Computation of Collision Distance and Gradient using an Automatic Sphere Approximation of the Robot Model with Bounded Error

Andreas Völz and Knut Graichen
Institute of Measurement, Control and Microtechnology, Ulm University, Ulm, Germany
Email: {andreas.voelz, knut.graichen}@uni-ulm.de

Abstract

This paper presents a method for computing the distance to collision and its gradient, which is a time-critical part of optimization-based motion planners like CHOMP. The computation is based on a sphere approximation of the robot model and an Euclidean distance transformation of the voxelized environment. As speed and accuracy depend on the number of spheres, an algorithm is proposed for error-bounded approximation of a simplified robot model.

1 Introduction

Motion planning for robots is the task of finding a path between a start configuration $\underline{x}_{\mathrm{S}}$ and a goal configuration $\underline{x}_{\mathrm{G}}$ so that the robot does not collide with itself or with obstacles in the environment. Typical sampling-based planners like probabilistic roadmaps or rapidly-exploring random trees focus on feasibility, i.e. quickly finding a valid path, and not on optimality, e.g. finding the shortest collision-free path. However, such planners often result in jerky paths that require subsequent post-processing steps like, for example, shortcutting or smoothing.

A promising alternative is constituted by optimization-based planners that start with an initial trajectory and iteratively improve it during the numerical solution until a feasible path is obtained. Popular approaches are covariant hamiltonian optimization (CHOMP) [19], stochastic trajectory optimization (STOMP) [8] and sequential convex optimization (TrajOpt) [14]. These approaches are based on the solution of optimization problems similar to

$$\min_{\underline{x}_1,\ldots,\underline{x}_N} \quad \sum_{k=1}^{N+1} \left\|\underline{x}_k - \underline{x}_{k-1}\right\|^2 \tag{1a}$$

$$\text{s.t.} \quad \underline{x}_0 = \underline{x}_{\mathrm{start}}\,, \quad \underline{x}_{N+1} = \underline{x}_{\mathrm{goal}} \tag{1b}$$

$$\underline{d}(\underline{x}_k) > \underline{d}_{\min}\,, \quad k = 1,\ldots,N\,, \tag{1c}$$

whereby the path is represented by the configurations $\underline{x}_k$ for $k = 0,\ldots,N+1$. The cost function (1a) minimizes the length of the path and possibly further terms. The boundary conditions (1b) fix the first and the last configuration to start and goal, respectively. Collision avoidance is either realized by a constraint (1c) that requires that the distance to collision $\underline{d}(\underline{x})$ is larger than a minimum distance $\underline{d}_{\min}$ or by penalizing small values of $\underline{d}(\underline{x})$ in the cost function, i.e.

$$\min_{\underline{x}_1,\ldots,\underline{x}_N} \sum_{k=1}^{N+1} \left\|\underline{x}_k - \underline{x}_{k-1}\right\|^2 + w \sum_{k=1}^{N} \max\left\{\underline{0}, \underline{d}_{\min} - \underline{d}(\underline{x}_k)\right\}^2 \tag{2}$$

with the weighting factor w. The numerical solution of these optimization problems involves the computation of the distance to collision $\underline{d}(\underline{x})$ and its derivatives $\nabla\underline{d}(\underline{x})$ with respect to the configuration $\underline{x}$. The evaluation of these functions for many different configurations constitutes the time-critical part of optimization-based planners.

The TrajOpt planner [14] uses routines of the collision detection library to evaluate the distance function. For use in optimization-based motion planning, these routines must compute a signed distance to push the trajectory out of collision, i.e. a positive distance for non-colliding objects and a negative distance for colliding objects. The former is often computed using the Gilbert-Johnson-Keerthi (GJK) algorithm [5], whereas the latter can be calculated by the expanding polytope algorithm (EPA) [17]. However, the computation of signed distances is not fully supported by all collision detection libraries.

In contrast, CHOMP [19] and STOMP [8] use a sphere approximation of the robot model in combination with a distance transformation of the environment to evaluate the distance function, an approach that has already been employed for collision avoidance in [6]. This has the advantage that collision distances are cheap to compute for spheres and can therefore be evaluated for a large number of configurations. However, finding a sphere-based approximation of the 3D-model that has a small number of spheres and a low approximation error is in general a hard task. Moreover, the application to collision avoidance requires a conservative approximation that fully encloses the robot model as otherwise collisions could be missed. Many approaches have been proposed for approximating general 3D-models by spheres, for example [10], [3] or [16]. Often a hierarchical structure is employed, whereby each level uses more spheres to provide a better approximation. Methods to create such sphere-trees are discussed in [13], [7], [12], [2], [1] and [15] to name just a few. Other variants include swept sphere volumes [9] or inner sphere trees [18].

A common approach to speed-up motion planning is the usage of a simplified robot model for collision checking, for example the convex hull of each link, instead of the

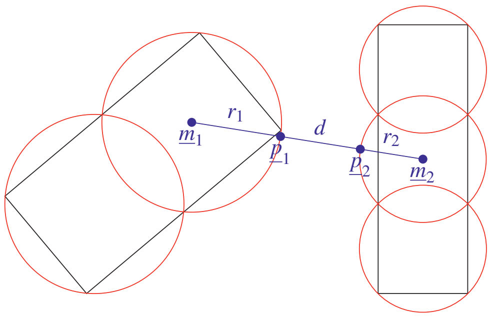


Figure 1 Distance between two links that are approximated by spheres.

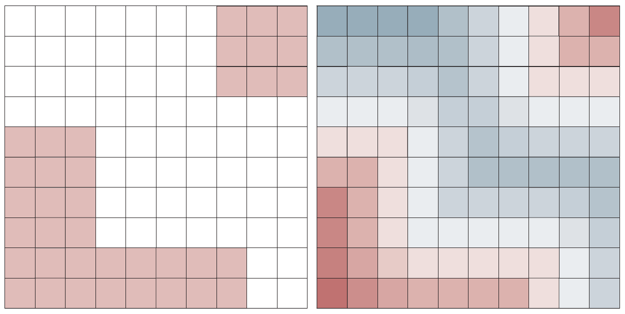


Figure 2 Workspace voxel grid and euclidean distance transformation.

high-resolution model that is used for visualization. The contribution of this paper is built on the assumption that this simplified robot model is made up of spheres, cylinders and boxes. With regard to this special case, geometrical considerations are used to derive algorithms for approximating cylinders and boxes with spheres in such a way that the maximum error is bounded by a predefined threshold. The resulting methods for automatic sphere approximation and computation of collision distance are both easy to implement and fast to compute. The properties of the proposed approach are critically evaluated for a single-arm and a dual-arm setup with respect to self-collisions and obstacle-collisions in randomly generated environments.

2 Collision Distance and Gradient

The robot model is approximated by spheres $S = (\underline{m}, r)$ with center $\underline{m}$ and radius r. Depending on the configuration $\underline{x}$, the sphere centers are transformed according to the forward kinematics and the distance between two spheres $S_1 = (\underline{m}_1, r_1)$ and $S_2 = (\underline{m}_2, r_2)$ is given by (cf. **Figure 1**)

$$d_S(\underline{x}; S_1, S_2) = \|\underline{m}_1(\underline{x}) - \underline{m}_2(\underline{x})\| - r_1 - r_2 \,. \qquad (3)$$

The partial derivatives with respect to $\underline{x}$ are computed by

$$\nabla d_S(\underline{x}; S_1, S_2) = \frac{\underline{m}_1(\underline{x}) - \underline{m}_2(\underline{x})}{\|\underline{m}_1(\underline{x}) - \underline{m}_2(\underline{x})\|} \left(J_{m_1}(\underline{x}) - J_{m_2}(\underline{x})\right) \qquad (4)$$

with the normalized direction vector and the Jacobians of the positions $\underline{m}_1$ and $\underline{m}_2$. Given two robot links $L_1 = \{S^1_1, \ldots, S^1_{n_1}\}$ and $L_2 = \{S^2_1, \ldots, S^2_{n_2}\}$ that are approximated by n_1 and n_2 spheres, respectively, as well as bounding spheres S^1_B and S^2_B for each link, the distance between L_1 and L_2 is given by

$$d_L(\underline{x}; L_1, L_2) = \min_{\substack{i=1,\ldots,n_1 \\ j=1,\ldots,n_2}} d_S(\underline{x}; S^1_i, S^2_j) \,, \qquad (5)$$

i.e. the minimal distance between all pairs of spheres. Since the distance and the contact points are only required if $d_L(\underline{x}; L_1, L_2) \le d_{\min}$, many computations can be skipped if $d_S(\underline{x}, S^1_B, S^2_B) > d_{\min}$ for the bounding spheres.
The environment is approximated by a three-dimensional voxel grid and its Euclidean distance transformation (EDT) that stores for each voxel the distance to the closest obstacle. It can be efficiently computed using the algorithm from [4], whose runtime does only depend on the number of voxels and not on the number of obstacles. An example of a two-dimensional voxel grid with several blocked voxels and its distance transformation is shown in **Figure 2**. The distance between a sphere $S = (\underline{m}, r)$ and obstacles is given by

$$d_{S,\text{edt}}(\underline{x}; S) = \text{edt}(\underline{m}(\underline{x})) - r \,, \qquad (6)$$

whereby $\text{edt}(\underline{m}(\underline{x}))$ evaluates the distance map at the position $\underline{m}(\underline{x})$. The gradient of the EDT can either be computed using finite differences or analytically in case that a trilinear interpolation is used to evaluate $\text{edt}(\underline{m}(\underline{x}))$. The partial derivatives of (6) with respect to $\underline{x}$ are then given using the gradient and the robot Jacobian. For a robot link $L = \{S_1, \ldots, S_n\}$ with n spheres and bounding sphere S_B, the distance between the link and obstacles is approximated by

$$d_{L,\text{edt}}(\underline{x}; L) = \min_{i=1,\ldots,n} d_{S,\text{edt}}(\underline{x}; S_i) \,. \qquad (7)$$

Analogous to the self-collision distance, the computation of (7) can be skipped if $d_{S,\text{edt}}(\underline{x}; S_B) > d_{\min}$ holds for the bounding sphere of the link L.
Finally, the robot model with n_L links $L_1, \ldots, L_{n_L}$ is considered. The collision distance for a link L_i $(i = 1, \ldots, n_L)$ is the minimum of the distance to other links (5) and the distance to obstacles (7), that is

$$d(\underline{x}; L_i) = \min\left\{ d_{L,\text{edt}}(\underline{x}; L_i),\, \min_{j \in \mathscr{I}_i} d_L(\underline{x}; L_i, L_j) \right\} , \qquad (8)$$

whereby the index set $\mathscr{I}_i$ identifies the links that are not allowed to collide with the link L_i. For example, the adjacent links L_{i-1} and L_{i+1} are not part of the set $\mathscr{I}_i$. Using (8), the collision distance constraint (1c) is defined as

$$\underline{d}(\underline{x}) = \begin{bmatrix} d(\underline{x}; L_1) \\ \vdots \\ d(\underline{x}; L_{n_L}) \end{bmatrix} \ge \begin{bmatrix} d_{\min} \\ \vdots \\ d_{\min} \end{bmatrix} = \underline{d}_{\min} \,. \qquad (9)$$

Compared to an exact distance computation, the sphere approximation and the discretized voxel grid involve an error ε. Given a maximum deviation of δ between the original shapes and the spheres and a voxel resolution of Δw, the maximum error is bounded by

$$\varepsilon \le \max\left\{ 2\delta,\, \delta + \sqrt{3}\Delta w \right\} . \qquad (10)$$

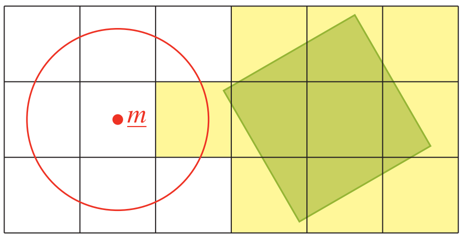


Figure 3 The yellow voxels are blocked by the green obstacle. Although the red circle is not in collision, the approximated distance would be negative.

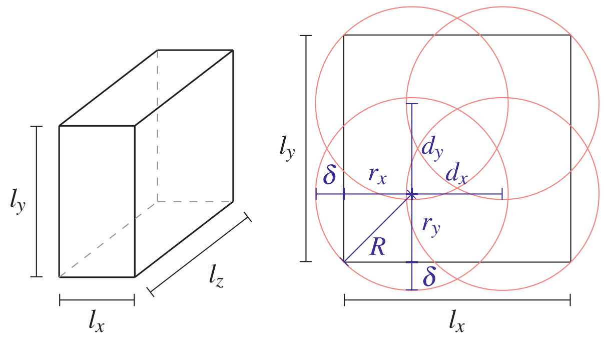


Figure 4 Lengths for approximation of box by spheres.

The error that is introduced by the voxel grid and the Euclidean distance transformation is illustrated in **Figure 3**. This approximation error can lead to infeasible optimization problems (1) although the original motion planning problem is solvable. Therefore, it is of interest to reduce the maximum error of the sphere approximation.

3 Automatic Sphere Approximation

The purpose of the automatic sphere approximation is to compute the necessary number of spheres and their positions so that the geometric shape is completely contained in the sphere set and the error δ is bounded by a predefined threshold $\delta_{\max}$. In this section, algorithms are stated for approximating boxes and cylinders.

3.1 Approximation of 3D-box by spheres

A three-dimensional box with side lengths l_x, l_y and l_z shall be approximated by spheres with radius R so that the maximum error δ is less than a predefined threshold $\delta_{\max}$. Without loss of generality, $l_x \le l_y \le l_z$ is assumed. Then, the maximum sphere radius is bounded by

$$R = \min\left\{ \frac{\sqrt{l_x^2 + l_y^2 + l_z^2}}{2}, \frac{l_x}{2} + \delta_{\max}, \frac{\sqrt{3}\,\delta_{\max}}{\sqrt{3}-1} \right\}. \quad (11)$$

In the first case, the whole box can be approximated by a single sphere without violating the maximum error bound. In the second case, the shortest side l_x limits the sphere radius and multiple spheres may be required for approximating the y- and the z-direction. The third case results from considering a corner point of the box. The distance between the sphere center and the corner point shall be R and the distance between the sphere center and the three faces of the box shall be $R - \delta_{max}$. Then, the equation $R^2 = 3(R - \delta_{\max})^2$ can be solved for R, which leads to the third condition in (11).

The algorithm proceeds along the following steps (compare **Figure 4** for notation). After determining the maximum sphere radius (11), the distances r_x, r_y and r_z between the first sphere center and the three faces of the box are computed. Next, the necessary numbers of spheres N_x, N_y and N_z along the three axes are determined. This step involves rounding up, which indicates that the box can be approximated with the same number of spheres and a smaller error. Therefore, the distances r_x, r_y and r_z, and the sphere radius R are recalculated.

1. In the first case, i.e. $R = \sqrt{l_x^2 + l_y^2 + l_z^2}/2$, the distances and the numbers of spheres are

$$r_x = \frac{l_x}{2} \qquad r_y = \frac{l_y}{2} \qquad r_z = \frac{l_z}{2} \quad (12a)$$
$$N_x = 1 \qquad N_y = 1 \qquad N_z = 1 \quad (12b)$$

   and no recalculation is necessary.

2. The second case, i.e. $R = l_x/2 + \delta_{\max}$, requires a further case analysis. To this end, the distance

$$r_{yz} = \frac{\sqrt{R^2 - (l_x/2)^2}}{\sqrt{2}} \quad (13)$$

   is considered. If $r_{yz} \ge l_y/2$, a single sphere suffices for the y-direction, and the distances and the numbers of spheres are given by

$$r_x = \frac{l_x}{2} \qquad r_y = \frac{l_y}{2} \qquad r_z = \sqrt{R^2 - r_x^2 - r_y^2} \quad (14a)$$
$$N_x = 1 \qquad N_y = 1 \qquad N_z = \left\lceil \frac{l_z}{2r_z} \right\rceil. \quad (14b)$$

   Due to the round-up, the distance

$$r_z = \max\left\{ \frac{l_z}{2N_z}, R - \delta_{\max} \right\} \quad (15)$$

   is recalculated. If $r_{yz} < l_y/2$, multiple spheres are required for the y- and the z-directions. In this case, the distances and the number of spheres are

$$r_x = \frac{l_x}{2} \qquad r_y = r_{yz} \qquad r_z = r_{yz} \quad (16a)$$
$$N_x = 1 \qquad N_y = \left\lceil \frac{l_y}{2r_y} \right\rceil \qquad N_z = \left\lceil \frac{l_z}{2r_z} \right\rceil \quad (16b)$$

   and the recalculation is done according to

$$r_y = r_z = \max\left\{ \frac{l_y}{2N_y}, \frac{l_z}{2N_z} \right\}. \quad (17)$$

3. In the third case, i.e. $R = \frac{\sqrt{3}}{\sqrt{3}-1}\delta_{\max}$, the distances and the number of spheres are given by

$$r_x = R - \delta_{\max} \quad r_y = R - \delta_{\max} \quad r_z = R - \delta_{\max} \quad (18a)$$
$$N_x = \left\lceil \frac{l_x}{2r_x} \right\rceil \quad N_x = \left\lceil \frac{l_y}{2r_y} \right\rceil \quad N_z = \left\lceil \frac{l_z}{2r_z} \right\rceil. \quad (18b)$$

Here, the recalculation is necessary for all distances

$$r_x = r_y = r_z = \max\left\{\frac{l_x}{2N_x}, \frac{l_y}{2N_y}, \frac{l_z}{2N_z}\right\}. \quad (19)$$

Once the necessary numbers of spheres and the distances between the first sphere center and the faces of the box are determined, the final sphere radius is recalculated as

$$R = \sqrt{r_x^2 + r_y^2 + r_z^2}. \quad (20)$$

For all directions that require more than one sphere, the distances between two sphere centers are computed by

$$d_x = \frac{l_x - 2r_x}{N_x - 1} \qquad d_y = \frac{l_y - 2r_y}{N_y - 1} \qquad d_z = \frac{l_z - 2r_z}{N_z - 1}. \quad (21)$$

Finally, the positions of all sphere centers are given by

$$\underline{m}_{i,j,k} = \begin{bmatrix} r_x + i\,d_x - l_x/2 \\ r_y + j\,d_y - l_y/2 \\ r_z + k\,d_z - l_z/2 \end{bmatrix}, \quad \begin{array}{l} i = 1,\dots,N_x \\ j = 1,\dots,N_y, \\ k = 1,\dots,N_z \end{array} \quad (22)$$

whereby the center of the box is used as reference frame.

3.2 Approximation of cylinder by spheres

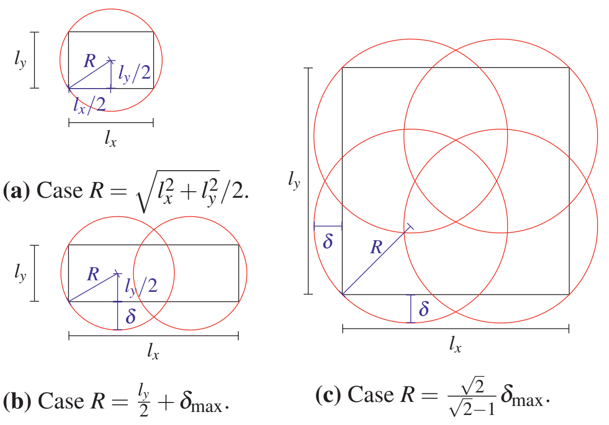
A cylinder with length l_{cyl} and radius r_{cyl} shall be approximated by spheres with radius R so that the maximum error δ is less than a predefined threshold $\delta_{\max}$. To this end, the cylinder is considered from two perspectives. First, the rectangle with lengths l_{cyl} and $2r_{\text{cyl}}$ is approximated by circles with radius R so that the maximum error δ_l is below the threshold $\delta_{l,\max}$. If a single circle suffices for the circular side of the cylinder, i.e. $R > r_{\text{cyl}}$, the approximation is done. Otherwise, the circle with radius r_{cyl} is approximated by smaller circles with radius $\sqrt{R^2 - r_l^2}$ so that the maximum error δ_r is below the bound $\delta_{r,\max} = \delta_{\max} - \delta_l$, whereby r_l is the distance between the first circle and the circular side of the cylinder. Due to the recalculation of the circle radius after rounding up, $\delta_{l,\max} = \delta_{\max}$ could be chosen. However, it makes sense to use a threshold $\delta_{l,\max} < \delta_{\max}$ to ensure that $\delta_{r,\max} > 0$ and $\delta = \delta_l + \delta_r < \delta_{\max}$.

3.2.1 Approximation of rectangle by circles

A rectangle with lengths l_x and l_y shall be approximated by circles with radius R so that the maximum error δ is less than a predefined threshold $\delta_{\max}$. The approximation is done similarly to the case of a three-dimensional box shown in the last section. Assuming $l_x \leq l_y$, the maximum circle radius is bounded by

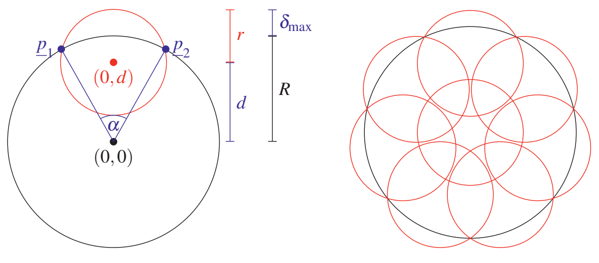
$$R = \min\left\{\frac{\sqrt{l_x^2 + l_y^2}}{2}, \frac{l_x}{2} + \delta_{\max}, \frac{\sqrt{2}\,\delta_{\max}}{\sqrt{2} - 1}\right\}, \quad (23)$$

which corresponds to a single circle, multiple circles along the y-direction, or multiple circles along both directions. These cases are illustrated in **Figure 5**. The algorithm first computes the distances r_x and r_y between the first circle center and the sides of the rectangle and afterwards the necessary numbers of circles N_x and N_y along the axes. As this step involves rounding up, the distances and the circle radius are recalculated. Finally, the distances between two circle centers d_x, d_y and the circle positions $m_{i,j}$ for $i = 1,\dots,N_x$ and $j = 1,\dots,N_y$ are determined.



(a) Case $R = \sqrt{l_x^2 + l_y^2}/2$. **(b)** Case $R = \frac{l_y}{2} + \delta_{\max}$. **(c)** Case $R = \frac{\sqrt{2}}{\sqrt{2}-1}\delta_{\max}$.

Figure 5 Approximation of rectangle by circles.



(a) Lengths and coordinates **(b)** Final approximation

Figure 6 Approximation of large circle by small circles.

3.2.2 Approximation of big circle by small circles

A large circle with radius R shall be approximated by smaller circles with radius r so that the maximum error δ is less than a predefined threshold $\delta_{\max}$. The main idea is sketched in **Figure 6**. The small circle is shifted from the center to the boundary of the big circle until the overlap is exactly $\delta_{\max}$. Using the intersection points $\underline{p}_{1,2}$, the central angle α and the necessary number of circles N can be computed.

Assuming that the center of the large circle is at position $(0,0)$, the small circle is shifted along the y-axis up to the position $(0,d)$ with

$$d = R + \min\{0, \delta_{\max} - r\}. \quad (24)$$

Afterwards, the two intersection points $\underline{p}_{1,2} = (\pm p_x, p_y)$ of the circles are calculated. Combined with the origin $(0,0)$, they describe a circular segment with the angle

$$\alpha = 2\arccos\left(\frac{p_y}{R}\right), \quad (25)$$

which leads to the necessary number of circles

$$N = \left\lceil \frac{2\pi}{\alpha} \right\rceil. \quad (26)$$

The rounding up implies that it is possible to approximate the big circle with the same number of small circles and a smaller error. Therefore, the angle

$$\alpha = \frac{2\pi}{N} \tag{27}$$

and the two intersection points

$$\underline{p}_{1,2} = \begin{bmatrix} \pm R\sin(\alpha) \\ R\cos(\alpha) \end{bmatrix} \tag{28}$$

are recalculated. This step shifts the small circle nearer to the origin, that is

$$d = p_y - \sqrt{R^2 - p_x^2}\,. \tag{29}$$

Finally, the positions of the circle centers are given by

$$\underline{m}_i = \begin{bmatrix} d\sin(i\alpha) \\ d\cos(i\alpha) \end{bmatrix}, \quad i = 1, \ldots, N\,. \tag{30}$$

In case that $d > r$, a region remains inside the large circle that is not covered by the smaller circles. This region can be considered as another circle with radius

$$\hat{R} = \frac{d}{\cos(\alpha/2)} - r \tag{31}$$

for which the approximation can be repeated recursively. **Figure 6b** shows an example for the approximation with one recursion.

4 Evaluation Results

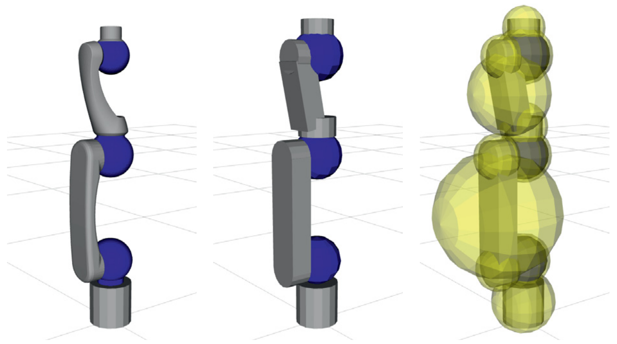
This section evaluates the proposed approach for computing the collision distance and the automatic sphere approximation. To this end, the model of a Schunk LWA 4p with six degrees of freedom is used in a single-arm and a dual-arm setup.

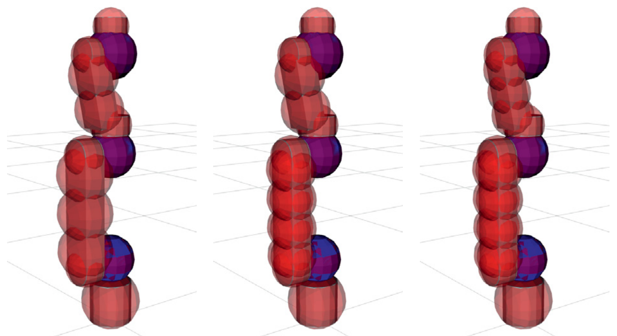
4.1 Sphere approximation

First of all, the generation of the sphere set is evaluated. **Figure 7** shows the high-resolution model of the Schunk LWA 4P as well as the simplified collision model that is made up of boxes, cylinders and spheres. Furthermore, the bounding spheres and the sphere approximation are visualized for maximum errors between 0.06 m and 0.01 m. **Table 1** lists the error bound $\delta_{\max}$, the resulting approximation error $\delta \leq \delta_{\max}$, the number of spheres and the computation time. As expected, smaller error bounds require more spheres in the approximation. The computation time is largely independent of the number of spheres and well below one millisecond, which allows for example to approximate grasped objects on-the-fly during motion planning. In the case of a dual-arm setup, twice the number of spheres is obtained.

4.2 Distance to self-collision

Next, the distance to self-collisions is evaluated in an empty environment. The flexible collision library (FCL)


(a) High-resolution mesh model **(b)** Simplified collision model **(c)** Model with bounding spheres


(d) Maximum error $\delta_{\max} = 0.06\,\mathrm{m}$ **(e)** Maximum error $\delta_{\max} = 0.05\,\mathrm{m}$ **(f)** Maximum error $\delta_{\max} = 0.04\,\mathrm{m}$

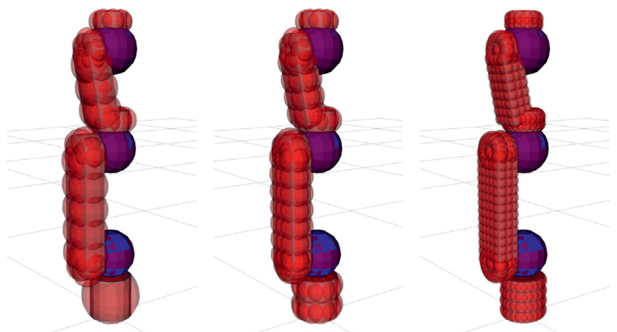

(g) Maximum error $\delta_{\max} = 0.03\,\mathrm{m}$ **(h)** Maximum error $\delta_{\max} = 0.02\,\mathrm{m}$ **(i)** Maximum error $\delta_{\max} = 0.01\,\mathrm{m}$

Figure 7 Automatic sphere approximation for Schunk LWA 4P and different values of the maximum error $\delta_{\max}$.

Table 1 Computation time and number of spheres for automatic sphere approximation of the Schunk LWA 4P arm (cf. Fig. 7).

Bound $\delta_{\max}$	Error δ	Spheres N_S	Comp. time
0.060 m	0.055 m	14	0.176 ms
0.050 m	0.042 m	19	0.128 ms
0.040 m	0.039 m	20	0.118 ms
0.030 m	0.029 m	54	0.148 ms
0.020 m	0.020 m	120	0.174 ms
0.010 m	0.010 m	641	0.179 ms

Table 2 Computation time and false positive rate for distance to self-collisions in the single-arm setup.

Bound $\delta_{\max}$	Mean time	Max time	FP rate
0.06 m	0.012 ms	0.048 ms	1.30 %
0.05 m	0.013 ms	0.032 ms	1.30 %
0.04 m	0.013 ms	0.033 ms	1.09 %
0.03 m	0.015 ms	0.048 ms	0.99 %
0.02 m	0.019 ms	0.072 ms	0.75 %
0.01 m	0.036 ms	0.101 ms	0.42 %
FCL coll. check	0.021 ms	0.050 ms	

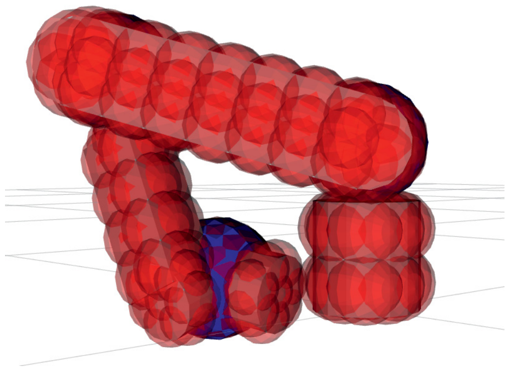


Figure 8 Example of a collision-free configuration with negative collision distance for the bound $\delta_{\max} = 0.02\,\text{m}$.

[11] is used for validation in the following. Note, however, that FCL is used only for collision detection on the simplified robot model and not for computation of the signed distance. Evaluation is performed by computing the collision distance for 100000 random configurations and checking each for collision with FCL. If the approximated collision distance is negative for a collision-free configuration, the sample is counted as false positive. Conversely, a positive collision distance for a colliding configuration would be counted as false negative and indicate an error in the algorithm or the implementation. See **Figure 8** for an example of a false positive configuration.

Table 2 lists the error bound $\delta_{\max}$, the mean and maximum computation times as well as the fraction of false positives for the single-arm setup. In addition, the mean and maximum time for collision checking with FCL is stated. As desired, the false positive rate decreases for lower error bounds. In return, the required mean and maximum computation times increase. However, the time increases significantly slower than the number of spheres (cf. Table 1), because in many cases the bounding sphere checks suffice and the distance between all pairs of spheres is rarely calculated. Except for the lowest bound, the collision distance is computed in a similar amount of time as FCL requires for collision checking.

In addition, the collision distance is evaluated for a dual-arm setup with two Schunk LWA 4P shown in **Figure 9**, whereby the arms are placed in a relative distance of 0.75 m. The mean and maximum computation times as well as the false positives are listed in **Table 3** for different values of the bound $\delta_{\max}$. Compared to the single-arm setup, the mean computation time is roughly three times higher, which accounts for the computation of collision distances between the two arms. Besides this, also the maximum computation times and the number of false positives are higher. Even for the lowest bound of $\delta_{\max} = 0.01\,\text{m}$, the sphere-based computation is faster than collision checking with FCL, which highlights the speed advantage of the approximation approach.

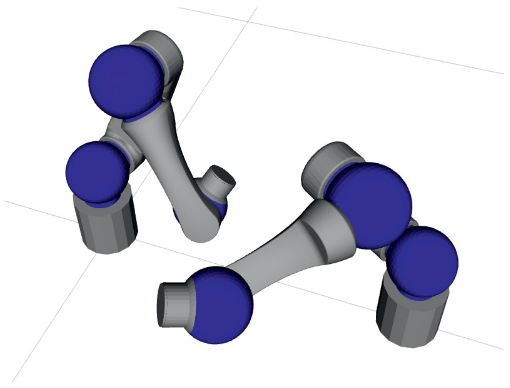


Figure 9 Dual-arm setup with two Schunk LWA 4P arms.

Table 3 Computation time and false positive rate for distance to self-collisions in the dual-arm setup.

Bound $\delta_{\max}$	Mean time	Max time	FP rate
0.06 m	0.036 ms	0.190 ms	3.00 %
0.05 m	0.037 ms	0.190 ms	2.91 %
0.04 m	0.037 ms	0.194 ms	2.50 %
0.03 m	0.041 ms	0.229 ms	2.21 %
0.02 m	0.048 ms	0.259 ms	1.70 %
0.01 m	0.085 ms	0.485 ms	0.96 %
FCL coll. check	0.110 ms	0.544 ms	

4.3 Distance to obstacles

The distance to obstacles is evaluated in 1000 randomly generated environments, whereby each planning scene is created by placing 10 boxes with side lengths between 0.05 m and 0.5 m at random poses. The collision distance is computed for 100 random configurations in each environment and collision checking with FCL is used as reference again.

Table 4 Computation time and false positive rate for distance to obstacles and varying workspace resolution.

Resolution Δw	Mean time	Max time	FP rate
0.05 m	0.019 ms	0.080 ms	23.41 %
0.04 m	0.018 ms	0.082 ms	18.35 %
0.03 m	0.018 ms	0.098 ms	13.82 %
0.02 m	0.018 ms	0.107 ms	9.51 %
0.01 m	0.017 ms	0.061 ms	5.26 %
FCL coll. check	0.021 ms	0.105 ms	

Table 5 Computation time and false positive rate for distance to obstacles in case collision checking is done against the voxelized obstacles with varying resolution.

Resolution Δw	Mean time	Max time	FP rate
0.05 m	0.029 ms	0.127 ms	14.36 %
0.04 m	0.030 ms	0.164 ms	11.11 %
0.03 m	0.031 ms	0.229 ms	8.17 %
0.02 m	0.039 ms	0.169 ms	5.64 %

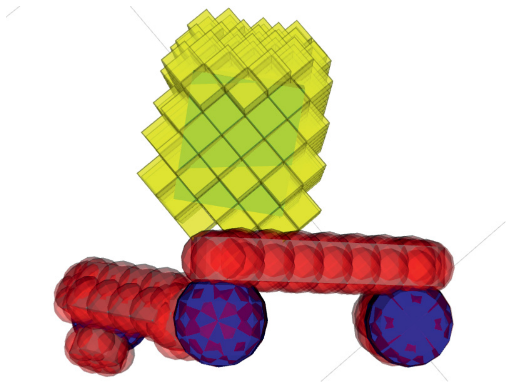


Figure 10 Example of a collision-free configuration with negative collision distance for the bound $\delta_{\max} = 0.02\,\mathrm{m}$ and a workspace resolution of $\Delta w = 0.05\,\mathrm{m}$ (spheres in red, obstacle in green, occupied voxels in yellow).

The evaluation results are listed in **Table 4** for different values of the workspace resolution Δw, i.e. the size of the voxels, and a maximum error for the sphere approximation of $\delta_{\max} = 0.02\,\mathrm{m}$. The computation time is independent of the workspace resolution, because it suffices to create the distance field once per environment and this initialization time is not included here. The number of false positives, i.e. configurations that are not detected as collision-free, is much higher compared to the self-collision evaluation, which is due to the errors introduced by voxelization of the obstacles and the Euclidean distance transformation. Therefore, the false positive rate can be reduced for smaller voxel sizes. **Figure 10** shows an example of a false positive configuration that illustrates how the workspace resolution affects the accuracy. To further investigate the influence of the workspace resolution, collision checking with FCL is performed for the case of voxelized obstacles and compared to the approximated collision distance. **Table 5** lists these results for varying voxel size Δw. As expected, the false positive rate is considerably lower than in Table 4. The remaining errors are mainly caused by the Euclidean distance transformation as self-collisions are less frequent (cf. Table 2).

The disadvantage of smaller voxel sizes is the increased memory consumption of the distance field and the required time to voxelize the obstacles and to compute the Euclidean distance transformation. **Table 6** shows the number of voxels in the three-dimensional grid and the average computation times for voxelization and distance transformation. As one might expect, the total computation time is roughly proportional to the number of voxels. Therefore, the selection of the workspace resolution depends on the time requirements of the application. The more time is available the smaller Δw can be chosen and the better is the accuracy of the collision distance. Less time for initialization is necessary if the environmental model is directly given as voxel grid, e.g. if point cloud data from sensors is mapped to an octree-based data structure, as in this case no voxelization is required. Otherwise, algorithms could be used that do not recompute the whole distance map and instead perform local updates to the EDT for moving objects, see for example [6]. In addition, it is possible to precompute high-resolution distance fields for known objects in the environment and exploit the property that distance fields are compositional with respect to the min-operation [19]. Using such techniques, it should be possible to reduce the number of false positives while keeping the initialization time low.

Table 6 Mean computation time for voxelization of obstacles and Euclidean distance transformation.

Resolution Δw	Voxels	Time vox.	Time EDT
0.05 m	64000	0.006 s	0.011 s
0.04 m	125000	0.008 s	0.019 s
0.03 m	287496	0.016 s	0.045 s
0.02 m	1000000	0.047 s	0.154 s
0.01 m	8000000	0.326 s	1.295 s

5 Conclusions

This paper presented a method for computing the distance to collision and its gradient based on a sphere approximation of the robot model and a distance transformation of the environment. The proposed algorithm for automatic sphere approximation is built on the assumption that a simplified collision model is available consisting only of spheres, cylinders, and boxes. Given a maximum allowed error, the algorithms compute the sphere radius, the necessary number of spheres as well as their positions. The procedure is easy to implement and results in fast computation times both for generating the sphere set and for computing the collision distance. Due to the simple structure and the remaining approximation error, the approach is not intended to replace accurate collision detection algorithms, but to provide fast approximate distances that can be used in an optimization-based motion planner. Compared to a handmade sphere approximation, the proposed approach has the advantage of a single parameter for tuning speed and accuracy. Furthermore, it can be used for on-the-fly approximation of grasped objects.

The evaluation showed that the main limitation of the approach is not the sphere approximation but the error that is introduced by voxelization of obstacles and the Euclidean distance transformation. Reducing this error requires smaller voxel sizes that in turn lead to increased initialization times. Therefore, depending on the application, the usage of exact algorithms implemented in state-of-the-art collision detection libraries might be the better choice.

While the approach could be improved by using a hierarchical sphere model or by integration of approximation algorithms for general 3D-models, the focus of ongoing research lies on the application of the fast distance computation as part of optimization-based motion planners for robot arms.

6 Literature

[1] Antonio Benitez, Maria del Carmen Ramírez, and Daniel Vallejo. Collision detection using sphere-tree construction. In *Proc. Int. Conf. on Electronics, Communications and Computers (CONIELECOMP)*, pages 286–291, 2005.

[2] Gareth Bradshaw and Carol O'Sullivan. Adaptive medial-axis approximation for sphere-tree construction. *ACM Transactions on Graphics (TOG)*, 23(1): 1–26, 2004.

[3] Angel P Del Pobil, Miguel A Serna, and Juan Llovet. A new representation for collision avoidance and detection. In *Proc. Int. Conf. on Robotics and Automation (ICRA)*, pages 246–251, 1992.

[4] Pedro Felzenszwalb and Daniel Huttenlocher. Distance transforms of sampled functions. Technical report, Cornell University, 2004.

[5] Elmer G Gilbert, Daniel W Johnson, and S Sathiya Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal on Robotics and Automation*, 4 (2):193–203, 1988.

[6] Michael Greenspan and Nestor Burtnyk. Obstacle count independent real-time collision avoidance. In *Proc. Int. Conf. on Robotics and Automation (ICRA)*, volume 2, pages 1073–1080, 1996.

[7] Philip M Hubbard. Approximating polyhedra with spheres for time-critical collision detection. *ACM Transactions on Graphics (TOG)*, 15(3):179–210, 1996.

[8] Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal. STOMP: Stochastic trajectory optimization for motion planning. In *Proc. Int. Conf. on Robotics and Automation (ICRA)*, pages 4569–4574, 2011.

[9] Eric Larsen, Stefan Gottschalk, Ming C Lin, and Dinesh Manocha. Fast proximity queries with swept sphere volumes. Technical report, TR99-018, Department of Computer Science, University of North Carolina, 1999.

[10] Joseph O'Rourke and Norman Badler. Decomposition of three-dimensional objects into spheres. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (3):295–305, 1979.

[11] Jia Pan, Sachin Chitta, and Dinesh Manocha. FCL: A general purpose library for collision and proximity queries. In *Proc. Int. Conf. on Robotics and Automation (ICRA)*, pages 3859–3866, 2012.

[12] Joe Pitt-Francis and Roy Featherstone. Automatic generation of sphere hierarchies from CAD data. In *Proc. Int. Conf. on Robotics and Automation (ICRA)*, volume 1, pages 324–329, 1998.

[13] Sean Quinlan. Efficient distance computation between non-convex objects. In *Proc. Int. Conf. on Robotics and Automation*, pages 3324–3329, 1994.

[14] John Schulman, Yan Duan, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, Jia Pan, Sachin Patil, Ken Goldberg, and Pieter Abbeel. Motion planning with sequential convex optimization and convex collision checking. *International Journal of Robotics Research*, 33(9):1251–1270, 2014.

[15] Klaus Steinbach, James Kuffner, Tamim Asfour, and Ruediger Dillmann. Efficient collision and self-collision detection for humanoids based on sphere trees hierarchies. In *Proc. Int. Conf. on Humanoid Robots (Humanoids)*, pages 560–566, 2006.

[16] Svetlana Stolpner, Paul Kry, and Kaleem Siddiqi. Medial spheres for shape approximation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(6):1234–1240, 2012.

[17] Gino Van Den Bergen. Proximity queries and penetration depth computation on 3d game objects. In *Game developers conference*, volume 170, 2001.

[18] Rene Weller and Gabriel Zachmann. Inner sphere trees for proximity and penetration queries. In *Robotics: science and systems*, volume 2, 2009.

[19] Matt Zucker, Nathan Ratliff, Anca D Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M Dellin, J Andrew Bagnell, and Siddhartha S Srinivasa. CHOMP: Covariant Hamiltonian optimization for motion planning. *International Journal of Robotics Research*, 32(9-10):1164–1193, 2013.