstein (2002), unmanned aerial Lai et al. (2019); Ji et al. (2020), ground Fnadi et al. (2019); Williams et al. (2016), and ships Zhu et al. (2020) vehicles. The forward prediction module enhances the completeness of trajectories and the stability of robots' attitudes, while rolling optimization permits the planner to adapt to environmental changes and correct numerous errors, resulting in high-quality trajectories and an improvement in the robot's stability and robustness. However, the MPC trajectory problem of robots is frequently a nonlinear optimization problem with multiple constraints, making it challenging to satisfy the real-time operation requirements. Incorporating the high-dimensional dynamic model and extending the predicted horizon can further enhance the planners' performance, improving fast-moving Fnadi et al. (2019) and obstacle avoidance Lindqvist et al. (2020) abilities, even achieving aggressive driving in hard field environments Williams et al. (2016), but increasing the solving complexity. Linearization is a commonly used technique to reduce optimization complexity but at the expense of solution quality Fnadi et al. (2019); Lindqvist et al. (2020). The model predictive path integral (MPPI) method gets the optimal trajectory by parallel sampling calculation but puts requirements on hardware such as graph processing unit (GPU) Williams et al. (2016). Obstacle avoidance makes optimization more challenging to meet the real-time requirement. Hard constraint methods, such as the corridor method Ji et al. (2020), add more constraints, while soft constraint methods complicate the optimization gradient Yang et al. (2021). Nonlinear optimization problems usually require professional solving libraries such as IPOPT Wächter and Biegler (2006), CasADi Andersson et al. (2019), and Acados Verschueren et al. (2021), which are difficult to deploy on embedded platforms or lightweight operating systems. While the volume of high-performance computers is not conducive to the miniaturization of robots.

Sampling-based motion is another type of local planner that has good real-time performance at the expense of optimality. Some relevant technologies have been introduced to expedite the solving of the MPC local planner. First, Motion Primitive Howard et al. (2014) uses a few parameters to encode long trajectories, to construct an efficient search tree to find the optimal trajectory Paranjape et al. (2015); Lopez and How (2017); Schwesinger et al. (2013). The designated motion primitives generate the mapping between state and action space. Robot models were taken into account to assure trajectory implementability when designing motion primitives. Some researchers have used

optimization techniques to find the optimal trajectory in the motion primitive space Ferguson et al. (2008). Low space dimension enables quick optimization solutions, but multiple times solving the boundary value problem (BVP) Mueller et al. (2015); Paranjape et al. (2015); Lopez and How (2017) which converts motion primitives into trajectories, incurs additional time consumption. The second technique is the pre-solving dataset and look-up method Liniger et al. (2015); Yang et al. (2017). Sample and solve to construct a trajectory dataset offline at first. In actual operation, the best trajectory can be quickly sought in the table. However, this method has two disadvantages: the solution quality is limited by the sampling precision. Reducing the sampling interval may increase the consumption of storage space and the difficulty of searching Frazzoli et al. (2005). Neural networks and imitation learning technologies have the potential to solve these issues. Using trajectory datasets as a guide, neural networks are capable of learning trajectory planning abilities due to their ability to match any nonlinear function. Currently, researchers have used neural networks to generate trajectories of robots with starting and ending points Lin et al. (2019) or get the planning skills from human manipulation datasets Ly and Akhloufi (2020).

The environment in which robots are located is continually changing, posing challenges to the input of environmental data into neural networks to accomplish obstacle avoidance. Some scholars use deep neural networks to process images Saleh et al. (2018) and point cloud Zhang et al. (2016) information to avoid obstacles, but they often need human teaching Ly and Akhloufi (2020) or reinforcement learning Zhang et al. (2016). Worse, the sampling and training are difficult and have no advantage in solving time. To accomplish obstacle avoidance, we adopted a re-optimization to construct a two-stage solution, which can simplify complex problems by focusing on different aspects during different stages of the solution Eiras et al. (2021), producing effective solving.

The innovations of this letter can be summed up as follows: ① For the model prediction trajectory planner, a lightweight fast-solving algorithm called MLP-PSO is proposed with two stages: pre-solving and re-optimization, which doesn't require the complex code library and can be implemented on lightweight processors. ② A multi-layer perceptron (MLP) network was designed and trained to imitate the solvers without considering obstacles to achieve fast and accurate optimization. ③ Experiments have demonstrated that this algorithm can precisely solve MPC motion planning problems while reducing average optimization time by at least 90%, enabling the robot to increase its control frequency or adopt higher-quality complex motion primitives. Moreover, this algorithm is suitable for various robots and motion primitives.

MPC Local Planner with Motion Primitive

This section begins with an overview of the mobile robot's planning and control system. Then, establish and simplify the MPC local planning problem with motion primitives.

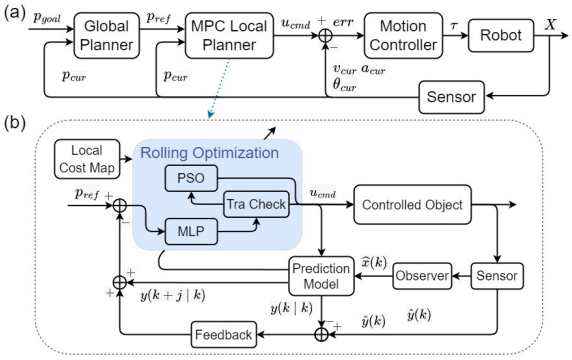

Figure 1. (a) The whole framework of robot planning and control system (b) The framework of MPC local planner with MLP-PSO algorithm.

System Framework

As depicted in Fig. 1(a), the planning and control system of mobile robots often utilizes a hierarchical control structure. The global planner is responsible for finding feasible paths between the current location and the terminal. Based on the global path and the local cost map, the MPC local planner generates smooth trajectories and motion commands, avoiding time-varying obstacles. The motion controller executes commands and eliminates interference.

Fig. 1(b) shows the structure of the MPC local planner. After obtaining global path p_{ref}, local cost map, and robot state information from sensors and observers, the planner provides the optimal trajectory and control sequence through forward prediction and rolling optimization.

Motion Primitive

The robot can be described by the following model:

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}) \quad \{h(\mathbf{x}, \mathbf{u}) = 0,\ \hat{h}(\mathbf{x}, \mathbf{u}) \leq 0,\ \mathrm{x} \notin \mathcal{O}\} \qquad (1)$$

where $\mathbf{x} \in \mathcal{X}$ represents the robot's state and $\mathbf{u} \in \mathcal{U}$ denotes its input. The h and $\hat{h}$ represent the collection of state and input constraints. The last equation is the collision-free constraint, where $\mathcal{O}$ is the obstacle set.

If the robots' inputs are generated according to specific rules of the motion primitive, the entire input sequence can be reproduced by solving the BVP problem with initial and terminal states. Then, the robot's state sequence can be predicted by the kino-dynamic model. Therefore, there is a one-to-one correspondence between the trajectory and terminal system states Howard et al. (2014); Lai et al. (2019). Using terminal state as a motion primitive $\boldsymbol{\xi}$, we can build the mapping between $\boldsymbol{\xi}$ and trajectories:

$$\mathcal{S} : \langle \mathbf{x}_0, \boldsymbol{\xi} \rangle \rightarrow \langle \hat{\mathbf{u}}(t), \hat{\mathbf{x}}(t), \hat{t}_f \rangle \qquad (2)$$

Model Predictive Local Planning

Model predictive local planning problem can be expressed in the following general form of MPC problem:

$$\min_{\mathbf{x}(t), \mathbf{u}(t), t_f} J = \Phi(\mathbf{x}(t_f)) + \int_{t_0}^{t_f} L(\mathbf{x}(t), \mathbf{u}(t), t_f)\, dt \qquad (3)$$

By introducing the motion primitive, the complexity of the MPC problem can be reduced significantly by decreasing the

number of optimization variables:

$$\min_{\boldsymbol{\xi}} J = \overline{\Phi}(\boldsymbol{\xi}) + \int_{t_0}^{t_f} \overline{L}(\boldsymbol{\xi})\, dt \tag{4}$$

The cost function is determined by the purposes of the MPC problem. In general, the local planner ensures the robot can track global paths with reference speed, minimum energy, stable attitude, and collision-free. In practical use, the MPC problem needs to be expressed in a discrete form. The model predictive local planning problem is defined as follows:

$$\begin{aligned} \min_{\boldsymbol{\xi}(\cdot)} \; J(\cdot) =& \sum_{i=1}^{N} \overline{\boldsymbol{X}}_i'^{T} H_1 \overline{\boldsymbol{X}}_i' + \Delta\boldsymbol{\xi}'^{T} H_2 \Delta\boldsymbol{\xi}' \\ &+ H_3 \sum_{i=1}^{N} MAP_{\boldsymbol{X}(i|k)} \end{aligned} \tag{5}$$

$$\begin{aligned} \text{s.t.} \;\; & \boldsymbol{X} = (x, y, \varphi, v, \dot{v}, \ddot{v}, ..., \omega, \dot{\omega}, \ddot{\omega}, ...)^T; \quad \boldsymbol{X}(0|k) = \hat{\boldsymbol{X}}_k; \\ & \boldsymbol{X}' = (x, y, \varphi)^T; \quad \overline{\boldsymbol{X}}_i' = \boldsymbol{X}'(i|k) - \boldsymbol{X}_{ref}'(i|k); \\ & \boldsymbol{\xi} = (v_{ter}, \omega_{ter})^T; \\ & \Delta\boldsymbol{\xi}' = (v_{ter}(k),\ \omega_{ter}(k)) - (v_{max}(k),\ \omega_{ter}(k-1)); \\ & \boldsymbol{X}(i|k) \in [\boldsymbol{X}_{min},\ \boldsymbol{X}_{max}]; \quad \boldsymbol{\xi} \in [\boldsymbol{\xi}_{min}, \boldsymbol{\xi}_{max}]; \\ & \boldsymbol{G}(\boldsymbol{X}(i|k),\ \boldsymbol{\xi}) = 0; \quad \boldsymbol{F}(\boldsymbol{X}(i|k),\ \boldsymbol{\xi}) < 0; \end{aligned}$$

Where N is the predicted horizon, H_1, H_2, H_3 is the cost coefficient matrix, and $MAP_{\boldsymbol{X}(i|k)}$ is the grid value of (x, y) on the local cost map. To assure the smoothness of the trajectory, the robot's state vector should include velocity v, angular velocity ω, and their higher derivatives $(\dot{v}, \ddot{v}, ..., \dot{\omega}, \ddot{\omega}, ...)$ in addition to the position vector (x, y, φ). To concisely express the trajectory, the terminal velocity v_{ter} and angular velocity ω_{ter} are chosen as the motion primitives, while their higher-order derivatives default to 0 and can be disregarded.

Focus on the cost function, the first item enables the robot to follow the global path, while the second item enables the robot to drive at the reference speed with minimal turning consumption. Finally, the local cost map created and renewed by signed distance field (SDF) technology is introduced to implement gentle constraints for obstacle avoidance. The Local cost map rasterizes the environment, resulting in the lower the grid value, the further it is to the obstacle. So minimizing this item can guide the robot away from obstacles. The difference between grids generates gradients for optimization.

MLP-PSO Algorithm

In this section, the overview of the MLP-PSO algorithm is described first, followed by the details of each module.

Algorithm Overview

The cost function of MPC local planner can be divided into two parts as follows.

$$J_1(\cdot) = \sum_{i=1}^{N} \overline{\boldsymbol{X}}_i'^{T} H_1 \overline{\boldsymbol{X}}_i' + \Delta\boldsymbol{\xi}'^{T} H_2 \Delta\boldsymbol{\xi}' \tag{6}$$

$$J_2(\cdot) = H_3 \sum_{i=1}^{N} MAP_{\boldsymbol{X}(i|k)} \tag{7}$$

$J_1(\cdot)$ ensures the robot can track the global path with a stable attitude, which is only related to robot states. While $J_2(\cdot)$ lets the robot avoid obstacles, changing by the environment. $MAP_{\boldsymbol{X}(i|k)}$ is the cost value of $\boldsymbol{X}(i|k)$ in the local cost map. Fig. 2 shows that the value of the whole cost function is the superposition of two parts.

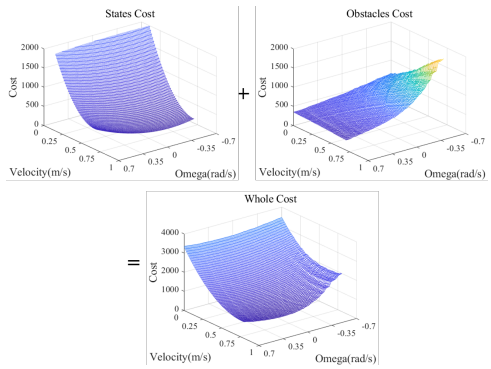

Figure 2. The cost function value of one real problem. The whole cost (under) is the superposition of states cost $J_1(\cdot)$ (upper left) and obstacles cost $J_2(\cdot)$ (Upper right). The x and y axis represent v_{ter} and ω_{ter} in ξ.

If the MPC problem is solved in an obstacle-free environment, the cost function degenerates into $J_1(\cdot)$. And if the optimal trajectory exists in the obstacle-free area, $J_2(\cdot)$ can also be ignored. The MLP-PSO algorithm, as depicted in Fig. 1(B) and **Algorithm 1**, is a two-stage approach for solving the MPC planning problem. An MLP neural network is introduced for the pre-solving stage, while the re-optimization stage is finished by the PSO algorithm.

1) **MLP pre-solving stage:** Ignoring environment and concentrating on the $J_1(\cdot)$ which is in a fixed state space, reduces the input dimension, training difficulty, and sampling complexity of the MLP network. Mathematically, the optimal result of $J_1(\cdot)$ is a high-dimensional nonlinear function involving the system states and the objective states. Since neural networks are capable of fitting any nonlinear function, they can be utilized to fit the result function. We offline-trained an MLP network to learn the optimization result function of the $J_1(\cdot)$ under constraints, achieving a fast, accurate, and space-efficient solving. The MLP neural network can provide motion primitives directly by taking the current state and target state of the robot as inputs.

2) **PSO re-optimization:** As Fig. 2, the cost of obstacles provides new gradients for optimization, which can be optimized by the PSO. The particles are arbitrarily initialized near the MLP outcomes and converge to the global optimal solution of the whole function, relying on obstacle gradients. PSO is a discrete optimization technique that does not require cost functions to have continuous derivatives. Therefore, it has been widely used to optimize motion planning problems directly on the cost map Lai et al. (2019); Zuo et al. (2019), avoiding the computational burden of converting the map into a continuous form.

The MLP-PSO algorithm solves local planning problems as shown in **Algorithm 1**. Firstly, we execute a forward traversal along the global path for a certain look-ahead distance from the present position of the robot to get the goal point (line 1). This look-ahead distance is determined by the robot's maximum velocity and predictive horizon. Then, acquire the pre-solution by the MLP network (line 2). The MLP result is used for forward simulation to

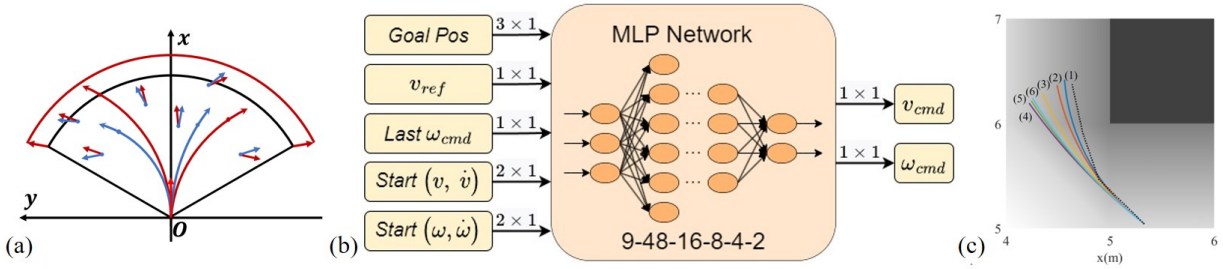

Figure 3. Related figures about MLP-PSO algorithm: (a) The reachable position of the robot on a certain predicted horizon. Each arrow represents a goal ${}^{l}\overline{\boldsymbol{X}}'_{goal}$. (b) The inputs, outputs, and structure of MLP neural network. (c) The re-optimization process of the PSO algorithm for obstacle avoidance. The number represents the optimal trajectory found in the sequence. The shaded region denotes the extension of the obstacle.

acquire predicted trajectories (line 3). If the trajectories are obstacle-free on the local cost map, the MLP result can be output directly to the control module (lines 4-5), and the optimization is finished. If not, the PSO algorithm is used for re-optimization to avoid obstacles (lines 7-23).

The MLP network and PSO algorithm can be decomposed into algebraic operations, without the need for complex code libraries. Therefore, the MLP-PSO algorithm can be deployed on multiple platforms, including some lightweight processors.

Problem Simplification

In this subsection, the MPC local planning problem is transformed into a simpler form.

First, according to the $\boldsymbol{SE(2)}$ transformation of current position $({}^{w}x_{cur}, {}^{w}y_{cur})$ and attitude ${}^{w}R$ in world frame $\boldsymbol{W}$, the reference global path $({}^{w}x_{ref}, {}^{w}y_{ref})$ can be changed to local frame $\boldsymbol{L}$. The position of the robot in the local frame ${}^{l}\boldsymbol{X}'(k)$ is $(0,0,0)^T$, which can be ignored in sampling.

$$\begin{bmatrix} {}^{l}x_{ref} \\ {}^{l}y_{ref} \\ {}^{l}\varphi_{ref} \end{bmatrix} = \begin{bmatrix} \cos{}^{w}\varphi_{cur} & \sin{}^{w}\varphi_{cur} & 0 \\ -\sin{}^{w}\varphi_{cur} & \cos{}^{w}\varphi_{cur} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} {}^{w}x_{ref} - {}^{w}x_{cur} \\ {}^{w}y_{ref} - {}^{w}y_{cur} \\ {}^{w}\varphi_{ref} - {}^{w}\varphi_{cur} \end{bmatrix} \tag{8}$$

Second, the most critical task of the local planner is generating a trajectory along the global path. The look-ahead method is prevalent and efficient. In Paden et al. (2016); Svec et al. (2011); Mora and Tornero (2015), robots can complete the path by tracking a point on the global path at a certain distance in front of them. This method significantly decreases the input dimension of the planner, enables the use of neural networks to plan, and simplifies sampling and training. Moreover, this method reduces constraints on trajectories and increases the optimization's degree of freedom to generate better trajectories while decreasing the influence of poor-quality global paths.

Thus, after two steps, the robot tracks a target point ${}^{l}\boldsymbol{X}'_{goal} = ({}^{l}x_{goal}, {}^{l}y_{goal}, {}^{l}\varphi_{goal})^T$ obtained from the global path to complete the global path in the local frame $\boldsymbol{L}$, as shown by the arrows in Fig. 3(a). The $J_1(\cdot)$ cost function can be simplified as follows.

$$J_1(\cdot) = {}^{l}\overline{\boldsymbol{X}}'^{T}_{goal} H_1 {}^{l}\overline{\boldsymbol{X}}'_{goal} + \Delta\boldsymbol{\xi}'^{T} H_2 \Delta\boldsymbol{\xi}' \tag{9}$$

$${}^{l}\overline{\boldsymbol{X}}'_{goal} = {}^{l}\boldsymbol{X}'_{goal}(k) - {}^{l}\boldsymbol{X}'(N|k) \quad \boldsymbol{X}' = (x, y, \varphi)^T;$$

Algorithm 1 MLP-PSO Algorithm

Input: robot states $\hat{\boldsymbol{X}}(k)$, global path, local cost map
Parameter: kino-dynamic model, cost function parameters
Output: terminal target states $\boldsymbol{\xi}(k)$, local trajectory $\boldsymbol{X}(i|k)$

1: Obtain the goal $\boldsymbol{X}'_{goal}$ from the global path.
2: Put $\hat{\boldsymbol{X}}(k)$ and $\boldsymbol{X}'_{goal}$ into MLP to get result $\boldsymbol{\xi}_{pre}(k)$.
3: Obtain $\boldsymbol{X}(i|k)$ by using $\boldsymbol{\xi}_{pre}(k)$ for forward prediction.
4: **if** predicted trajectory $\boldsymbol{X}(i|k)$ is collision-free **then**
5: $\quad\boldsymbol{\xi}(k) = \boldsymbol{\xi}_{pre}(k)$, **return** $\boldsymbol{\xi}(k)$, $\boldsymbol{X}(i|k)$
6: **else**
7: $\quad\boldsymbol{\xi}^* = \boldsymbol{\xi}_{pre}(k), \ \ c^* = J(\boldsymbol{X}(i|k), \boldsymbol{X}'_{goal}, \boldsymbol{\xi}^*, MAP)$
8: $\quad\Xi \leftarrow$ Particles initialization nearby $\boldsymbol{\xi}_{pre}(k)$.
9: $\quad\boldsymbol{\xi}^*_n \leftarrow \boldsymbol{\xi}_n, \ c^*_n \leftarrow \infty, \ \boldsymbol{\delta}_n \leftarrow rand, \ \forall n \in [1, size(\Xi)]$
10: $\quad$**for** t = 1 to ITERS **do**
11: $\quad\quad$**for** $\boldsymbol{\xi} \in \boldsymbol{\Xi}$ **do**
12: $\quad\quad\quad$Obtain trajectory $\boldsymbol{X}(i|k)$ by putting $\boldsymbol{\xi}_n$ into robots' model for forward prediction.
13: $\quad\quad\quad c_n = J(\boldsymbol{X}(i|k), \boldsymbol{X}'_{goal}, \boldsymbol{\xi}_n, MAP)$
14: $\quad\quad\quad$**if** $c_n < c^*_n$ **then**
15: $\quad\quad\quad\quad c^*_n = c_n, \ \ \boldsymbol{\xi}^*_n = \boldsymbol{\xi}_n$
16: $\quad\quad\quad$**end if**
17: $\quad\quad\quad$**if** $c_n < c^*$ **then**
18: $\quad\quad\quad\quad c^* = c_n, \ \ \boldsymbol{\xi}^* = \boldsymbol{\xi}_n$
19: $\quad\quad\quad$**end if**
20: $\quad\quad\quad\boldsymbol{\delta}_n = \boldsymbol{\delta}_n + k_1 \cdot rand \cdot (\boldsymbol{\xi}^*_n - \boldsymbol{\xi}_n) + k_2 \cdot rand \cdot (\boldsymbol{\xi}^* - \boldsymbol{\xi}_n)$
21: $\quad\quad\quad\boldsymbol{\xi}_n = \boldsymbol{\xi}_n + \boldsymbol{\delta}_n$
22: $\quad\quad$**end for**
23: $\quad$**end for**
24: $\quad$Obtain $\boldsymbol{X}(i|k)$ by using $\boldsymbol{\xi}^*$ for forward prediction.
25: $\quad\boldsymbol{\xi}(k) = \boldsymbol{\xi}^*$, **return** $\boldsymbol{\xi}(k)$, $\boldsymbol{X}(i|k)$
26: **end if**

MLP Neural Network

The global optimal solution for optimization problems is a function of the inputs and parameters. The existence of variable constraints, nonlinear links, and generation rules of motion primitives makes the analytical function of optimization problems either non-existent or challenging to find and express. The neural networks can directly learn and adapt the result function from the result dataset, imitating the solution procedure. An MLP network was designed in this study.

Fig. 3(b) depicts the structure of the MLP neural network, which has 9 inputs and 2 outputs. The inputs are: goal position $({}^{l}x_{goal}, {}^{l}y_{goal}, {}^{l}\varphi_{goal})$ (*Goal Pos*, 3×1), reference velocity by setting (v_{ref}, 1×1), last terminal angular velocity (*Last* ω_{ter}, 1×1), start velocity and acceleration (*Start* $(v, \dot{v})$, 2×1), start angular velocity and angular acceleration (*Start* $(\omega, \dot{\omega})$, 2×1). Start $(v, \dot{v})$ and $(\omega, \dot{\omega})$ are current states of robot. The outputs are the motion primitive composed of terminal velocity (v_{ter}, 1×1) and terminal angular velocity (ω_{ter}, 1×1). After determining the inputs and outputs of the neural network, hidden layers are added to form a complete structure. Based on the common expansion-contraction rule, after multiple tests, the established MLP neural network has 6 layers, with a structure of 9-48-16-8-4-2. The activation function of the hidden layer is $ReLU$, while $Sigmoid$ is used for the output layer.

To satisfy the calculation of the MLP network, the inputs and outputs are normalized to $[0, 1]$ by the boundaries, which are determined by different methods. The boundary of the robot's states (such as v, ω) can be obtained through theoretical analysis based on the capabilities of actuators such as motors. The pre-experiments find the coefficients of the cost function and the boundary of *Goal Pos*. As shown in Fig. 3(a), for a certain predicted horizon, the reachable position of the robot should be a sector. The MAX look-ahead distance is determined by the MAX v and a of the robot, whereas the MAX angle the robot can rotate within a given time is limited by the MAX ω.

Compared to traditional look-up methods, the MLP neural network's result is continuous on state space, producing more accurate results by fitting for non-sampled points. And the MLP method only requires the storage of the network's parameters (hundreds of floating-point numbers), without the need to store complex search trees or look-up tables, thereby saving a significant amount of space. The computation cost of forward propagation is also superior to that of searching in complex trees.

Sampling and Training

The sampling space is a 9-dimensional space composed of neural network inputs. The sampling only needs to be carried out within the boundary, and points outside the sampling place can be normalized to the boundary. The sampling can be divided into two steps: first, uniform sampling to assure coverage, followed by adding white noise with standard deviation as sampling interval to points to improve randomness. Each sampling point can be converted into an MPC problem and solved by mature solvers such as IPOPT Wächter and Biegler (2006) to get the result.

The dataset can be divided into a training set and a testing set with a ratio of 7:3. The MLP neural network is trained with a weighted mean-square error (WMSE) loss on velocity and angular velocity based on backpropagation and batch training algorithm. The batch size was set to 200 and the training cycle was 1000. The loss-function of training is:

$$Loss = w_v \cdot ||v_d - v_n||^2 + w_\omega \cdot ||\omega_d - \omega_n||^2 \qquad (10)$$

Where v_d and ω_d are the command data in the training set, while v_n and ω_n are the outputs of the network. To maintain the stability of the robot's orientation, we pay more attention to the ω error by setting the weigh w_v and w_ω as 1, 2.

The MLP network employs the dataset as a teacher to imitate the solver. The trained MLP network can be regarded as a solver for a specific optimization problem.

PSO Algorithm

PSO is an intelligent search technique that optimizes the cost function by utilizing the motion of particles. The discrete cost function can be optimized directly due to the direction of particles, which only depends on the values of the previous and subsequent iterations.

The particle swarm is initialized randomly near the result of the MLP network, which is also the initial global optimal result (lines 7-9). Then, the optimal solution is searched through iteration. In each iteration, for each particle, the BVP problem is solved to get the full input sequence of the robot (line 12). Then, utilize the prediction model and cost function $J(\cdot)$ to get the cost (line 13). Through the comparison of costs, each particle's and global optimal result are updated (lines 14-19). The optimization is stopped when the cost falls below the set value or hits the iteration limit. The optimal result is output to the control module for execution (line 25).

The process of a re-optimization is shown in Fig. 3(c). It can be seen that as the optimization goes on, the robot's trajectory moves away from obstacles, getting the optimal result considering obstacles. Because the MLP network produces approximations, the PSO algorithm can get the optimal result much more quickly than optimizing directly.

Implementation and Experimental Robots

In this chapter, we first outline the deployment steps of the MLP-PSO method onto the robot, followed by the presentation of the car-like robot and spherical robot used for testing.

Implementation

For any robot, the deployment of the MLPPSO method involves the following steps:

- Establishing the robot model and selecting appropriate motion primitives based on the model.
- Formulating the model predictive trajectory planning problem and simplifying it by motion primitives.
- Determining system state boundaries and sampling rules to create multiple planning problems, solving them by mature solvers to obtain planning results and form the complete dataset.
- Partitioning the dataset into training and testing sets for training the MLP neural network.
- Deploying the trained MLP neural network and PSO algorithm on the robot platform.

By forming the new dataset, and adjusting the MLP network's inputs and outputs, adaptation to more robots can be achieved, such as quadcopter in $\boldsymbol{SE(3)}$ space.

Car-like robot

The car-like robot is one of the most common unmanned ground vehicles (UGV), which achieves free motion on the X-Y plane by changing the steering angle of the front wheels, as shown in Fig. 4(a). To ensure the smoothness of

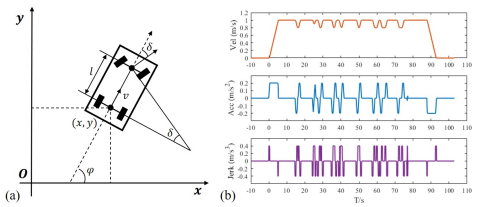

Figure 4. (a) The motion diagram of the car-like robot. (b) The v, α, and j curves of a JLT trajectory.

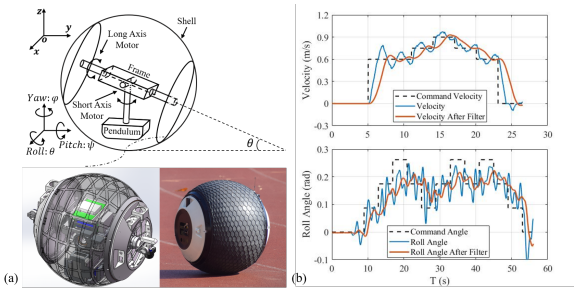

Figure 5. (a) The motion diagram, design drawing, and physical drawing of the spherical robot. (b) The control effect of attitude controller.

the trajectory, Jerk limited trajectory (JLT) Haschke et al. (2008) is introduced as the motion primitive, and the terminal constraints of its BVP problem are the terminal velocity v_{ter} and angular speed ω_{ter}. Then, the kino-dynamic model of the robot can be expressed as Lai et al. (2019):

$$\begin{aligned} &\dot{x} = v\cos\varphi, \quad \dot{y} = v\sin\varphi, \quad \dot{\varphi} = \omega = v \cdot \sin\delta/l \\ &\dot{v} = \alpha, \quad \dot{\alpha} = j, \quad \dot{\omega} = \beta, \quad \dot{\beta} = j^{\omega} \\ &v \in [v_{min}, v_{max}],\ \alpha \in [\alpha_{min}, \alpha_{max}],\ j \in \{0, j_{min}, j_{max}\} \\ &\omega \in [\omega_{min}, \omega_{max}],\ \beta \in [\beta_{min}, \beta_{max}],\ j^{\omega} \in \{0, j^{\omega}_{min}, j^{\omega}_{max}\} \end{aligned} \tag{11}$$

The state vector of the robot is $[x, y, \varphi, v, \alpha, j, \omega, \beta, j^{\omega}]$. And the v, α, and j curves of a JLT trajectory are shown in Fig. 4(b). The values of the jerk are $\{max, min, 0\}$. After integration, the α is continuous, and the v is continuous and smooth. When both v and w satisfy the constraints of JLT, the robot's trajectories are very smooth curves. After incorporating JLT into the MPC local planner, the PSO algorithm is utilized to solve optimization problems, similar to the application in Lai et al. (2019).

Spherical Robot

The spherical robot is an innovative type of special operation robot with large application potential in field exploration, security patrol, planet exploration, etc Liu et al. (2008). As shown in Fig. 5(a), our laboratory has developed a spherical robot driven by a 2-DOF pendulum. By adjusting the output torque of two motors to adjust the position of the pendulum, the spherical robot can achieve motion on the X-Y plane.

The characteristics of the spherical robot Vrunda et al. (2010); Wang et al. (2021), including strong nonlinearity, under-actuation, non-holonomic, and a high time latency, present challenges for the planning control system. It has been determined that the attitude controllers of velocity v and roll angle θ are necessary. As shown in Fig. 5(b), even with the controllers, the attitude of the robot still has some fluctuations. Therefore, the robot's dynamic model must be incorporated into the local motion planner. Due to the complexity of the controller and robot model, it is challenging to develop a theoretical model, which can't be optimized in real-time either. An ARX model is used to identify a simplified linear dynamic model of the cycle composed of the controller and robot, which is the control object of the motion planner. The complete kino-dynamic model is as follows:

$$\begin{aligned} &\dot{x} = v\cos\varphi, \quad \dot{y} = v\sin\varphi, \quad \dot{\varphi} = \omega = v \cdot \tan\theta/r \\ &\dot{\mathbf{x}} = \boldsymbol{A} \cdot \mathbf{x} + \boldsymbol{B} \cdot \mathbf{u}, \mathbf{x} = [v, \dot{v}, \theta, \dot{\theta}]^T, \mathbf{u} = [v_{cmd}, \theta_{cmd}]^T \\ &\mathbf{x} \in [\mathbf{x}_{min}, \mathbf{x}_{max}],\ \mathbf{u} \in [\mathbf{u}_{min}, \mathbf{u}_{max}] \end{aligned} \tag{12}$$

Where $\boldsymbol{A}$ and $\boldsymbol{B}$ are given by the system identification method. θ is the roll angle of the spherical robot. The whole state vector is $[x, y, \varphi, \omega, v, \dot{v}, \theta, \dot{\theta}]$. The motion primitives are also the terminal velocity v_{ter} and angle velocity ω_{ter}. In which, the ω_{ter} can be converted to θ_{ter} through the formula $\omega = v \cdot \tan\theta/r$. The MPC local planning problem of the spherical robot is solved by IPOPT Wächter and Biegler (2006). Obstacle avoidance is achieved by adding hard constraints to states.

Results

In this section, the details of the MLP training are presented first. Next, the solution accuracy of the MLP-PSO algorithm was evaluated using arbitrarily generated data. After deploying the algorithm to the car-like and spherical robot, the effect and time consumption of the MLP-PSO and traditional solvers (PSO for car and IPOPT for spherical) are compared experimentally. Subsequently, the spherical robot is used to evaluate the obstacle avoidance ability of the MLP-PSO algorithm.

Sampling, training, and testing all run in a single thread on a mini PC with an Intel i7-8559U CPU (2.70 GHz, quad-core 64-bit) and 32G RAM.

The MPC local planner runs at a frequency of 10 Hz on robots, which is a commonly used frequency of the local planner Lai et al. (2019); Fnadi et al. (2019); Zhu et al. (2020). The frequency of the global planner is 1 Hz, so there might be multiple global paths in the resulting figure.

MLP Training Results

The predictive horizon of the MPC local planner is set to 2s. After the offline sampling and generation of the data set based on the pre-experiment parameters, two pre-solving MLP networks are trained for selected motion primitives of the car-like and spherical robot. Tab. 1 displays the data set size, time consumption, and training error.

After 1000 cycles of training, the root mean square error (RMSE) of velocity v and angular velocity ω on the test set are less than 0.005 and 0.015. In comparison to the robot's v range $[0, 1]$m/s and ω range $[-0.6, 0.6]$rad/s, the error ratios are 0.5% and 1.25%, which are negligible. While the MAX error can reach 2% and 8% on v and ω. Because the MAX error occurs by accident, it is acceptable in applications. The total time consumption for sampling and training is about 6 and 2.5 hours respectively, not high and meeting application requirements. As the car-like robot shows, when the model

Table 1. The details of MLP Training

Robot	Samples Number			Time Consumption(s)			Error on $v(m/s)$		Error on $\omega(rad/s)$	
	Training Set	Test Set	All	Sampling	Training	All	RMSE	MAX	RMSE	MAX
Car-like Robot	329275	141118	470393	14126.94	7537.43	21664.37	0.00401	0.01907	0.01448	0.09724
Spherical Robot	167991	71996	239987	4718.75	3811.49	8530.24	0.00145	0.00570	0.01076	0.05676

Table 2. The Solution Accuracy of MLP-PSO Algorithm

Robot	Scenario	Error on $v(m/s)$		Error on $\omega(rad/s)$	
		Average	MAX	Average	MAX
Car-like	Clear	0.00538	0.01875	0.01140	0.08426
	Obstacle	0.00457	0.00618	0.00330	0.03571
Spherical	Clear	0.00150	0.00452	0.00891	0.02648
	Obstacle	0.00013	0.00031	0.00502	0.01400

and motion primitives are more complex, more detailed sampling is required, and training time also increases.

Solution Accuracy

In this section, 1000 random samples are generated in environments with and without obstacles to compare the MLP-PSO algorithm and traditional solvers (PSO or IPOPT, as mentioned above). The results are shown in Tab. 2. The results given by traditional solvers are regarded as **ground truth**.

The obstacle-free scenario only requires the application of the MLP network, whereas the obstacle environment necessitates re-optimization with the PSO algorithm.

No matter what the robot is, the average errors of velocity v and angular velocity ω are less than 1% of the value range. The v error is smaller, 0.5% for the UGV and 0.1% for the spherical robot. Considering rare extreme situations, the MAX v error is 2% of the value range, while for ω is 7%, which is acceptable as MAX error occurs by chance. The interference of obstacles can be effectively resolved by PSO re-optimization, which can also decrease fitting errors of MLP, reducing the average errors. It can be seen that the complex motion primitives of car-like robots present greater challenges to the MLP-PSO, resulting in bigger errors.

Overall, the MLP-PSO algorithm satisfies requirements and can be implemented on robots for further testing.

Car-like robot Experiment

The car-like robot completes multiple curved paths in a complex corridor environment by **the simulator**.

The reference velocity v_{ref} varies. The robot tries to complete the straight path at 1m/s and achieve small radius non-skid turns at 0.8m/s or even 0.6m/s. Five experiments were conducted to eradicate randomness. The statistics which evaluate path completion and solve time are presented in Tab. 3 as the mean $\pm$ standard deviation. Fig. 6(a) depicts the robot's driving result, velocity, and angular velocity curve.

It can be seen from the results that the MLP-PSO algorithm can drive the robot along the global path with the same performance as the traditional solver, with no significant differences in statistics (errors less than 5%) and curves. The curves demonstrate that the robot's v and ω are continuous and smooth, satisfying the JLT constraints. As shown, the robot's trajectory tends to move away from obstacles, proving that PSO re-optimization has achieved the designed function.

As for time consumption, the MLP-PSO algorithm is clearly superior. In total implementation time, the MLP-PSO algorithm is 93.2% less than traditional methods. The obstacle-free scenario only requires the MLP network, with an average solving time of 0.142ms, which can be negligible. In obstacle scenarios, re-optimization is required, with an average total optimization time of 2.594ms, which is also 87.4% faster than traditional methods. In extreme circumstances, the improvement is more obvious. Traditional methods may require around 95ms maximum and can only plan at 10Hz. While the MLP-PSO algorithm requires less than 8ms, enabling the local planner's operating frequency to be increased to 20Hz, 50Hz, or even higher. This also allows for the use of more complex motion primitives to meet higher control demands.

From the trajectory, it can be observed that the complexity of the robot traversed areas varies, with both wide and narrow areas. The small solving time variance result indicates that complex environments with multiple obstacles do not significantly increase the solving time. The soft constraint obstacle avoidance method makes multiple obstacles generate more optimization gradients on the map rather than imposing complex constraints, which has minimal impact on the solving complexity.

Spherical Robot Experiment

The spherical robot, shown in Fig. 5, was utilized to conduct **real physical experiments**, which was designed and developed by our laboratory.

The task is an 8-figure multi-point patrol with eleven waypoints. The v_{ref} setting is the same as the car-like robot. Since the turns of the spherical robot are accomplished by adjusting roll angle θ, some statistics and curves are changed to θ. The results are illustrated in Tab. 3 and Fig. 6(b).

The results indicate that the IPOPT solver and MLP-PSO algorithm can both solve the local planning problem, guiding the spherical robot to complete the patrol task effectively and in a stable state. The is no significant difference between the two methods' statistics and result trajectories. Under the commands given by the MLP-PSO algorithm, the non-holonomic, non-linear, under-actuated, and critically stable spherical robot can achieve smooth acceleration and

Table 3. The Results of Experiments

Robot	Solver	Execution Time (s)	Running Distance (m)	Average ω (rad/s)	Range of ω or θ (rad/s or $^\circ$)
Car-like Robot	PSO	83.54 ± 2.54	94.25 ± 3.68	0.171 ± 0.011	[-0.528, 0.571]
	MLP-PSO	81.64 ± 1.36	90.20 ± 1.32	0.166 ± 0.004	[-0.518, 0.572]
Spherical Robot	IPOPT	56.91 ± 0.50	65.84 ± 1.10	0.185 ± 0.006	[-17.86, 18.35]
	MLP-PSO	55.71 ± 0.23	64.63 ± 0.62	0.180 ± 0.002	[-16.19, 20.29]

Robot	Solver	Average Solving Time (ms)			Range of Solving Time (ms)
		All	MLP	MLP-PSO	
Car-like Robot	PSO	20.51 ± 6.961	/	/	[8.394, 94.058]
	MLP-PSO	1.388 ± 1.370	0.142 ± 0.052	2.594 ± 0.920	[0.091, 7.763]
Spherical Robot	IPOPT	15.89 ± 3.602	/	/	[10.65, 82.91]
	MLP-PSO	0.272 ± 0.673	0.107 ± 0.037	2.827 ± 0.703	[0.063, 4.915]

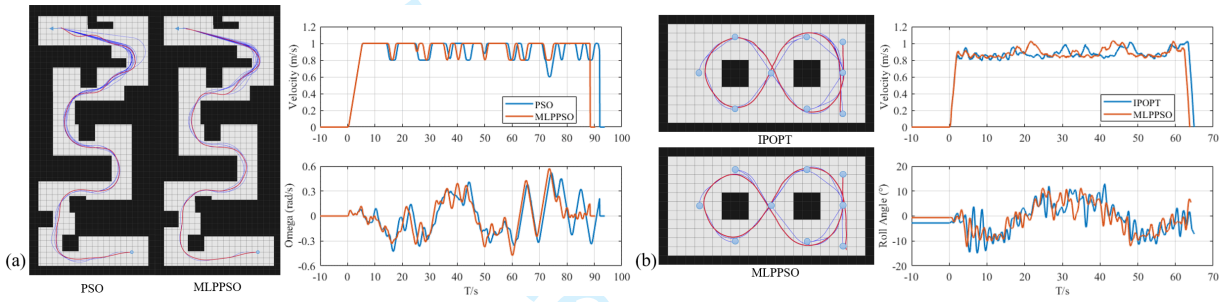

Figure 6. (a) The diving result (blue line: global paths; red line: driving path), velocity curve, and angular velocity curve of the car-like robot with two different solvers. (b) The diving result (blue line: global paths; red line: driving path), velocity curve, and roll angle curve of the spherical robot with two different solvers.

deceleration based on actual conditions in order to increase efficiency. The roll angle can also be adjusted on command to achieve turning without violent oscillation or divergence.

The average solving time for the MLP-PSO algorithm is 0.107ms and 2.827ms in obstacle-free and obstacle environments, which is 99.3% and 82.2% less than traditional methods. The benefits are readily apparent. Because there are few obstacles, the overall average solving time decreases substantially (around 98.3%). Furthermore, the maximum solving time is less than 5ms (a decrease of 94.1%), which is a significant improvement over the IPOPT solver (64.03ms).

In conclusion, experiments on car-like and spherical robots demonstrate that the MLP-PSO algorithm can effectively solve the model-predictive trajectory planning problem to guide the robot running along the global path with the same performance as the traditional solver, reduce the solving time by more than 82% even in complex environments. The results prove that the MLP-PSO algorithm can adapt to various motion primitives and be implemented on real robots.

Obstacle Avoidance Experiment

In the experiments, two kinds of robots successfully avoided obstacles, especially the car-like robot running in the narrow corridor. From Fig. 6(a), it is evident that the global path (blue line) did not account for the width of the robot, which may guide the robot to turn very near to the obstacle and cause collisions. The PSO algorithm re-optimized the trajectory to guide the robot away from the obstacle when it reached the turning point. After moving away from obstacles, the robot returned to the global path quickly. In conclusion, the MLP-PSO algorithm can achieve obstacle avoidance and the soft constraint scheme is effective.

Conclusion

This paper proposes a two-stage solving method called MLP-PSO for model predictive trajectory planners, addressing the challenges posed by complex models and long predicted horizons. At first, to quickly solve the standard MPC planning problem, an MLP neural network was trained on an offline dataset to imitate the solver without considering obstacles. Then, the obstacle avoidance task is accomplished by the PSO algorithm employed for re-optimization. Experiments have demonstrated that the solving quality of the MLP-PSO algorithm is identical to that of traditional solvers. Guided by the commands from the MLP-PSO algorithm, robots can effectively complete global paths while satisfying the predetermined requirements of the local planner. The MLP-PSO algorithm has a significant advantage in solving complexity and time consumption, reducing the average solving time by more than 90%. The maximum solving time is less than 10ms, making it possible to improve moving performance by increasing the

planning frequency to 20Hz and higher or adopting higher-quality complex motion primitives. Moreover, the MLP-PSO algorithm is suited for various robots and motion primitives. As for application, the MLP-PSO algorithm does not rely on code libraries, allowing MPC local planners to be deployed on low-performance processors and embedded platforms.

MLP-PSO offers a potential optimization method for complex MPC planning problems that cannot be solved in real-time on robot platforms. We will try this application in the future. In addition, the application field of the MLP-PSO algorithm can be broadened by increasing the prediction horizon, adding robot and motion primitive types, and integrating dynamic models and constraints.

References

Andersson JAE, Gillis J, Horn G, Rawlings JB and Diehl M (2019) CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation* 11(1): 1–36. DOI:10.1007/s12532-018-0139-4.

Eiras F, Hawasly M, Albrecht SV and Ramamoorthy S (2021) A two-stage optimization-based motion planner for safe urban driving. *IEEE Transactions on Robotics* 38(2): 822–834.

Ferguson D, Howard TM and Likhachev M (2008) Motion planning in urban environments: Part i. In: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Fnadi M, Plumet F and Benamar F (2019) Model predictive control based dynamic path tracking of a four-wheel steering mobile robot. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Frazzoli E, Dahleh MA and Feron E (2005) Maneuver-based motion planning for nonlinear systems with symmetries. *IEEE transactions on robotics* 21(6): 1077–1091.

Haschke R, Weitnauer E and Ritter H (2008) On-line planning of time-optimal, jerk-limited trajectories. In: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. pp. 3248–3253. DOI:10.1109/IROS.2008.4650924.

Howard TM, Pivtoraiko M, Knepper RA and Kelly A (2014) Model-predictive motion planning: Several key developments for autonomous mobile robots. *IEEE Robotics and Automation Magazine* 21(1): 64–73.

Ji J, Zhou X, Xu C and Gao F (2020) Cmpcc: Corridor-based model predictive contouring control for aggressive drone flight .

Lai S, Lan M and Chen BM (2019) Model predictive local motion planning with boundary state constrained primitives. *IEEE Robotics and Automation Letters* 4(4): 3577–3584.

Lin J, Wang L, Gao F, Shen S and Zhang F (2019) Flying through a narrow gap using neural network: an end-to-end planning and control approach. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 3526–3533.

Lindqvist B, Mansouri SS, Agha-mohammadi Aa and Nikolakopoulos G (2020) Nonlinear mpc for collision avoidance and control of uavs with dynamic obstacles. *IEEE Robotics and Automation Letters* 5(4): 6001–6008. DOI:10.1109/LRA.2020.3010730.

Liniger A, Domahidi A and Morari M (2015) Optimization-based autonomous racing of 1: 43 scale rc cars. *Optimal Control Applications and Methods* 36(5): 628–647.

Liu D, Sun H and Jia Q (2008) A family of spherical mobile robot: Driving ahead motion control by feedback linearization. In: *Systems and Control in Aerospace and Astronautics, 2008. ISSCAA 2008. 2nd International Symposium on*.

Lopez BT and How JP (2017) Aggressive 3-d collision avoidance for high-speed navigation. In: *ICRA*. pp. 5759–5765.

Ly AO and Akhloufi M (2020) Learning to drive by imitation: An overview of deep behavior cloning methods. *IEEE Transactions on Intelligent Vehicles* 6(2): 195–209.

Mora MC and Tornero J (2015) Predictive and multirate sensor-based planning under uncertainty. *IEEE Transactions on Intelligent Transportation Systems* 16(3): 1493–1504. DOI: 10.1109/TITS.2014.2366974.

Mueller MW, Hehn M and D'Andrea R (2015) A computationally efficient motion primitive for quadrocopter trajectory generation. *IEEE transactions on robotics* 31(6): 1294–1310.

Paden B, Cap M, Yong SZ, Yershov D and Frazzoli E (2016) A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles* 1(1): 33–55.

Paranjape AA, Meier KC, Shi X, Chung SJ and Hutchinson S (2015) Motion primitives and 3d path planning for fast flight through a forest. *The International Journal of Robotics Research* 34(3): 357–377.

Rsmann C, Hoffmann F and Bertram T (2017) Kinodynamic trajectory optimization and control for car-like robots. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Saleh K, Attia M, Hossny M, Hanoun S, Salaken S and Nahavandi S (2018) Local motion planning for ground mobile robots via deep imitation learning. In: *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, pp. 4077–4082.

Schwesinger U, Rufli M, Furgale P and Siegwart R (2013) A sampling-based partial motion planning framework for system-compliant navigation along a reference path. In: *2013 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, pp. 391–396.

Svec P, Schwartz M, Thakur A and Gupta SK (2011) Trajectory planning with look-ahead for unmanned sea surface vehicles to handle environmental disturbances. In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. pp. 1154–1159. DOI:10.1109/IROS.2011.6095021.

Ulrich I and Borenstein J (2002) Vfh+: reliable obstacle avoidance for fast mobile robots. In: *IEEE International Conference on Robotics and Automation*.

Verschueren R, Frison G, Kouzoupis D, Frey J, van Duijkeren N, Zanelli A, Novoselnik B, Albin T, Quirynen R and Diehl M (2021) acados – a modular open-source framework for fast embedded optimal control. *Mathematical Programming Computation* DOI:10.1007/s12532-021-00208-8.

Vrunda, A, Joshi, , , Ravi, N, Banavar, , , Rohit and Hippalgaonkar (2010) Design and analysis of a spherical mobile robot. *Mechanism and Machine Theory* 45(2): 130–136.

Wächter A and Biegler LT (2006) On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming* 106: 25–57.

Wang Y, Guan X, Hu T, Zhang Z, Y W, Z W, Y L and G L (2021) Fuzzy pid controller based on yaw angle prediction of a spherical robot. In: *Proceedings. 2000 IEEE/RSJ International*

Conference on Intelligent Robots and Systems (IROS 2021) (Cat. No.00CH37113), volume 1.

Williams G, Drews P, Goldfain B, Rehg JM and Theodorou EA (2016) Aggressive driving with model predictive path integral control. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. pp. 1433–1440. DOI:10.1109/ICRA.2016.7487277.

Yang K and Sukkarieh S (2010) An analytical continuous-curvature path-smoothing algorithm. *IEEE Transactions on Robotics* 26(3): 561–568. DOI:10.1109/TRO.2010.2042990.

Yang S, He B, Wang Z, Xu C and Gao F (2021) Whole-body real-time motion planning for multicopters. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 9197–9203.

Yang X, Sreenath K and Michael N (2017) A framework for efficient teleoperation via online adaptation. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 5948–5953.

Zhang T, Kahn G, Levine S and Abbeel P (2016) Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search. In: *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE, pp. 528–535.

Zhu D, Gan W, Hu Z, Yang L, Shi X and Chen Y (2020) A hybrid control strategy of 7000 m-human occupied vehicle tracking control. *IEEE Transactions on Intelligent Vehicles* 5(2): 251–264. DOI:10.1109/TIV.2019.2955901.

Zuo Z, Yang X, Zhang Z and Wang Y (2019) Lane-associated mpc path planning for autonomous vehicles. In: *2019 Chinese Control Conference (CCC)*. pp. 6627–6632. DOI:10.23919/ChiCC.2019.8866609.