Cost Function Design in MPPI

January 31, 2024

1 Cost Function Design in MPPI

In the realm of dynamic obstacle avoidance for robotic arms, the design of the cost function plays a pivotal role in the MPPI algorithm. The primary objective of our cost function is to guide the robot, ensuring compliance with dynamic constraints, avoiding collisions with the environment, and efficiently reaching the target point. Different strategies, such as Greedy and Sensitive policies, express their uniqueness through the tailored design of their respective cost functions. The efficacy of the cost function design directly influences MPPI's ability to explore the state space, thereby impacting the planning efficiency of the algorithm.

The cost function comprises several sub-cost functions, including:

- Dynamic constraint cost function (considering joint acceleration, velocity, and position limits).
- Target distance cost function (addressing distance to the target in joint space and Cartesian space).
- Self-collision cost function.
- Environment collision cost function (involving cost calculations between key points of the robotic arm and SDF/Gradient Map).

This paper introduces a series of optimization tricks to enhance the MPPI cost function. Inspired by the application of the SDF in 2D navigation, we incorporate considerations for the robot's current velocity and SDF gradient direction, adapting it to the dynamic obstacle avoidance tasks of robotic arms. We employ the Sparse Reward mechanism to mitigate significant errors in the robot's arrival at the target point. Additionally, we deviate from conventional approaches that use manipulability indices to optimize the manipulator configuration, opting instead for a multiprocessing technique which leveraging Track-IK to calculate the target joint position based on the current joint state.

2 Enhanced SDF for manipulator

2.1 The robot body representation

When using SDF for collision detection of robotic arms, the robot body is often approximated as a set of overlapping spheres. Specifically, the spheres can be simplified as a set of key points on the robotic arm and their corresponding radii r.

The spheres are grouped according to robot links. Given the local coordinate P of a sphere under $link_i$ and the joint state q, the spheres' position in the workspace can be obtained through a simple coordinate transformation using the kinematic model:

$$P^{'} = T(base\_to\_link_i) * P$$

In this way, environment collision detection for robot arm is simplified to a three-dimensional collision query problem of sets of spheres: querying whether the SDF value corresponding to any key point $P^{'}$ is less than a threshold $r + \epsilon$.

Specifically, this paper derives the mapping relationship between joint velocity q_{vel} and the velocity of key points on the robotic arm. Assuming MPPI generates M batches of joint trajectories $q_M(t = 1, ..., H)$, the corresponding trajectories of key points in workspace $P^{'}_M(t = 1, ..., H)$ can then be calculated using Equation 1. The velocity of the key points $\mathrm{Vel}^{'}_M(t = 1, ..., H)$ can then be obtained via simple differentiation.

$$\begin{bmatrix} -1 & 1 & 0 & \dots & 0 & 0 & 0 \\ 0 & -1 & 1 & \dots & 0 & 0 & 0 \\ 0 & 0 & -1 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & -1 & 1 & 0 \\ 0 & 0 & 0 & \dots & 0 & -1 & 1 \\ 0 & 0 & 0 & \dots & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} Pos(1) \\ Pos(2) \\ Pos(3) \\ \vdots \\ Pos(H-2) \\ Pos(H-1) \\ Pos(H) \end{bmatrix} = \begin{bmatrix} Vel(1) \\ Vel(2) \\ Vel(3) \\ \vdots \\ Vel(H-2) \\ Vel(H-1) \\ Vel(H) \end{bmatrix}$$

2.2 SDF Map Creation

We begin with a boolean voxel representation of the environment, known as voxel grid, obtained directly from the camera point cloud . A signed Euclidean Distance Transform (EDT) is then applied to the voxel grid to generate a SDF map, which represents, for each voxel point x, the minimum distance $d(x)$ to the nearest obstacle boundary, The distance values are negative for points inside obstacles, zero at obstacle boundaries, and positive outside of obstacles.

Inspired by the application of the inflation layer in car navigation, we adapt a similar concept to robotic arms. Building upon the SDF Map, we design a corresponding workspace potential function, also referred to as the SDF Potential. This allows for a finer control of the distance between the robot and obstacles, providing increased safety margins in proximity to obstacles.

$$\text{SDFPotential}(x) = \begin{cases} 1, & if\ \text{d(x)}\sigma \\ \exp(-\beta(x-\sigma)), & if\ \sigma \leq \text{d(x)} \leq \tau \\ 0, & otherwise \end{cases}$$

where $\sigma = r + \epsilon$

The SDF gradient indicates the direction of steepest change at each point x in the SDF map. We compute the gradient in the $i \in x, y, z$ directions at each point px using central finite differencing:

$$\text{SDFGradient}(x) = \frac{\partial d}{\partial x_i}(x) \approx \frac{d(x + \delta \mathbf{e}_i) - d(x - \delta \mathbf{e}_i)}{2\delta}$$

where $\mathbf{e_i}$ is the corresponding basis vector.

2.3 GVOM SDF Cost

Previous works that incorporated SDF into the MPPI cost function exhibited issues where the robot could rapidly pass through $\text{Coll}_p(x)$ or get stuck close to obstacles $\text{Coll}_{pv}(x)$.

To address this, our method $\text{Coll}_{ppv\theta}(x)$ introduces dynamic modulation of the SDF cost based on the robot's current velocity and the SDF gradient. Let the gradient represent the direction of steepest descent in SDF map. If the robot's velocity and gradient directions are aligned, indicating the robot is moving away from obstacles, the cost is lowered to encourage that motion and avoid stopping near obstacles.

Whereas if the velocity and gradient directions are misaligned, indicating movement towards obstacles, the cost is increased to discourage that behavior and avoid rapid passing through obstacles just to minimize accumulated cost.

In the case of a 2D mass point, x denotes the specific position of the robot; for Franka, x indicates the point with the minimum SDF value among the key points for each link i:

$$\begin{aligned} \text{Coll}_p(x) &= \sum\nolimits_{\text{each\_link}} \text{SDFPotential}(x) \\ \text{Coll}_{pv}(x) &= \sum\nolimits_{\text{each\_link}} \text{SDFPotential}(x) \cdot \|\text{RobotVel}(x)\| \\ \text{Coll}_{ppv\theta}(x) &= \sum\nolimits_{\text{each\_link}} w_1 \cdot \text{SDFPotential}(x) \\ &\quad + w_2 \cdot \text{SDFPotential}(x) \cdot \|\text{RobotVel}(x)\| \cdot (1 - \alpha \cdot \cos(\theta)) \end{aligned}$$

where

$$\begin{aligned} x^* &= \text{argmin}_{x \in \text{key points(link}_i\text{)}} \text{SDFMap}(x) \\ \theta &= \cos^{-1}(\text{SDFGradient}(x), \text{RobotVel}(x)) \\ \alpha &\in [0, 1] \end{aligned}$$

Algorithm 1 GVOM SDF Cost

Input:

Voxel Grid obtained from point cloud

Robot's key points for each link $pts(link_i)$

Current Robot Joint State q_t

Precompute Maps

1: SDF Map $d(x) \leftarrow EDT(\text{Voxel Grid})$

2: SDF Potential Map $\leftarrow \exp(-\beta(d(x) - r))$

3: SDF Gradient Map $\leftarrow FD(d(x))$

Environment Collision Cost

4: $pts \leftarrow FK(q_t), \text{Vel}(pts) \leftarrow FD(pts)$

5: $pt(link_i), \text{Vel}(pt(link_i)) \leftarrow \text{argmin}(\text{SDFMap}(pts(link_i)))$

6: Check SDF Potential/Gradient Map(pt)

7: Compute collision cost using Equation $\text{Coll}_{ppv\theta}(pt)$

2.4 the pseudo code of GVOM SDF Cost

This section introduces the GVOM SDF Cost algorithm.

Lines 1-3 involve the precomputation of Maps. EDT on the Voxel Grid yields the SDF Map, followed by computing the SDF Potential Map for nuanced distance control. The Gradient Map is derived through finite differencing based on the SDF Map.

Lines 4-7 compute environment collision cost. Line 4 involves updating key point positions using the forward kinematic model. After transformation to the world frame, velocities are obtained via finite differencing. Line 5 identifies the key point with the minimum SDF value for each link from the SDF Map. Line 6 retrieves the values of SDF Potential and Gradient Maps for the selected points in each link. Line 7 computes the collision cost $\text{Coll}_{ppv\theta}(pt)$.