3D Voxel Grid Based Path Planning for Robotic Manipulators using Matrix Multiplication Technique

Alternatives:
- Efficient distance calculation / Efficient repulsive field calculation technique
- Matrix Multiplication-Driven Repulsive Fields for 3D Voxel-Based Robotic Manipulator Path Planning
- Robotic Manipulator Path Planning Optimization Using Matrix-Derived Repulsive Fields Based on 3D Voxel Grid

Jakob Baumgartner, Gregor Klančar

November 22, 2023

for keywords use the SEO algorithm to find which terms are popular

Abstract

1 Introduction

2 Background

1-2 PAGES

- PRESENT DISTANCE CALCULATION FOR MANIPULATORS (sensors, lidar, ir, bounding boxes)
- PRESENT PATH PLANNING METHODS FOR MANIPULATOR (optimization, sampling, biological, learning)
- similar to Distance Transform (a kind of inverted distance transform)
- method was inspired by Khatib APF (however, it evolved into a different method)
- different existing APF manipulator implementation articles
- VFH

3 Methodology

3 PAGES

3.1 Optimization Algorithm

We use inverse kinematics algorithm to calculate joint velocities based on the primary goal, that is EE position and secondary task of obstacle avoidance. (INSERT: a sentence or two about ik algorithm, also a reference) We also use additional secondary tasks, such as mid-joints position and more importantly manipulability task, which allows us to avoid some singularity positions of the manipulator arm and some local minima, which are a consequence of the local optimization approach used.

We start our algorithm in a start joint configuration, that can be previous goal position. We run optimization loop, that runs step by step from start to goal configuration, while avoiding obstacles / applieing constraints and limits. Each step we calculate the joint velocities and integrate them into new joint positions. Each of so gotten positions is one of the points on our joints trajectory. The priamry IK task is to take the manipulator EE to the desired goal position and orientation, we describe how to calculate cartesian velocities in the chapter ref:Attractive Velocity, which are than calculated into joint velocities. The size of this primary velociti is constant for the entire part of the optimization and is reduced only when approaching the goal.

To avoid obstacles we calculate the avoidance velocities as descrubed in chapter ref:Repulsive Velocity. As one of the biggest problems of the APF Khatib method is local minima points, where the robot gets stuck in a point, where attractive and repulsive forces cancel each other out, we use transformation into the null space (INSERT: add citation of null space robotics manipulators) to only move the robot away from the obstacles in such a way as to not interfere with the main task. If not, the robot could not even approach the obstacles, which however is necessary in some situations, as it allows it to reach the goal points for which to reach the manipulator segments need to be positions in proximity to the obstacles. For this approach to work we need enough degree of manipulability left for the secondary task, so that the main / primary task doesnt overwhelm the secondary task. As described in the chapter bellow (ref: Attractive Velocity) we solve this problem by normalizing the primary attractive velocity.

this can still sometimes lead to loss of manipulability, add terciary, or secondary maipulability task

To add another level of certainity that the primary task wont overwhelm the secondary task we added the primary task execution slowdown that slows down the movement of the manipulator towards the primary goal and allows the secondary task more manipulability, so that it can better configure the arm in such a way, as to avoid the obstacles.

- optimization algorithm
- robot kinematics (include the exact-reduced method)
- primary task of distance goal

- secondary task of repulsive field
- damped least squares
- task slowdown option
- secondary task of manipulation measure
- different tasks merging equation

3.2 Constraints and Limits

- what is the point of this part anyways
- YES, BUT HOW DO WE APPLY THOSE - this chapter only makes sense if we describe how the limits are applied

- add x_{start} and x_{goal}

In robotic manipulators, effective path planning and motion control are subject to constraints related to free space, joint position limits, and joint velocity limits. These constraints are essential to ensure safety and practical functionality.

Free Space Constraints: These constraints ensure the manipulator operates in unoccupied space, avoiding obstacles. They are defined as:

$$\mathcal{C}_{free}(\vec{x}) = \begin{cases} 1, & \text{if } \vec{x} \text{ is in free space} \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

This binary representation indicates whether a point in the workspace is in free space or obstructed. During the execution of our algorithm, the repulsive field guides the entire manipulator into the empty task space, which in turn moves the manipulator in the obstacle free configutaion space.

Joint Position Limits: Each joint of the manipulator has a range of motion determined by its mechanical design. These limits prevent the joints from exceeding their design capabilities, ensuring reliable operation:

$$\theta_{i,min} \leq \theta_i \leq \theta_{i,max} \tag{2}$$

Here, $\theta_{i,min}$ and $\theta_{i,max}$ represent the minimum and maximum positions for each joint.

Joint Velocity Limits: Velocity limits are set for each joint to maintain controlled movements and prevent mechanical stress. These limits facilitate precise and stable motion:

$$\dot{\theta}_{i,min} \leq \dot{\theta}_i \leq \dot{\theta}_{i,max} \tag{3}$$

The values $\dot{\theta}_{i,min}$ and $\dot{\theta}_{i,max}$ specify the minimum and maximum velocities for each joint.

3.3 Attractive Velocity

-equation references
- citations
- check if quat log equation is correct

Our method ~~innovatively~~ employs inverse kinematics approach (IK) to guide the end effector (EE) towards its target, marking a departure from Khatib's joint coordinates approach in favor of a Cartesian coordinates framework. This is particularly beneficial in scenarios involving redundant manipulators, where determining an optimal goal joint configuration in advance is challenging.

When calculating translational velocity, we avoid the conventional gradient of the squared distance approach, which leads to high initial velocities and subsequently slow speeds near the target. Our aim is a consistent velocity throughout the trajectory, with controlled deceleration near the goal. This is achieved by first calculating the unit vector towards the target for direction, then modulating its magnitude using a sigmoid function, specifically the arctangent function, to prevent overshooting and ensure stable approaching motion.

$$\vec{v} = \frac{\vec{x}_{EE} - \vec{x}_g}{||\vec{x}_{EE} - \vec{x}_g||} \times \frac{\arctan(k_{sigm}\, ||\vec{x}_{EE} - \vec{x}_g||)}{pi/2} \tag{4}$$

In the above equation (eq. 4), $\vec{v}$ represents the end effector's translational velocity towards the target, combining direction and magnitude. The terms $\vec{x}_{EE}$ and $\vec{x}_g$ denote the current and goal positions of the EE, respectively, in Cartesian coordinates. The unit vector calculation, $\frac{\vec{x}_{EE} - \vec{x}_g}{||\vec{x}_{EE} - \vec{x}_g||}$, ensures motion directed towards the target. Finally, the sigmoid function, particularly the arctangent component, modulates this velocity to avoid overshooting, balancing speed and precision. The constant k_{sigm} allows us to set how close to the goal does the robot EE start slowing down.

The rotational velocity error of the EE is needed for ensuring goal orientation of the EE. In our approach, orientations are depicted using rotation matrices. Specifically, R represents the current EE orientation, while gR signifies the goal EE orientation. The disparity between these orientations is encapsulated by the relative rotation matrix dR. This matrix is formulated by multiplying the goal orientation matrix gR with the transpose of the current orientation matrix cR^T. To ensure that it represents a pure rotation without any scaling we than normalize the so gotten matrix .

$$dR = \frac{gR \cdot cR^T}{||gR \cdot cR^T||} \tag{5}$$

The relative rotation matrix value is converted into a quaternion, which is then logarithmically transformed ~~to represent the rotational error vectorially~~. The components of this quaternion, excluding the real part, than form the rotational error vector ω.

$$dR \mapsto dQ = a + b\,i + c\,j + d\,k \tag{6}$$

$$dQl = 2 \cdot \log(dQ) = al + bl\,i + cl\,j + dl\,k \tag{7}$$

There needs to be an explanation why log, where is this from. Maybe a reference.

$$\vec{\omega} = \begin{bmatrix} bl \\ cl \\ dl \end{bmatrix} \tag{8}$$

To get the full velocity of the end effector (EE), $\tilde{\mathrm{v}}_{\mathrm{ATT}}$, we combines translational and rotational velocities, which we scale using proportional gains k_p and k_r.

$$\tilde{\mathrm{v}}_{\mathrm{ATT}} = \begin{bmatrix} k_p \times \vec{v} \\ k_\omega \times \vec{\omega} \end{bmatrix} \tag{9}$$

3.4 Repulsive Velocity

We compute obstacle distances in task space with our unique matrix kernel method. Focusing on task space is beneficial as it provides a more direct and realistic representation of the environment, leading to better spatial awareness, quicker responsiveness, easier sensor integration, and more effective obstacle avoidance. Our method is particularly effective when integrating noisy sensor inputs like LIDAR or depth cameras for obstacle avoidance, as it robustly filters and interprets spatial data.

- kinematic calculations
- null spaces calculations
- how to join multiple velocities - weights
- general overview

***

- object detection is done in task domain and not c-space (more logical)
- repulsive field calculation - matrix "convolution" method
- matrix size and shape selection
- equation for repulsive kernel values (non-linear)
- PLOT: (ERK) kernel graphics
- PLOT: (ERK) linear kernel graphics
- PLOT: kernel field shape
- interpolation of the repulsive field
- what if there are obstacles behind wall (usually not the case, depth sensors show only thin walls, some noise doesnt matter, non-linear kernel, possible additional pre-convolution to convert obstacle grid to edges)
- effecient calculation in dinamic environments, lacking prediction capabilities (MPC)
- good for working in "statistical, noisy" obstacle grids

3.4.1 Kernel Selection

3.4.2 Interpolation

4 Implementation

5 Results

3 PAGES

- include execution times
- PLOT: kernel on robot graphics

6 Discussion

1 PAGE

- add the limitations of such method (already mentioned by Khatib)
- the limitations of local search
- number of parameters that need to be tuned (are there actually that many?)

7 Conclusion