A Heuristic Framework for Path Planning the Largest Volume Object from a Start to Goal Configuration

Evan Shellshear and Robert Bohlin
Fraunhofer Chalmers Centre, Gothenburg, Sweden

Keywords: Path Planning, Heuristic, Largest Volume.

Abstract: In this article we present a heuristic algorithm to compute the largest volume of an object in three dimensions that can move collision-free from a start configuration to a goal configuration through a virtual environment. The results presented here provide industrial designers with a framework to reduce the number of design iterations when designing parts to be placed in tight spaces.

1 INTRODUCTION

Being able to determine whether a virtual object can pass through a virtual environment collision-free is a fundamental problem in virtual design. Depending on the exact problem at hand, there exist hundreds of papers addressing this problem from all sorts of aspects. In this paper we focus on a lesser studied problem of utmost importance for industrial designers. We are interested in being able to compute the largest object than can travel collision-free from a start configuration to a goal configuration. This academic investigation is motivated by numerous real life problems and the basic motivation for this article is the virtual verification and automation of car designs. In particular, one wants to know if a new car design can pass through an assembly line without colliding with other objects and if it does collide, what are the minimal design changes that need to be made to avoid the collisions. This information can also be used for future design problems if the environment and trajectory remain the same (as is typical for factory installations).

The most important contribution of the results in this paper is to grant the designer the ability to carry out product design changes much earlier in the production phase. It is well-known that the later the design changes occur during the production cycle the more expensive these changes are, (Folkestad and Johnson, 2002). Hence, the framework presented here could be used to save significant amounts of money by allowing designers to figure out correct and allowable designs much earlier in the design phase. Typical applications where computing the non-colliding design of numerous parts play an important role include, *inter alia*, the ability to route and path plan objects through tight spaces such as engines (Hermansson et al., 2012) and other assembly components or even assembly lines, designing robots for tight spaces in path planning applications (Spensieri et al., 2013; Spensieri et al., 2008; Carlson et al., 2013) as well as almost any other product design scenario where the assembly of parts occurs. Once the set of colliding parts has been discovered one can then apply any of numerous algorithms to effect the necessary design changes to avoid collisions, (Björkenstam et al., 2012; Vanderhyde and Szymczak, 2008).

The problem we are interested in can be defined more exactly as follows. Let a start configuration (position and orientation) and goal configuration in $\mathbb{R}^3$ be given. Then we wish find a path from the start configuration to the goal configuration that allows an object of the largest possible volume to pass collision-free along it. In this paper we will not pursue the ambitious goal of finding the global maximum volume. We will compute a local maximum volume by adjusting the positions and orientations of a propitious initial path. The work presented in this paper extends the results and algorithms presented in (Shellshear et al., 2014) and (Ilies and Shapiro, 1999), where the path-planning object's path is fixed.

The remainder of this paper is organized as follows. In the next section we present our algorithm and its analysis. In the final section we conclude.

Shellshear E. and Bohlin R..
A Heuristic Framework for Path Planning the Largest Volume Object from a Start to Goal Configuration.
DOI: 10.5220/0005002102640271
In *Proceedings of the 11th International Conference on Informatics in Control, Automation and Robotics* (ICINCO-2014), pages 264-271
ISBN: 978-989-758-040-6

2 THE ALGORITHM

In this section we present our algorithm to compute a path that maximizes the volume of an object to pass collision-free along it. Note that this maximization is only local and not global. This is because our method of optimization only considers paths that can be reached via successive small perturbations of the clearance maximizing path between the start and goal configurations. Throughout, unless stated otherwise, by clearance we mean minimum distance.

The first step of any such algorithm is to define a start and goal configuration. In addition to the start and goal configurations, the degrees-of-freedom (DOF) of the object are important too (LaValle and Kuffner Jr, 2001). In this paper, we allow the object, the volume of which is to be maximized, to move with six DOF, i.e. translations in any direction and any type of orientation in $\mathbb{R}^3$.

Given a start and goal configuration, Algorithm 3 computes a volume maximizing path between the start and goal configuration and returns a voxelized representation of the volume maximizing object as well as the path for it to take. We now give an overview of the main algorithm presented in Algorithm 3 and afterwards we then analyze each of the most important parts in detail.

The first step of the algorithm is to find a path from the start configuration to the goal configuration that will then be locally optimized. Because we want to maximize the volume of an object to move collision-free from the start to goal configurations, we choose initially to find a path that maximizes the volume of the largest ellipsoid that can pass along it. The choice of ellipsoid is based on the balance between the many DOF an ellipsoid has, which can be used to approximate a given shape, as well as its simplicity.

After having found an initial path that maximizes the volume of any ellipsoid moving from the start to goal configuration, we then double the ellipsoid in each of its three dimensions and voxelize the result. The larger ellipsoid is then moved along the previously computed path removing voxels from it as they collide with the surroundings. We store the set of colliding voxels and then working backwards from the last colliding voxel, we attempt to make local adjustments to prevent each voxel from colliding. This is presented in pseudocode in the while loop, starting at line 4 in Algorithm 1, with this new object and new path. We carry out this while loop until we have reached some cut-off criterion or maximum number of iterations. Then we return the best found object and its path.

After attempting to retain as many colliding voxels as possible, we then attempt to add voxels around the shape computed by Algorithm 1 in Algorithm 2. This is done by increasing the clearance from each voxel to its surroundings and adding new voxels next to voxels that have a large enough clearance.

We now present all three algorithms in detail.

Algorithm 1: AdjustPath(Path, Object, OriginalObject).

1: **Input**: The current path *Path* that the current object *Object* is to travel along.
2: **Output**: A path and object.
3: Let IncreaseVolume be a boolean and set it equal to **true**
4: **while** Iterations $<$ Max Number of Iterations **and** $IncreaseVolume =$ **true do**
5: Move *Object* along *Path* and use the last colliding cell(s) to adjust the path to prevent the cell(s) from colliding. Call the adjusted path *AdjustedPath* and the object *AdjustedObject*.
6: **if** Volume(AdjustedObject) $>$ Volume(Object) **then**
7: $Object \leftarrow AdjustedObject$.
8: $Path \leftarrow AdjustedPath$.
9: **else**
10: $IncreaseVolume =$ **false**
11: **end if**
12: **end while**
13: Return *Path* and *Object*

Algorithm 2: NewPath(CurrentPath, CurrentObject).

1: **Input**: The current path *CurrentPath* that the current object *CurrentObject* is to travel along.
2: **Output**: A path and object.
3: Path plan *CurrentObject* from the start to goal configuration of *CurrentPath* so as to maximize the clearance of it to the surroundings and record the clearance of each boundary voxel. Call this path *Path*.
4: $V \leftarrow Volume(CurrentObject)$.
5: **for** All boundary voxels of *CurrentObject* **do**
6: **if** Clearance of voxel along whole path $>$ size of voxel **then**
7: Add voxels on all free sides of the current voxel of *CurrentObject*.
8: **end if**
9: **end for**
10: **if** $V < Volume(CurrentObject)$ **then**
11: $CurrentPath \leftarrow Path$.
12: **end if**
13: Return *CurrentPath* and *CurrentObject*.

Algorithm 3: Computing the volume maximizing path for a user defined object between the start and goal configurations.

1: **Input**: A start and a goal configuration as well as an object that should pass along the path.
2: **Output**: A voxel representation of the largest object that can go from the start to the goal configuration based on the user defined object.
3: Let *OptPath* be an empty path.
4: Find an initial path from start to goal configurations that maximizes the volume of an ellipsoid. Set *OptPath* equal to this path.
5: Voxelize the volume defined by the user and call it *OrigObject*. Move *OrigObject* along the path from start to goal configurations removing colliding voxels from it. Call the remaining voxelized object *RemObject*.
6: $Path \leftarrow OptPath$.
7: $Object \leftarrow RemObject$.
8: $OptPath, RemObject \leftarrow AdjustPath(Path, Object, OrigObject)$
9: **while** $OptPath \neq Path$ **or** $RemObject \neq Object$ **do**
10: $Path \leftarrow OptPath$
11: $Object \leftarrow RemObject$
12: $OptPath, RemObject \leftarrow NewPath(Path, Object)$
13: **end while**
14: Return *OptPath* and *RemObject*.

2.1 Computing the Initial Path

The authors anticipate that a good choice of initial path for the maximum volume object will play a major role in the number of iterations of the while loop in Algorithm 3 and 1, which both attempt to increase the volume of the final object. Hence, we have developed a method of finding a good initial path that comprises a major part of Algorithm 3 and is the computation in Step 4 in Algorithm 3.

In finding the initial path we have a number of choices. One could simply take the clearance maximizing path, (Geraerts and Overmars, 2005). However, because one often has significant clearance in the direction of movement, we chose to find a path that maximizes the volume of a rotationally symmetry ellipsoid. Our goal is to produce a path that exploits additional clearance in at least one dimension, i.e. the direction of motion. In the following, because the ellipsoid is rotationally symmetric, we write r for the radii of the two equal axes (i.e. if we call the radii r_1, r_2 and r_3, then $r_2 = r_3 = r$). To compute the path for an ellipsoid with maximal volume, we start with the path of maximum clearance for a point.

There exist numerous methods to produce a clearance maximizing path between the start and goal configurations. Initially one could use any of numerous path planning algorithms, (Bohlin and Kavraki, 2000; LaValle and Kuffner Jr, 2001), to generate an initial path and then optimize this path to maximize clearance, (Geraerts and Overmars, 2005). Other methods involve including the clearance maximization criteria in the path planning algorithm itself, (Zheng et al., 2011; Kim et al., 2003). However one chooses to compute the path, we will assume it as given. As is usual in the path planning literature (Bohlin and Kavraki, 2000), our path will be defined by a set of n configurations, $\{x_i\}$, $i = 1, \ldots, n$, with the position and orientation between configurations x_i and x_{i+1} being defined by the linear interpolation of the position and orientation between both configurations.

The reason for us to initially begin with a clearance maximizing path is that it means that the path will be located on the 3D medial axis, (Geraerts and Overmars, 2005), between the start and goal configurations. This choice of path as the initial path to locally optimize is to allow the radii of the ellipsoid, orthogonal to the direction of motion, to be as large as possible because they will be constrained by the minimum distance from the ellipsoid's path to the surroundings. The reader should note that in real applications the medial axis can be disconnected and discontinuous. Hence, when the ellipsoid travels from the start to goal configuration, it is only important that the ellipsoid follows the medial axis along sections of the path that have low clearance and along other sections it is less important and the medial axis can be used more to guide the motion.

Given the initial path that determines the center point of our ellipsoid, we also wish to determine an initial orientation of the ellipsoid that maximizes the volume of the ellipsoid for the current path. With the assumption on the radii stated earlier (i.e. $r_2 = r_3 = r$), we fix the orientation of the first axis of the ellipsoid to be equal to the tangent of the medial axis. The tangent has the property that it is "locally optimal" because it points in the direction of the next part of the medial axis path, i.e. to the next point of maximal clearance, and hence should allow r_1 to be as large as possible (for the next instantaneous movement).

Given that we now have the center point and orientation of our ellipsoid, all that remains is to compute the radii that give the maximum volume. To do that we first create a corridor map of the medial axis as described in (Geraerts and Overmars, 2007). We know that any object contained in this corridor is collision-free. After having done this we then compute the maximum value of r_1, say R, for each configuration x_i by finding the first point intersected by a line in the

direction of the ellipsoid's first axis' orientation going through the center point. One can now use the corridor map to find the maximal values of r for each value of r_1, $0 \leq r_1 \leq R$. That is, we continually shorten the length of the first axis, r_1, and see how much we can increase r by shortening r_1. To make sure that these values provide a collision-free movement for the ellipsoid, we move the ellipsoid along the medial axis with radii equal to the current r_1 and r. If a collision occurs, then we reduce r and move the ellipsoid along the path again continuing in this way until a value is found for which the combination of r_1 and r is collision-free. For each value of r_1, we then save the maximum, collision-free value of r in a local table defining the relationship between r_1 and r at the configuration x_i. Once we have completed this computation for all configurations $\{x_i\}$, we then find the configurations that constrain r_1 and r the most and use them to choose radii that maximize the volume of ellipse given all constraints on r_1 and r. Note that these choices of radii are not guaranteed to be collision-free and we take up this point in the coming local optimizations of the ellipsoid's volume.

The previously computed ellipsoid is restricted in a number of ways. When finding the largest r_1 possible we only used the tangent to determine the orientation, which can be less than optimal for paths with sudden changes in the direction of the path. Hence, it could be possible that, by slightly deviating from these orientations and possibly also adjusting the center point of our ellipsoid, one can increase the volume of the ellipsoid. We now introduce a method to increase the volume of the ellipsoid by slightly perturbing its position and orientation so that such a perturbation leads to an increase in volume.

The function we are attempting to increase by locally perturbing the ellipsoid's position and orientation is given in Equation 1. We attempt to maximize the volume, Equation 1, while making sure that the ellipsoid is collision-free, Equation 2, and simultaneously perturb the ellipsoid's radii, centerpoint and orientation. This is carried out at each configuration x_i. However, when doing this, we first start at the configuration x_i that defines the maximum collision-free values of r_1 and r and attempt to increase the radii there. If this succeeds, then we move onto the next configuration that defines the current maximum collision-free values of r_1 and r. We repeat this until it is no longer possible to increase r_1 and r.

We now explain the notation presented in Equations 1 and 2. For each configuration x_i, let p_j $(j = 1,\ldots,m)$, be the points on the ellipsoid that are closest to the surroundings and let σ_j stand for the clearance for each point p_j. Let the outward surface normals to

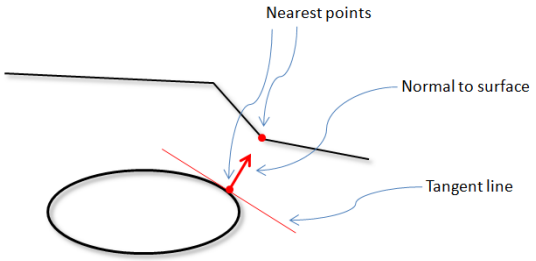

Figure 1: Normal to an ellipse's surface.

the ellipsoid at these points be N_j, i.e. the vector perpendicular to the tangent plane at p_j and which also face out of the ellipsoid (see Figure 1). Let c stand for the ellipsoid's center point and o stand for the ellipsoid's orientation.

$$\max V(c,o,r_1,r) \tag{1}$$

$$\left(\frac{\partial p_j}{\partial c}+\frac{\partial p_j}{\partial o}+\frac{\partial p_j}{\partial r_1}+\frac{\partial p_j}{\partial r}\right)\cdot N_j \leq \sigma_j,\ j=1,\ldots,m. \tag{2}$$

To solve this problem we linearize each of the functions in Equation 1 and 2. In the following, V_0 is the original volume, A is the first order approximation to the change in the volume gradient, $(p_j)_0$ is the current p_j value and P_j is the first order approximation to the change in p_j. We also denote the vector (dc,do,dr_1,dr) by dz.

$$V \approx V_0 + Adz \tag{3}$$

$$p_j \approx (p_j)_0 + P_j dz,\ j=1,\ldots,m. \tag{4}$$

To these equations we add another equation to guarantee that between configurations the ellipsoid will not collide with its surroundings. In the following equation let ε_i stand for the minimum distance between the ellipsoids and its surrounds between the current configuration x_i and the next for the volume maximizing ellipsoid as computed earlier,

$$|dc| + r_1|do| + |dr_1| + |dr| < \varepsilon_i. \tag{5}$$

The r_1 that $|do|$ is multiplied by is there to provide an upper bound on the movement of any point on the ellipsoid when o changes by do. By letting V^T denote the transpose of a vector V, we use these linearized forms to write the linearized version of the maximization problem presented in Equation 1 and 2, with the added collision-free guarantee, as

$$\max Adz \tag{6}$$

$$(N_j)^T P_j dz \leq \sigma_j,\ j=1,\ldots,m, \tag{7}$$

$$|dc| + r_1|do| + |dr_1| + |dr| < \varepsilon_i. \tag{8}$$

The above uses the fact that the slight change in p_j, $dp_j = P_j dz$, should fulfill $dp_j \cdot N_j \leq \sigma_j$. Another

restriction that we can add that reduces the possible values of dc is that we only want to look at movements of the ellipse center point, c, that are orthogonal to the path because movements along the path can be incorporated by applying the maximization problem there. We define the orthogonal directions to be orthogonal to the tangent of the medial axis at x, $t(x)$. Hence our complete optimization problem becomes,

$$\max Adz \tag{9}$$

$$(N_j)^T P_j dz \leq \sigma_j,\ j = 1,\ldots,m, \tag{10}$$

$$dc \cdot t(c) = 0, \tag{11}$$

$$|dc| + r_1|do| + |dr_1| + |dr| < \varepsilon_i. \tag{12}$$

In the previous set of equations, Equation 10 guarantees that, as we optimize the ellipsoid's shape at the current configuration, the ellipsoid will adjust it's position in a locally optimal way. Equation 12 guarantees that these changes are so that the optimized ellipsoid has a shape that is guaranteed to be collision-free between the current configuration and the next (by definition of ε_i).

To compute A and P_j we proceed as follows. As is clear changing the position or orientation of the ellipsoid does not effect the volume. However increasing r_1 and r does. As the volume of the ellipsoid is equal to:

$$\frac{4}{3}\pi r_1 r^2, \tag{13}$$

the reader can check that the linearization of A is then equal to (dz is 9 element vector: 3 elements for small position changes, 3 elements for the small orientation changes, 1 for each radii change)

$$(0,0,0,0,0,0,\frac{4}{3}\pi r^2, \frac{4}{3}\pi r_1 r, \frac{4}{3}\pi r_1 r). \tag{14}$$

The P_j are computed by noticing that movements in the position of the ellipsoid directly affect the corresponding position of the points p_j, hence we just add their contributions to the points. When a point is rotated through Euler angles α, β and $\gamma \approx 0$, then we can use the following linearization of the rotation for small angles to compute the movement of the point (Nüchter et al., 2010),

$$\begin{pmatrix} 1 & -\gamma & \beta \\ \gamma & 1 & -\alpha \\ -\beta & \alpha & 1 \end{pmatrix}. \tag{15}$$

Computing the effects of increases or decreases in the radii on the coordinates of p_j can be carried out as follows: assume first of all that the ellipsoid axes are aligned with the Cartesian axes and that the ellipsoid is located at the origin (although the location of the ellipsoid does not effect the size of the increase or decrease in any of the radii). To simplify the coming notation we consider the case where we only adjust r_1, the other cases being analogous. Note that one can use the parametrization of a point on an ellipsoid to compute its movement in the direction of an axis caused by changes in the radii. Points on an ellipsoid can be represented by the following parametric equations,

$$x = r_1 \cos u \cos v \tag{16}$$

$$y = r \cos u \sin v \tag{17}$$

$$z = r \sin u, \tag{18}$$

where $-\frac{\pi}{2} \leq u \leq \frac{\pi}{2}$ and $-\pi \leq v \leq \pi$. Hence for a point at (x,y,z) with angles $(u,v) = (\theta,\phi)$ an increase in radius of Δr_1 results in a movement of the r_1 axis of $\Delta r_1 \cos\theta \sin\phi$.

Hence, given the orientation (in Euler angles) of the ellipsoid as (α,β,γ) (with corresponding rotation matrices $R(\alpha)$, $R(\beta)$ and $R(\gamma)$), the final movement of the point, located at (θ,ϕ) in parametric coordinates, caused by the adjustment in the first radius is equal to $\Delta r_1 \Delta_1$, whereby

$$\Delta_1 := R(\alpha)R(\beta)R(\gamma)(\cos\theta\sin\phi, 0, 0). \tag{19}$$

Similarly for the other two radii, let Δ_k with $k = 2,3$ be the adjustment for the radius corresponding to the second and third ellipsoid axis respectively. The correctness of this is guaranteed by the fact that for a rotation R, vector v and scalar λ, $R(\lambda v) = \lambda R v$. Hence P_j has the form

$$\begin{pmatrix} 1 & 0 & 0 & 1 & -\gamma & \beta & \pm(\Delta_1)_x & \pm(\Delta_2)_x & \pm(\Delta_3)_x \\ 0 & 1 & 0 & \gamma & 1 & -\alpha & \pm(\Delta_1)_y & \pm(\Delta_2)_y & \pm(\Delta_3)_y \\ 0 & 0 & 1 & -\beta & \alpha & 1 & \pm(\Delta_1)_z & \pm(\Delta_2)_z & \pm(\Delta_3)_z \end{pmatrix}$$

where the plus or minus is determined by where p_j is located with respect to the corresponding axis plane and $(\Delta_k)_x$ stands for the x coordinate of the Δ_k vector (similarly for y and z).

If the previous optimization is successful, then we can repeat it until no more increases in the ellipsoid's volume are possible. Figure 2 demonstrates the previous optimization applied to an ellipse.

2.2 Voxelizing the Ellipsoid

In this section we explain Step 5 in Algorithm 3. To make sure that we are able to gain the maximum benefit from the algorithms we double the ellipsoid's radii in all three dimensions and then voxelize the resulting ellipsoid. We voxelize the entire volume with voxels of a fixed size, say $\delta > 0$. The choice of delta is influenced by the available computational power. A smaller δ results in more tests and hence a higher computational demand. To voxelize the ellipsoid, we find the bounding box of the ellipsoid and divide the

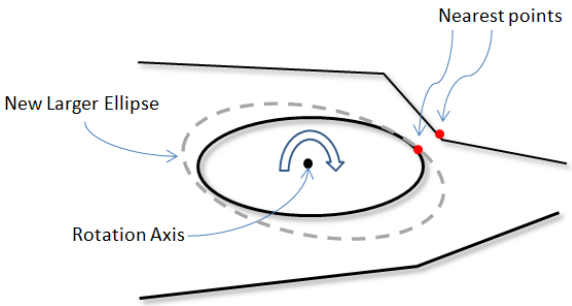

Figure 2: Increasing the ellipse's volume while rotating and translating it.

box into small voxels and find supercover of the ellipsoid (i.e. all voxels that meet the ellipsoid, (Cohen-Or and Kaufman, 1995)) . These voxels will then define a (26,6)-neighbor closed surface (Cohen-Or and Kaufman, 1995) (i.e. any of the 26 voxels surrounding a voxel are a neighbor to the voxel if they are non-empty and all empty voxels surrounding a voxel are neighbors if they are adjacent to any one of the voxel's faces) because they voxelize the closed surface defined by the ellipsoid. We then remove all voxels of the bounding box that are not part of the boundary or inside of the object. We do this by finding all boundary voxels of the voxelized bounding box of the ellipsoid that are not intersected by the ellipsoid. We then carry out a breadth-first search of all such voxel's 26-neighbors that are not part of the supercover of the voxels intersected by the ellipsoid.

2.3 Path Planning with the Voxelized Ellipsoid on the New Path

After having computed the initial path and voxelization of the ellipsoid, we then move our voxelized object along the initial path (given by *OptPath* in Algorithm 3) adjusting its configuration to remove collisions. This is the content of Algorithm 1.

To move the voxelized ellipsoid along the path and find any voxel colliding with surroundings, we use the technique in (Shellshear et al., 2014) (although one could also use the algorithms presented in (Ilies and Shapiro, 1999)). The method presented in (Shellshear et al., 2014) computes the set of all colliding voxels along the path from the start to goal configurations. The technique presented in (Shellshear et al., 2014) is also able to find the colliding voxels, which we will require below. For simplicity, in the following sections we will call the remaining voxelized volume the voxelized ellipsoid even though the remaining shape after removing all colliding voxels may not look like an ellipsoid at all.

As the voxelized ellipsoid moves along the path

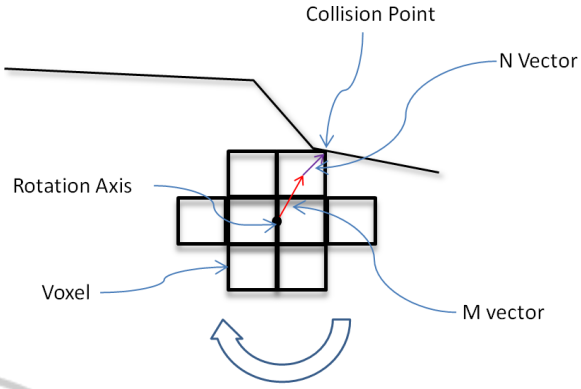

Figure 3: Rotating the voxelized ellipsoid (in 2D) to prevent collisions with voxels.

we find all colliding voxels and store them. Once the voxelized ellipsoid has completed the motion, we then use the set of colliding voxels to locally adjust the path to try and avoid collision along the path. We start with the last colliding voxel along the path. We know that if we can locally adjust (these adjustments are described in more detail below) the movements of the voxelized ellipsoid during this part of the path so that the voxel is not colliding and does not introduce any new collisions, then the voxel will not collide with anything else on the path before this point because it was not colliding previously. Hence each local adjustment can be done without affecting previous collisions along the path. So if we are successfully able to locally adjust the path, then we let the new set of voxels travel along the path from the current configuration until the end and see if the newly added voxel collides with anything else. If it does collide with something, then we repeat the previous local adjustments to the path at the point of collision and repeat the aforementioned.

We now describe how to carry out these local adjustments. To locally adjust the set of voxels, we created a number of slight adaptations on a methods typically used in computer graphics to remove penetrations, (Zachmann et al., 2000). So to locally adjust the set of voxels so that they are no longer colliding, we developed four different strategies. The first is the simplest and involves attempting to rotate the voxelized ellipsoid to remove the collision. Let N be the vector pointing from the centerpoint of the colliding voxel to the point of contact with the surroundings and M be the vector from the voxelized ellipsoid's centerpoint to the colliding voxel's centerpoint, see Figure 3. We then rotate the voxelized ellipsoid away from the colliding voxel around the axis equal to the cross product of N and M and passing through the objects centerpoint. We do this until another collision occurs or the voxel is no longer colliding. If

rotating in such a way cannot move the voxelized ellipsoid into a collision-free configuration, then we try translating the voxelized ellipsoid in the direction of the colliding voxel's centerpoint to the voxelized ellipsoid's centerpoint until the same happens. If neither works then we attempt to translate (in the same direction as before) until a new collision happens and then try to rotate as before and vice versa.

We then carry this out working through the colliding voxels always looking at the one that collided previously (i.e. collided before the current one we are analyzing). We do this until some cut-off criterion is reached or until the previous attempt to prevent the voxel from colliding failed at which point we stop trying to add voxels.

2.4 Path Plan with the Voxelized Ellipsoid and Adding Voxels Around Boundary Voxels

We now explain the content of Algorithm 2. In this algorithm we path plan with the voxelized ellipsoid, computed from Algorithm 1. We plan a path so that the path maximizes the clearance from the voxelized ellipsoid to its surroundings. Once we have found the clearance maximizing path for our voxelized ellipsoid with any suitable algorithm, (Zheng et al., 2011; Kim et al., 2003), we then also measure the minimum distance from each boundary voxel to the surroundings along the path. This can be efficiently achieved via any number of methods, e.g. bounding volume hierarchies (Larsen et al., 1999), distance fields (Zachmann et al., 2000), etc. For each boundary voxel we save the minimum distance found over the entire path.

After completing this (Step 3 in Algorithm 2), we then have a list of minimum distances for all voxels in the set of boundary voxels. We wish to now add voxels to all sides of a given boundary voxel (that are not already occupied by boundary voxels), that has a large enough clearance to guarantee that the newly added voxels will not collide with the surroundings on the current path. To provide this guarantee we only add voxels to all unoccupied 6-neighbor sides of a voxel if the clearance of the voxel to the surroundings is greater than or equal to the side length but less than a face diagonal of the voxel. If the clearance is greater than or equal to a face diagonal of the voxel but less than the major diagonal of the voxel, then we add all unoccupied 18-neighbors, (Cohen-Or and Kaufman, 1995), around the voxel. Otherwise if the clearance is greater than or equal to the major diagonal of the voxel then we add voxels to all unoccupied 26-neighbors, (Cohen-Or and Kaufman, 1995), of the current voxel. By adding voxels in this fashion we can guarantee that the newly added voxels are collision-free. This is because the minimum distance from any part of the newly added voxel will be less than the current voxel's clearance.

After carrying out this part of the algorithm, the final computed object is a set of voxels approximating the largest object that can pass collision-free from the start to goal configurations.

3 PRACTICAL IMPLEMENTATION DETAILS

Due to space limitations we present a possible practical implementation of the previous results but not actual simulation results. Such results will be presented in a future publication. In each step requiring path planning the maximimum clearance, the algorithm in (Kim et al., 2003) can be used. To compute the pertubations of the initial ellipsoid to maximize its volume, a suitable optimization method would be to solve it via linear programming using tools such as (COIN-OR, 2014). Voxelizing the ellipsoid can be carried out using the straight-forward implementation from (Cohen-Or and Kaufman, 1995). To compute the maximum volume along the new path given the voxelization, the fast algorithm presented in (Shellshear et al., 2014) can be used. Finally to quickly compute distances, the efficient PQP package can be used, (Larsen et al., 1999). While each of these previous algorithms has been demonstrated to be fast, the running time of the algorithm is expected to be determined mostly by the iterations in Algorithm 1.

4 CONCLUSIONS

In this article we have presented a heuristic to solve the designer's problem of determining the maximum volume object that can be path planned from a given start configuration to a goal configuration. The paper fills an important gap at the intersection of many areas of research as outlined in the introduction. In addition, while developing the algorithms, the authors paid attention to using well-known algorithms and data structures that have been demonstrated to be efficient in practice. Hence, we would expect that an implementation of the algorithms and ideas presented here should not produce computationally intractable problems.

In future work we intend to implement the algorithms presented here as well as address practical issues that arise during their implementation. Future

work will also address trying to find the globally volume maximizing path among all paths from the start to the goal configuration. In addition, we took only strictly better improvements to the volume of the object in Step 10 in Algorithm 2 and Step 6 in Algorithm 1. A further avenue for investigation could be to replace this with a simulated annealing style optimization to allow non-optimal adjustments so that one has the possibility of escaping local maxima.

ACKNOWLEDGEMENTS

This work was carried out at the Wingquist Laboratory VINN Excellence Centre, and is part of the Sustainable Production Initiative and the Production Area of Advance at Chalmers University of Technology. It was supported by the Swedish Governmental Agency for Innovation Systems.

REFERENCES

Björkenstam, S., Segeborn, J., Carlson, J. S., and Bohlin, R. (2012). Assembly verification and geometry design by distance field based shrinking. In *4th CIRP Conference on Assembly Technology and Systems-CATS 2012, University of Michigan, Ann Arbor, USA on May 21-23, 2012*.

Bohlin, R. and Kavraki, L. (2000). Path planning using lazy prm. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 521–528. IEEE.

Carlson, J. S., Spensieri, D., Söderberg, R., Bohlin, R., and Lindkvist, L. (2013). Non-nominal path planning for robust robotic assembly. *Journal of manufacturing systems*, 32(3):429–435.

Cohen-Or, D. and Kaufman, A. (1995). Fundamentals of surface voxelization. *Graphical models and image processing*, 57(6):453–461.

COIN-OR (2014). COmputational INfrastructure for Operations Research. http://www.coin-or.org/.

Folkestad, J. E. and Johnson, R. L. (2002). Integrated rapid prototyping and rapid tooling (irprt). *Integrated Manufacturing Systems*, 13(2):97–103.

Geraerts, R. and Overmars, M. H. (2005). On improving the clearance for robots in high-dimensional configuration spaces. In *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 679–684. IEEE.

Geraerts, R. and Overmars, M. H. (2007). The corridor map method: Real-time high-quality path planning. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 1023–1028. IEEE.

Hermansson, T., Bohlin, R., Carlson, J. S., and Söderberg, R. (2012). Automatic path planning for wiring harness installations (wt). In *4th CIRP Conference on Assembly Technology and Systems-CATS 2012, University of Michigan, Ann Arbor, USA on May 21-23, 2012*.

Ilies, H. T. and Shapiro, V. (1999). The dual of sweep. *Computer-Aided Design*, 31(3):185–201.

Kim, J., Pearce, R. A., and Amato, N. M. (2003). Extracting optimal paths from roadmaps for motion planning. In *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, volume 2, pages 2424–2429. IEEE.

Larsen, E., Gottschalk, S., Lin, M. C., and Manocha, D. (1999). Fast proximity queries with swept sphere volumes. Technical report, Technical Report TR99-018, Department of Computer Science, University of North Carolina.

LaValle, S. and Kuffner Jr, J. (2001). Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400.

Nüchter, A., Elseberg, J., Schneider, P., and Paulus, D. (2010). Linearization of rotations for globally consistent n-scan matching. In *IEEE International Conference on Robotics and Automation (ICRA)*, page 7.

Shellshear, E., Tafuri, S., and Carlson, J. (2014). A multithreaded algorithm for computing the largest non-colliding moving geometry. *Computer-Aided Design*, 49(0):1 – 7.

Spensieri, D., Bohlin, R., and Carlson, J. S. (2013). Coordination of robot paths for cycle time minimization. In *CASE*, pages 522–527.

Spensieri, D., Carlson, J. S., Bohlin, R., and Söderberg, R. (2008). Integrating assembly design, sequence optimization, and advanced path planning. *ASME Conference Proceedings*, (43253):73–81.

Vanderhyde, J. and Szymczak, A. (2008). Topological simplification of isosurfaces in volumetric data using octrees. *Graphical Models*, 70(1):16–31.

Zachmann, G. et al. (2000). *Virtual Reality in Assembly Simulation: Collision Detection, Simulation Algorithms, and Interaction Techniques*. Fraunhofer-IRB-Verlag.

Zheng, L., Cho, Y.-K., Liu, X., and Wang, W. (2011). Cvt-based 2d motion planning with maximal clearance. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 2281–2287. IEEE.