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

March 18, 2024

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

Abstract

1 Introduction

1.1 Motion Planning

Kinematic redundancy [**?**, **?**] enables a manipulator to follow a predefined task space trajectory using the endeffector (EE), while simultaneously, optimising for an additional task with the remaining movement capacity without impacting the trajectory adherence. This is possible because the robot's degrees of freedom (DOF) go beyond what is required to perform the primary task. Consequently, the robot can adopt different joint configurations optimised according to the secondary task while performing the primary task. Common secondary tasks [**?**] include avoiding singularities, optimising the manipulability measure, minimising joint torques and avoiding obstacles in the operating space.

The task of finding the joint trajectories of a manipulator is called motion planning [**?**]. It consists of finding a sequence of joint configurations for a robot so that the robot can move along this path from its initial configuration to

the goal configuration without colliding with itself, static obstacles or other agents in the environment. In addition to collision avoidance, motion planning for manipulators can optionally take into account various constraints, such as position, velocity, acceleration or jerk constraints for the joint angle or end effector, precision of the end effector with respect to position and orientation, stability of the manipulator, avoidance of singularities, or any number of other criteria.

There are numerous methods for planning manipulator movements [**?**, **?**], they can rougly be seperated into global and local approaches. Global, sampling-based, methods such as PRM [**?**], RRT* [**?**, **?**], RRT-Connect [**?**], Informerd RRT* [**?**], BIT* [**?**] and others offer a globally optimal solution based on a global search in configuration space. However, the generated trajectories are not always smooth or optimal, and the performance of the methods may be insufficient for real-time operation.

~~Recently, a number of learning-based methods using data-driven techniques have been proposed to improve or accelerate the functionality of sampling-based methods.~~

~~Trajectory optimisation methods such as CHOMP, STOMP and TrajOpt, on the other hand, use optimization to improve an initial seed trajectory. Consequently, the optimality of the solution is highly dependent on this initial trajectory. Nevertheless, these methods are capable of generating smooth trajectories, and although they can be too computationally intensive for high DOF dynamic real-time environments, they are generally effective in finding constrained motion plans.~~

Local motion planning approaches employ optimization techniques, two common ones are inverse kinematics [**?**, **?**], that is based on finding a least squares solution of the manipulator joint velocities, and quadratic programming (QP) [**?**, **?**, **?**, **?**]. Both methods are fast, suitable for real-time applications in dynamic environments and provide smooth solutions. However, since they do not plan further than one step ahead, they tend to get stuck in local minima. Therefore, they are often combined with a higher-level planner, for global static environment based ~~motion~~ path planning, while local optimisation takes dynamic environment changes into account. In the following text we will focus on inverse kinematics based approaches.

1.2 Kinematic Obstacle Avoidance

~~Different control schemes have been proposed. Acceleration-based control excels in precise handling of motion changes, velocity-based control offers consistent and smooth movement, while force and torque-based control provides direct control over joint forces for robust physical interactions. (CITATIONS in COMMENT BELLOW)~~

Various researchers have adopted different approaches to kinematic avoidance, as detailed in [**?**, **?**].

Colbaugh and Glass (1989) [**?**, **?**] tackled this problem in a two-step process. Initially, they calculated the robot's end-effector trajectory. Subsequently, they

used optimization to enhance the robot's dynamic response during trajectory execution and for tasks like obstacle avoidance.

Sciavicco and Siciliano [?, ?], as well as Egeland [?], independently introduced the concept of task-space augmentation, later revisited by Searji [?]. This method involves extending the primary task Jacobian with the secondary task Jacobian into a square matrix, yielding a singular solution.

Maciejewski and Klein (1985) [?], and Nakamura and Hanafusa (1987) [?], presented the concept of task prioritization and a null space for avoidance. In this framework, the secondary task is restricted to velocities that do not directly affect the primary task's movement. Their method calculates avoidance joint velocities based on the minimum distance between a point on the manipulator and an avoidance point, object, or obstacle. Žlajpah [?] improved the concept by proposing a reduced operational space formulation, that reshapes the Jacobain of the avoidance task from three cartesian axis to only the direction of obstacle avoidance. Petrič [?] suggested a way to smoothly transition between avoidance and trajectory following tasks.

1.3 Artificial Potential Field

These approaches typically do not focus on the environmental context, often representing it as geometric primitives and then calculating the distance between these primitives and the lines describing the manipulator.

Khatib [?] proposes the concept of an Artificial Potential Field. This field consists of a repulsive component that deflects the end effector away from obstacles, depicted as geometric primitives, and an attractive component that draws the manipulator towards its target. In the following years many different modifications and improvement of the original APF idea have been proposed, often focused on removal of local minimas in the potential field. Khim and Khosla [?] suggested the use of harmonic functions to solve the problem of local minima. Pinto et al. [?] proposes to vary the field based on the distance of robot from obstacles to fill the local minima. Many researchers tried using APF as a heuristic to better guide sampling based approaches [?, ?, ?].

Many of the recent works focus on use of a variation of artificial potential field to plan motion of the manipulator. Xia et al. [?] uses a variation of APF for manipulator motion [?]. Park et al. [?] used a numerical Jacobian in combination with APF for motion planning. Zhang et al. [?] proposes dynamic repulsive field based on direction and speed between point on robot and obstacle, it also suggests decision making force that moves the robot away from certain local minima. Chen et al. proposes ~~an application~~ using APF in joint space and a variable kinematic optimization step [?]. Long [?] suggests creating motion plan of the manipulator using APF, he extends it using RRT to calculate virtual attractive point for the robot to move towards in case of local minima. Zhu et al. [?] proposes use of APF in combination with MPC, to plan in environment with dynamic obstacles.

1.4 Environment representation

To be able to generate collision free trajectory we need to have a representation of the static environment. One common approach involves enclosing the obstacles around the manipulator within a collection of simple geometric primitives and then constructing a tree-like structure to facilitate efficient navigation and path planning [?]. Two of the commonly used methods for this purpose include AABB (axis-aligned bounding boxes) [?, ?, ?] and OBB (oriented bounding boxes) [?, ?]~~and FDH (fixed directions hulls).~~. The problem is generation of such hulls based on partial sensor measurements of the environment. Han et al. [?] used a complicated pipeline to convert point-cloud sensor measurements to octree, than to voxel grid and finally into convex hulls, used for collision detections. Another significant method worth mentioning is the use of Octomaps [?]. Octomaps employ an octree data structure to represent 3D environments efficiently, making them particularly suitable for areas with large open spaces.

In real environment we usually collect depth data with LiDAR, radar of RGBD camera sensors, that usually return clouds of points, that tell us the distance of objects in robots environment. While approaches have been proposed to use the point-cloud data directly to find obstacle free areas for robot operation [?], the data is often further converted into voxel grids [?, ?, ?].

Voxel grid [?, ?] representation divides the space into a regular grid of volumetric elements, or voxels, which can be used to create a more manageable approximation of the environment. While this approach offers a balance between detail and computational efficiency, it can introduce discretization errors, particularly when modeling objects with smooth surfaces or intricate details. The fidelity of the representation is dependent on the size of the voxels: smaller voxels can capture more detail but require more memory and computation, while larger voxels result in coarser approximations but are more memory and computation efficient. Adaptive voxel grids have been explored [?], where the voxel size can vary throughout the space to provide higher resolution in regions of interest while conserving resources in less critical areas. Nießner et al. [?] proposed Voxel hashing, for more efficient memory management in sparse environments. Dryanovski et al. [?] proposed variable voxel height and saving information about occupied and about free measurements, as well as the information about number of measurements for a specific cell.

Voxel grids can incorporate probabilistic information [?, ?], such as in occupancy grid maps, where each voxel holds a probability indicating the chance of an obstacle's presence. The occupancy probability of a voxel can be updated dynamically using sensor measurements and Bayesian updating methods. As new sensor data is collected, the probabilities are revised to reflect the increased or decreased likelihood of the presence of an obstacle in the voxel space.

Another type of data that voxel ~~grids~~ fields can hold are ESDF (Euclidian Signed Distance Field) or TSDF (Truncated Signed Distance Field) [?], in which case each of the voxels contains information how far nearest obstacle is in its vicinity.

1.5 ESDF creation

ESDF grids can be generated directly using sensor measurements. Oleynikova et al. [**?**] proposed a method for calculating the ESDF from TSDF. Han et al. [**?**] proposed a way to integrate point-cloud data into ESDF using ray-casting.

Another way is to generate ESDF from occupancy grids, that can be previously generated using sensors or based on predefined maps. When generating ESDF from occupancy grid, common approach is the Brushfire method [**?**], that spreads from obstacles until it calculates the distance for every field on a grid. Jump Flooding Algorithm (JFA) [**?**] is a similar method, that can be implemented on a GPU for faster parallelized distance calculation.

Methods that allow for the dynamical distance field generation are rare. Zhou et al. [**?**] propose use of pairs of points on trajectory and obstacles in their quadcopter trajectory optimization algorithm.

-Wavefront
-D* Klančar
-Euclidian distnace algorithms
-distance transforms
-voronoi diagrams
-ray casting, bullet casting ...
10-ish citatov iz seznama ni klicanih v tekstu !

-fix citation styles (sometimes et al, sometimes not, sometimes names, sometimes years ... not same format)

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 Kinematic Control

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. cite: DMP Quaternions article Petrič, Žlajpah, Ude

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

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, ADD a singularity damping factor. Maybe even some variation of a changing damping.

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 $\delta_{\min}$ 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.

THIS WILL TOTALLY STOP MOVEMENT IN TIGHT SPACES, NEED ADDITIONAL PART. (Maybe use some kind of derivative of Rep Field).

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.

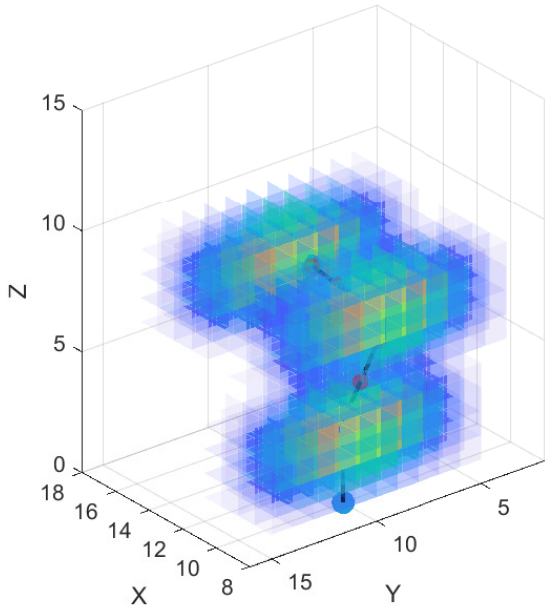
Our goal is to calculate repulsive forces at various points of interest on the manipulator. These points add reference are strategically distributed across the manipulator to ensure that the entire area of the robot is covered by the matrices of the repulsive field.

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.



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

-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 share a passing similarity to the Sobel operator, a 2D convolutional filter frequently utilized in computer vision for calculating image gradients at specific points.~~

Our kernels are designed as three-dimensional structures with a primary kernel axis aligned along a specific Cartesian direction, corresponding to the calculated repulsive velocity. The two secondary kernel 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 jump between max positive and max negative magnitude at the kernel's center. The function 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 changes as the point on 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 (eq. 9). Moreover, it is essential for the magnitudes at the kernel's periphery to be minimal, promoting a smooth increase in repulsive velocity when approaching the obstacle rather than a sudden spike.

$$num_{primary} = \frac{2 \times \text{range}}{\Delta\text{grid}} \tag{9}$$

The length of the orthogonal axes influences the peripheral detection range for obstacles. Excessively wide kernels may generate repulsive velocities for objects that are not in the path of the robot, whereas too narrow kernels might only detect obstacles aligned directly with the Cartesian direction in the point of interest. When selecting the width and height of the kernel, we must consider the density of the neighboring points of interest on the robot, ensuring that the

~~collective fields~~ combination of kernels adequately cover the entire 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. 10).

$$\vec{P} = \vec{p}_{\text{POI}} \times \Delta\text{grid} \tag{10}$$

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. 11).

$$\vec{P} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} \lfloor \vec{p}_{\text{POI}}(1) \rfloor & \lceil \vec{p}_{\text{POI}}(1) \rceil \\ \lfloor \vec{p}_{\text{POI}}(2) \rfloor & \lceil \vec{p}_{\text{POI}}(2) \rceil \\ \lfloor \vec{p}_{\text{POI}}(3) \rfloor & \lceil \vec{p}_{\text{POI}}(3) \rceil \end{bmatrix} \tag{11}$$

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. 12).

$$\vec{V}rep_{xyz,ijk} = \text{calc\_rep\_vel}(X[i], Y[j], Z[k]) \quad \forall i, j, k \in \{1, 2\} \tag{12}$$

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. 13). ~~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.

$$\Delta\vec{P} = \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} = \begin{bmatrix} \frac{\left(P_x - \left(X(1) + \frac{1}{2}\Delta\text{grid}\right)\right)}{(X(2) - X(1))} \\ \frac{\left(P_y - \left(Y(1) + \frac{1}{2}\Delta\text{grid}\right)\right)}{(Y(2) - Y(1))} \\ \frac{\left(P_z - \left(Z(1) + \frac{1}{2}\Delta\text{grid}\right)\right)}{(Z(2) - Z(1))} \end{bmatrix} \tag{13}$$

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.

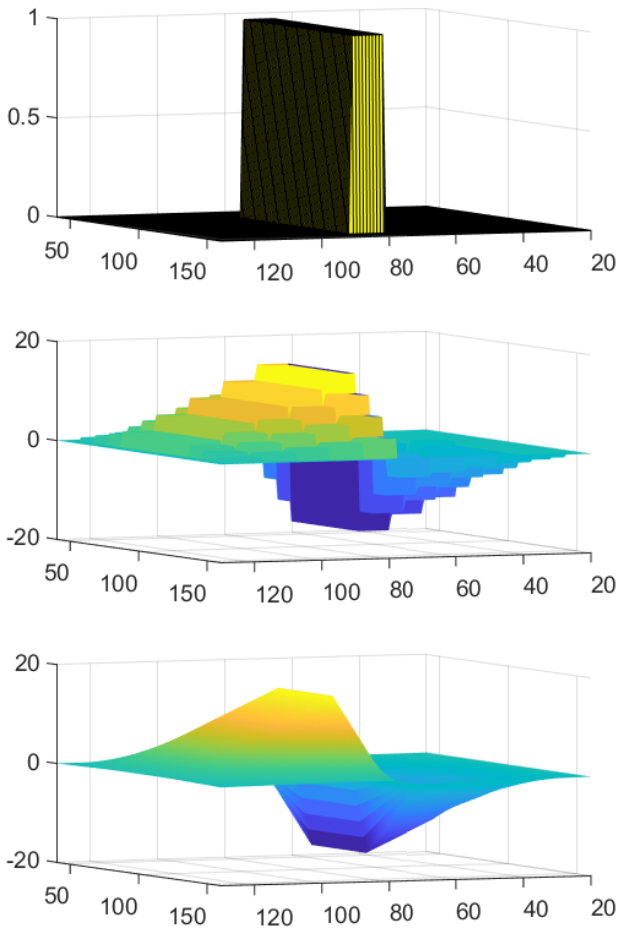
$$\vec{V}rep_{xyz,jk} = \vec{V}rep_{xyz,0jk}(1 - \Delta x) + \vec{V}rep_{xyz,1jk}\,\Delta x \quad \forall\, j,k \in \{1,2\} \tag{14}$$

$$\vec{V}rep_{xyz,k} = \vec{V}rep_{xyz,0k}(1 - \Delta y) + \vec{V}rep_{xyz,1k}\,\Delta y \quad \forall\, k \in \{1,2\} \tag{15}$$

$$\vec{V}rep_{xyz} = \vec{V}rep_{xyz,0}(1 - \Delta z) + \vec{V}rep_{xyz,1}\,\Delta z \tag{16}$$

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
- IMPORTANT: maybe add 2D mobile platform results
- use pybullet or some engine that tells you distance from obstacles and run the same task, compare distances after execution, smoothness of paths (MAYBE FOR PUBLICATION, NOT CONFERENCE)

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