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 30, 2023

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

Abstract

1 Introduction

2 Background

1-2 PAGES

Maybe add a sentence about computer effeciency of the method, as it is not dependent on the number of obstacles.

- 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.

optimization algorithm diagram

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 section 3.4. 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 section 3.3 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

Another way we make sure that the primary task doesn't take over the avoidance task is using execution slowdown as described in section 3.3.1.

- robot kinematics the exact-reduced method
- secondary task of manipulation measure
- different tasks merging equation

3.1.1 Exact Reduced Inverse Kinematics Method

- cite žlajpah article
- explain how it works
- equation
- scalar simplification

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}

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.

$$\vec{v} = \frac{\vec{x}_{EE} - \vec{x}_g}{||\vec{x}_{EE} - \vec{x}_g||} \times \frac{\arctan(k_{sigm}\,||\vec{x}_{EE} - \vec{x}_g||)}{pi/2} \tag{1}$$

In the above equation (eq. 1), $\vec{v}$ 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 .

$$dR = \frac{gR \cdot cR^T}{||gR \cdot cR^T||} \tag{2}$$

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.

$$dR \mapsto dQ = a + b\,i + c\,j + d\,k \tag{3}$$

$$dQl = 2 \cdot \log(dQ) = al + bl\,i + cl\,j + dl\,k \tag{4}$$

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

$$\vec{\omega} = \begin{bmatrix} bl \\ cl \\ dl \end{bmatrix} \tag{5}$$

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.

$$\tilde{\mathrm{v}}_{\mathrm{ATT}} = \begin{bmatrix} k_p \times \vec{v} \\ k_\omega \times \vec{\omega} \end{bmatrix} \tag{6}$$

3.3.1 Primary Task Slowdown

To ensure the primary task doesn't overpower the secondary task, we've integrated a primary task execution slowdown. This mechanism can reduce the manipulator's velocity towards its primary goal, leaving more maneuverability space for the secondary tasks.

$$\xi_p = \frac{1}{1 + \kappa_{\mathrm{sec}} \left(\frac{1}{\delta_{min}} \right)} \tag{7}$$

Is this equation correct, maybe add a singularity damping factor.

The slowdown factor (eq. 7) is influenced by the constant κ_{sec} and the robot's minimum distance from an obstacle δ_{min}. We calculate the minimal distance in the repulsive velocities phase of the algorithm, as explained in section 3.4. As per the equation, a large δ_{min} minimizes the slowdown effect, allowing uninterrupted primary task execution. Conversely, a small $\delta_{\min}$ increases the slowdown, by making the ξ_p factor smaller, giving the secondary task more time for corrective actions.

3.4 Repulsive Velocity

In our method, we compute repulsive velocities within the task space using a novel matrix kernel multiplication approach. Concentrating on the task space is advantageous as it provides a more direct and realistic representation of the environment. ~~This leads to improved spatial awareness, faster responsiveness, and is particularly well-suited for integrating noisy sensor inputs like LIDAR or depth cameras, efficiently processing and using this spatial information to accurately determine repulsive velocities, even amidst noise.~~ add some references for this statement

Repulsive velocities tell the manipulator in which direction to move, so that it avoids nearby obstacles. These velocities drop to zero when the manipulator maintains a minimum safe distance from obstacles, and rise to their highest when it nears an obstacle, facilitating immediate evasive action.

Our goal is to calculate repulsive forces at various points of interest on the manipulator. These points add reference are strategically distributed across the manipulator to ensure that the entire area of the robot is covered by the matrices of the repulsive field.

During the calculation of the repulsive field, kinematic equations are utilized to transform the position of points from the robot's internal joint space to the Cartesian global coordinate space.

The space of obstacles in which our robot operates is represented as an occupancy grid, where individual voxels have values ranging from 0 (indicating no obstacle presence) to 1 (certainty of an obstacle), with typical values lying somewhere between these extremes. Since the obstacle space is discrete (has finite resolution), while the Cartesian space is continuous, we propose two methods for mapping from Cartesian space to the occupancy grid space. The simpler approach involves mapping the point directly to the center of the nearest occupancy grid voxel, based on Euclidean distance. However, this discretization can sometimes lead to discontinuities. Therefore, we propose a second approach: linear interpolation of the calculated repulsive field to achieve a continuous field value.

Once the position of a point on the manipulator has been mapped to the obstacle space using one of these methods, we can employ our repulsive convolutional kernels to calculate repulsive velocities. For each Cartesian direction (x, y, z), we will generate a repulsive kernel, as described in the section 3.4.1. Then, from the obstacle grid, for each of the three directions, we extract a matrix cutout window of the same dimension as the kernels, centered on the point of interest. By performing the Hadamard (element-wise) product of the extracted obstacle grid window and our repulsive matrix, we obtain a resultant matrix. Summing all the elements of this matrix gives us the repulsive velocity for each of the three Cartesian directions at the selected point.

PLOT: POI on the ROBOT + ONE OR MORE KERNELS ON THE ROBOT

$$\tilde{\mathrm{v}}_{\mathrm{poi}} = \begin{bmatrix} \sum_{(i,j,k)} W_x \odot K_x \\ \sum_{(i,j,k)} W_y \odot K_y \\ \sum_{(i,j,k)} W_z \odot K_z \end{bmatrix} \tag{8}$$

-there is an option of interpolation plot different matrixes, that are sligthely moved ...
-in results maybe compare smoothenss of interpolated and non-interpolated method

- 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
- 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

The fundamental concept of our directional kernels lies in computing the repulsive field individually for each direction within the Cartesian coordinate system. Our filters can be likened to a variation of the Sobel reference operator, a 2D convolutional reference filter frequently utilized in computer vision for calculating image gradients at specific points. However contrary to the Sobel operator, our filter is not calculating the gradients of the matrix, as the inputs are weighted depending on the distance from the center of the kernel.

Our kernels are designed as three-dimensional structures with a primary axis aligned along a specific Cartesian direction, corresponding to the calculated repulsive velocity. The two secondary axes are orthogonal to this primary axis. The distribution of values along the primary axis is inversely symmetric, exhibiting positive values on one side and negative values on the other, with the magnitude reaching zero at the kernel's center. The nature of the increase in magnitude along the primary axis of the kernel defines the shape of the repulsive velocity field, determining how the repulsive velocity escalates as the manipulator approaches an obstacle.

The length of the primary axis is critical, as it dictates the detection range for obstacles. Longer kernels can detect obstacles further away from the robot, essentially extending the 'safety zone' around the robot. If the primary axis is too long, it can lead to extra calculations and may cause the robot to unnecessarily avoid obstacles that aren't in its immediate path, making its movement and path planning less efficient. A kernel with a primary axis that is too short might restrict the robot's ability to maneuver, detecting obstacles potentially too late, compromising the robots capacity to avoid obstacles effectively. Moreover, it is essential for the magnitudes at the kernel's periphery to be minimal, promoting a smooth increase in repulsive velocity rather than a sudden spike.

In the orthogonal axes, the magnitude of values diminishes with increasing distance from the core of the kernel, nearing zero at the edges using gaussian distribution. This gradation is vital for a smooth velocity transition when obstacles emerge in the kernels range. The peak magnitude is located at the kernel's core, along the primary axis. The polarity of the values is determined by their position relative to the primary axis. The length of the kernel along the secondary axes influences the peripheral detection range for obstacles. Excessively large kernels may generate repulsive velocities unnecessarily, whereas too narrow kernels might only detect obstacles aligned directly with the Cartesian direction in the point of interest. Therefore, when selecting the width and height of the kernel, one must consider the density of the neighboring points of interest on the robot, ensuring that their collective fields adequately cover the manipulator's surrounding area.

PLOT: kernel with gaussian functions

EQUATION: gaussian equations

- choice of velocity profile (gaussian, linear ... maybe could move this part to implementation / experiments)

3.4.2 Interpolation of the Repulsive Velocity

It is essential that the velocity contributions affecting the robot change smoothly. However, since our obstacle grid is discretely defined, achieving perfect continuity can be challenging. Increasing the resolution of the obstacle field can theoretically bring us closer to continuous behavior, but in practice, we are constrained by finite resolution. To ensure that the velocity remains continuous when transitioning from one cell of the obstacle grid to another at a point of interest (POI), we employ trilinear interpolation. ~~This technique allows for a smooth and continuous linear approximation of velocities in all three Cartesian directions (x, y, and z) as the POI moves between cells.~~

We start by scalling the coordinates of POI into the grid koordinate system, by multiplying it by grid resolution (eq. 9).

$$\vec{P} = \vec{p}_{\text{POI}} \times \Delta\text{grid} \tag{9}$$

We get the indexes of the surrounding cells by first scaling the POI position by grid resolution and than rounding the position to the nearest lower and upper integer positions (eq. 10).

$$\vec{P} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} \lfloor \vec{p}_{\text{POI}}(1) \rfloor & \lceil \vec{p}_{\text{POI}}(1) \rceil \\ \lfloor \vec{p}_{\text{POI}}(2) \rfloor & \lceil \vec{p}_{\text{POI}}(2) \rceil \\ \lfloor \vec{p}_{\text{POI}}(3) \rfloor & \lceil \vec{p}_{\text{POI}}(3) \rceil \end{bmatrix} \tag{10}$$

Once we got the indexes of the eight surrounding cells of our POI, we use our kernel matrix multiplication method, to calculate the 3x1 repulsive velocity vectors for all the cells (eq. 11).

$$\vec{V}rep_{xyz,ijk} = \text{calc\_rep\_vel}(X[i], Y[j], Z[k]) \quad \forall i, j, k \in \{1, 2\} \tag{11}$$

Trilinear interpolation method works on a 3-dimensional regular grid. Before we can start with the interpolation we need to calculate the distance between POI and smaller coordinates of the cells where we calculated the repulsive velocities (eq. 12). ~~Since the repulsive values we calculate for the cells are alligned with the centers of the cells, we need to move before the interpolation the positions of known grid points by half of the cell width.~~ The calculated repulsive velocity values are located at the centers of the cells. Therefore, before interpolation, we shift the values of the cells coordinates by half the resolution of the obstacle grid for each direction.

$$\Delta\vec{P} = \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} = \begin{bmatrix} \frac{\left(P_x - \left(X(1) + \frac{1}{2}\Delta\text{grid}\right)\right)}{(X(2) - X(1))} \\ \frac{\left(P_y - \left(Y(1) + \frac{1}{2}\Delta\text{grid}\right)\right)}{(Y(2) - Y(1))} \\ \frac{\left(P_z - \left(Z(1) + \frac{1}{2}\Delta\text{grid}\right)\right)}{(Z(2) - Z(1))} \end{bmatrix} \tag{12}$$

The result of the interpolation is independent of the order of the operations. We first interpolate along the x-axis, followed by along the y-axis and finally along z-axis.

$$\vec{V}rep_{xyz,jk} = \vec{V}rep_{xyz,0jk}(1 - \Delta x) + \vec{V}rep_{xyz,1jk}\,\Delta x \quad \forall j, k \in \{1, 2\} \tag{13}$$

$$\vec{V}rep_{xyz,k} = \vec{V}rep_{xyz,0k}(1 - \Delta y) + \vec{V}rep_{xyz,1k}\,\Delta y \quad \forall k \in \{1, 2\} \tag{14}$$

$$\vec{V}rep_{xyz} = \vec{V}rep_{xyz,0}(1 - \Delta z) + \vec{V}rep_{xyz,1}\,\Delta z \tag{15}$$

The final result is a repulsive velocity vector that transitions smoothly between the discrete values calculated at distinct points in the obstacle grid.

PLOT: surrounding cells, interp grid

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
- good for parellelization
- number of parameters that need to be tuned (are there actually that many?)

7 Conclusion