2016 IEEE International Conference on Robotics and Automation (ICRA)
Stockholm, Sweden, May 16-21, 2016

Path Planning for Robotic Manipulators Using Expanded Bubbles of Free C-Space

Adnan Ademovic[1] and Bakir Lacevic[1]

Abstract— **This paper presents a novel method for using volumetric information in the C-space for generating collision-free paths for robotic manipulators in the presence of obstacles. The method is based on the RRT paradigm, i.e., incrementally building trees in C-space in order to connect the initial configuration to the final one. Bubbles of free C-space are used within a carefully modified extend procedure to add lengthy edges to the tree. In particular, we propose an expanded version of the bubble, which enables even more efficient exploration. The method is validated and compared to other planning algorithms within a simulation study that includes several types of manipulators and obstacle scenarios.**

I. INTRODUCTION

The majority of sampling-based planning algorithms, such as probabilistic roadmaps (PRM) [1], [2] or rapidly exploring random trees (RRT) [3], [4], [5] rely on sampling the C-space of the robot using random configurations and building a carefully structured graph by connecting those configurations. Validity of local paths (graph edges) is usually established by invoking collision checking routines that provide Boolean information whether a specific point in the C-space implies the collision of the robot with a set of obstacles in the workspace. The computed graph, which captures the connectivity of collision-free C-space, can be easily used to extract the collision-free path that connects the initial configuration $\mathbf{q}_{start}$, to the final configuration $\mathbf{q}_{goal}$.

Both RRT and PRM are practically one-dimensional since they represent sets of one-dimensional edges (local paths). As such, they do not provide any volumetric information about the free C-space. A simple way to do so is by incorporating so-called *bubbles of free C-space* [6]. The bubble is a collision-free local volume around a given configuration. They were originally used to "inflate" the existing, one-dimensional path for the purpose of its modification with respect to possible changes in the environment. The bubble itself is computed using a minimum distance between the robot and the set of obstacles in the workspace. This distance may be observed as a "safety" margin that provides the rough information about how far the robot can move from the specific configuration without colliding with any of the obstacles. When this margin gets translated into C-space, a simple Boolean information whether the current configuration is collision-free or not is enriched by a guaranteed collision-free local volume. A price to pay is a necessity for a distance/proximity query, which usually takes little more time than a collision check, at least when routines from standard libraries, such as PQP or FCL are used (see e.g., [7], [8]). Nevertheless, there have been several attempts to use bubbles for generating the path. In [9] and [10], bubbles are used within the RRT algorithm to adapt the length of the edges that are being added to the tree. In [11], [12], the bubbles are chained within a deterministic A^*-based search algorithm that computes the collision-free tubes of free C-space for robotic manipulators. In [13], a learning approach is presented to improve the performance of bubble-based planning. The common denominator of the above mentioned algorithms is a considerably increased execution time, when compared to classical RRT. The main reason behind this is most likely related to frequent usage of proximity queries. A related approach, at least regarding the use of "bubble" terminology, can be found in In [14]. Here, a planning algorithm uses volumetric information, which is stored and maintained within the tree of n-dimensional balls in C-space. However, these volumes do not necessarily represent the exact collision-free regions, but serve to bias the sampling process. A similar method is proposed in [15] to substantially reduce the number of collision checks.

In this paper, we present another bubble-based algorithm for path planning. In particular, we are interested in using the bubbles within the RRT context to facilitate the procedures of extending and connecting the trees. The main goal is to speed up the algorithm by more efficient exploration of the free C-space. This was achieved with the help of two things. First, we present a modified method for extending and connecting the trees. The method uses (expanded) bubbles to establish collision-free local paths in an efficient way that apparently saves time. Second, we propose the *expanded bubble of free C-space* that can be substantially larger than the original bubble. The expanded bubble can be computed with a slight (or even without) additional cost with respect to the original bubble. The alternative approach to using distance queries to gain more information about the free C-space can be found in our forthcoming paper [16].

The remainder of the paper is organized as follows. Section II provides a brief background on bubbles of free C-space followed by the definition of the expanded bubble of free C-space. Section III presents the RRT-based algorithm that uses expanded bubbles of free C-space for path planning. Results of the simulation study are presented in Section IV, while Section V concludes the paper.

II. EXPANDED BUBBLES OF FREE C-SPACE

Bubbles of free C-space were introduced in [6]. It is a straightforward way to compute a collision-free region

Authors are with the Faculty of Electrical Engineering, University of Sarajevo, 71000 Sarajevo, Bosnia and Herzegovina. {aademovic1,blacevic}@etf.unsa.ba

around a given configuration using a proximity query from the workspace. The bubble $\mathcal{B}(\mathbf{q}, d_c)$ at a configuration $\mathbf{q}$ is computed using minimum distance d_c between the robot and the set of workspace obstacles. For the robotic manipulator with n revolute joints, a bubble $\mathcal{B}(\mathbf{q}, d_c)$ is given by [6]:

$$\mathcal{B}(\mathbf{q}, d_c) = \left\{ \mathbf{y} \;\middle|\; \sum_{i=1}^{n} r_i \left| y_i - q_i \right| \leq d_c \right\}, \qquad (1)$$

where q_i denotes the angle of the i-th joint and r_i represents the radius of the cylinder whose axis is collocated with the axis of the i-th joint and that encloses all the links starting from i-th joint to the end-effector. The expression $\sum_{i=1}^{n} r_i \left| y_i - q_i \right|$ is a conservative upper bound on the displacement of any point on the manipulator when the configuration changes from $\mathbf{q} = (q_1 \; q_2 \; \ldots \; q_n)^T$ to $\mathbf{y} = (y_1 \; y_2 \; \ldots \; y_n)^T$. Moving the robot from configuration $\mathbf{q}$ to an arbitrary configuration $\mathbf{y}$ within a bubble means that no point on the kinematic chain will move more than d_c and thus no collision will occur. From equation (1), it is obvious that bubbles have a diamond shape.

As pointed out in [6], the computation of bubbles is rather conservative. This can result in unnecessary small bubbles. One way to avoid this is to relax the conservative upper bounds that appear in the definition of the bubble. However, such an approach may result in less convenient mathematical representation of the bubble and render its shape nonconvex, which adversely affects the planning purposes [6]. Another approach is to use more information about the distance from the workspace. In this paper, we will incline toward the second approach. In that regard, the following function, called *distance profile* is defined.

Definition 1: Let $\mathbf{p}(s) : [0, L] \rightarrow \mathbb{R}^3$ be the curve in $\mathbb{R}^3$ that represents the wire-model of the robotic manipulator, where L is the robot length, and s is the curve's natural parameter. The *distance profile* $d(s)$ is defined as:

$$d(s) = \rho\left(\mathbf{p}(s), \mathcal{WO}\right), \qquad (2)$$

where $\mathcal{WO}$ is the set of obstacles in the workspace and ρ represents the metric function, i.e., a minimum distance between its two arguments.

In other words, function $d(s)$ assigns each point $\mathbf{p}(s)$ on the kinematic chain a minimum distance to the set of obstacles $\mathcal{WO}$. As such, it represents a generalization of the minimum distance d_c between the robot as a whole and the set of obstacles. Fig. 1 shows an example of the workspace with the corresponding distance profile $d(s)$.

The concept of distance profile can be easily extended from wire models to 3D articulated models. Namely, instead of defining $d(s)$ as a distance from $\mathbf{p}(s)$ to $\mathcal{WO}$, one can consider the distance from $\mathcal{WO}$ to a "slice" of the robot obtained via intersection of the robot with the plane that is perpendicular to its wire model at the point $\mathbf{p}(s)$.

Now, consider a robotic manipulator $\mathcal{R}$ with n links (Fig. 2). Let a_i be the length of the i-th link and let $L_k = \sum_{j=1}^{k} a_j$, where $k = 1, 2, \ldots, n$. For convenience, we declare $L_0 = 0$. Each point $\mathbf{p}(s)$ of the kinematic chain has a minimum distance $d(s)$ to a set of workspace obstacles.

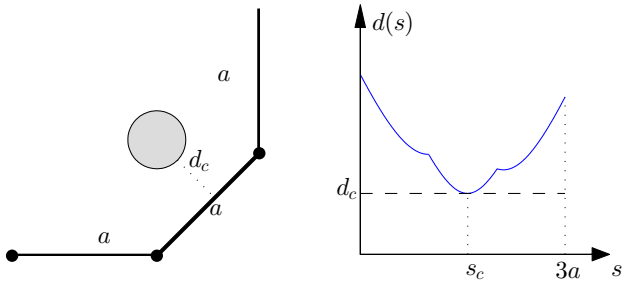

Fig. 1. 3 DOF plannar manipulator with a single obstacle (left) and the corresponding distance profile $d(s)$ (right)

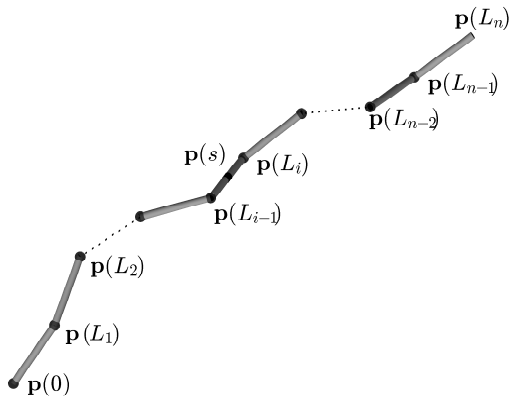

Fig. 2. Illustration of n-DOF robotic manipulator. The positions of link endpoints are represented as functions of the natural parameter.

The concept of an expanded bubble yields from the following condition: no point $\mathbf{p}(s)$ should be displaced more than $d(s)$ when the robot moves from the current configuration $\mathbf{q}$ to an arbitrary configuration $\mathbf{y}$ inside the bubble. For an arbitrary $s \in (L_{i-1}, L_i]$, meaning that point $p(s)$ belongs to i-th link, the following condition can be imposed:

$$\sum_{k=1}^{i} r_{ik}(s) \left| y_k - q_k \right| < d(s). \qquad (3)$$

Here, $r_{ik}(s)$ represents the radius of the cylinder that is coaxial with the k-th joint, which encloses all parts of the robot $\mathcal{R}_k(s)$, starting from k-th joint. The notion $\mathcal{R}_k(s)$ stands for the "incomplete" robot that starts at the point $\mathbf{p}(L_{k-1})$ (location of the k-th joint) and ends at the position $\mathbf{p}(s)$. The index "i" in $r_{ik}(s)$ stands just to indicate that the natural parameter s corresponds to the i-th link.

In i-dimensional subspace of the C-space ($i \leq n$), equation (3) represents the family of diamond-shaped bubbles. Fig. 3 shows an example of the family of bubbles, obtained when inequality (3) is applied to the second link of the 2 DOF planar manipulator. As seen from Fig. 3, the resulting set of points, which satisfies the inequality (3) is not necessarily diamond shaped (inner white region). Nevertheless, it is clearly convex (since the intersection of convex sets is again convex). Furthermore, a slightly smaller, diamond shaped bubble can be inscribed in it (see Fig. 3). This bubble,

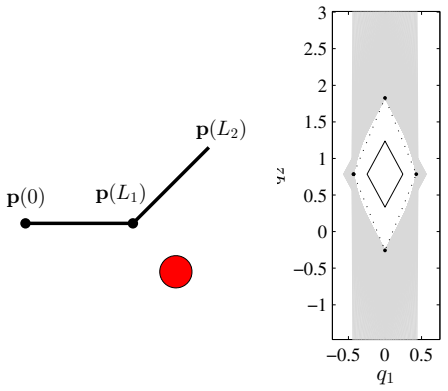

Fig. 3. Left - 2 DOF planar robot with a single obstacle. Right - a family of diamond-shaped bubbles (depicted in gray) that correspond to inequality (3), when parameter s sweeps the interval $(L_1, L_2]$. The original bubble, computed by Equation (1), is also shown.

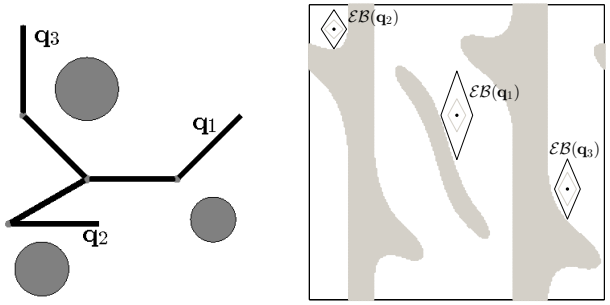

Fig. 4. Three different configurations of a 2 DOF planar manipulator (left). Corresponding expanded bubbles in C-space $(-\pi, \pi]^2$ with obstacles indicated (right). Original bubbles are depicted for comparison.

denoted by $\mathcal{B}_i(\mathbf{q}, s)$, has the following $2i$ vertices:

$$\mathbf{v}_{j,1/2} = q_j \pm \Delta q_{ij}, \quad j = 1, 2, \ldots, i, \tag{4}$$

where Δq_{ij} is computed as:

$$\Delta q_{ij} = \min_{s \in (L_{i-1}, L_i]} \frac{d(s)}{r_{ij}(s)}. \tag{5}$$

Note that $\mathcal{B}_i(\mathbf{q}, s)$ is computed with respect to the i-th link, i.e., interval $s \in (L_{i-1}, L_i]$. To include all the links, it is sufficient to compute all the potential vertices, using (4) and (5), when i goes from 1 to n. Therefore, the final, collision-free region $\mathcal{EB}(\mathbf{q}, d(s))$, to which we refer to as the *expanded bubble of free C-space*, is a diamond shaped polytope with the following $2n$ vertices:

$$\mathbf{v}_{k,1/2} = q_k \pm \Delta q_k, \quad k = 1, 2, \ldots, n, \tag{6}$$

where

$$\Delta q_k = \min_i \Delta q_{ik}. \tag{7}$$

Analytically, $\mathcal{EB}(\mathbf{q}, d(s))$ can be expressed as:

$$\mathcal{EB}(\mathbf{q}, d(s)) = \left\{ \mathbf{y} \;\middle|\; \textstyle\sum_{i=1}^{n} \frac{1}{\Delta q_i} |y_i - q_i| \leq 1 \right\}. \tag{8}$$

Fig. 4 shows several expanded bubbles computed for a 2 DOF planar manipulator at different configurations. Clearly, expanded bubbles can occupy substantially larger volumes of free C-space compared to ordinary bubbles. A price to pay is a necessity to compute the exact distance profile $d(s)$ instead of just computing a minimum distance d_c (which is actually a minimum value of $d(s)$). Note that the expanded bubbles easily accommodate for joint limits, identically as in the case of the original bubbles [6].

It is worth pointing out that any estimate $\hat{d}(s)$ of the distance profile $d(s)$ can be used to compute the expanded bubble. To ensure that the resulting bubble is collision-free, $\hat{d}(s)$ has to be the underestimate of $d(s)$, i.e., $\hat{d}(s) \leq d(s)$, $\forall s \in [0, L]$. When $\hat{d}(s)$ tends to $d(s)$, the resulting bubble $\mathcal{EB}(\mathbf{q}, \hat{d}(s))$ tends to reach the size of $\mathcal{EB}(\mathbf{q}, d(s))$

A simple way to obtain function $\hat{d}(s)$ is to use a piecewise constant function, denoted by $d_N(s)$, that has the following property. Given a partition of the kinematic chain $0 = s_0 < s_1 < s_2 < \ldots < s_{N-1} < s_N = L$, then, in any interval $s \in (s_{k-1}, s_k]$, $k = 1, 2, \ldots, N$, the function $d_N(s)$ takes the following constant value:

$$d_N(s) = \inf_{s \in (s_{k-1}, s_k]} d(s). \tag{9}$$

A trivial partition with $N = 1$ would imply that the distance profile $d(s)$ is a constant function and equals to d_c. Consequently, the corresponding expanded bubble $\mathcal{EB}(\mathbf{q}, d_N(s))$ "collapses" to the original, unexpanded bubble. An intuitive partition with $N = n$ and $s_k = L_k$ implies that $d(s)$ takes constant values for each separate link. Note that this function can be obtained without additional cost when compared to computation of the minimum distance d_c. Namely, computing the minimum distance between the robot and the environment usually relies on computing the distances for each link separately. Simulation results show that even such simple partition implies significant improvements in exploration capabilities and execution time when expanded bubbles are used within a planning algorithm.

III. PLANNING ALGORITHM

The planning algorithm is based on the RRT-Connect planner, while introducing several changes in the way trees are formed. It forms two trees of bubbles τ_a and τ_b, starting from $\mathbf{q}_{init}$ and $\mathbf{q}_{goal}$ respectively. The trees are simultaneously updated until they connect, and a solution is found. Updates are made using Extend and Connect functions cyclically.

At the base of the algorithm is the ConnectPoints function, which receives two C-space points and returns whether or not there is a straight, unobstructed line in C-space connecting those two points. It requires a tree τ and a parent bubble $\mathcal{B}_{par}$ exclusively to add newly formed bubbles to the tree, and makes no other use of them. The key arguments are the starting point $\mathbf{q}_1$ and the goal point $\mathbf{q}_2$ that form the potential local path that needs to be validated. Finally, the function uses the Boolean input *use_bubbles* that indicates whether bubbles or collision checks are used for path validation.

ConnectPoints is a recursive function that bisects the line connecting the denoted points. In case that bubbles are not used for validation, the algorithm checks collision for the

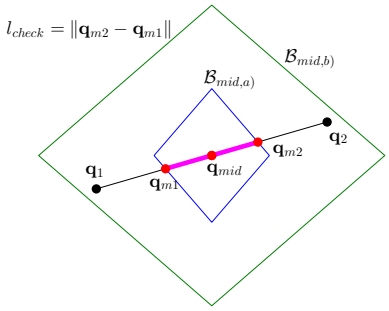

Fig. 5. Case *a* - the bubble intersects the segment at two points. Case *b* - the bubble contains the complete segment.

midpoint, and if there is no collision, calls itself for each of the halves of the segment, still disallowing bubble-based validation. The segment lengths always halve until they reach a tiny length of a certain threshold ε_0, at which point it is safe to assume there is no collision on the segment.

In case bubbles are used for validation, the procedure starts by creating a bubble at the midpoint of the segment, as shown in Fig. 5. The algorithm is agnostic on if the expanded version of the bubble is used or not, and both cases will be observed. If the bubble is large enough to encompass one point, due to symmetry it will encompass both points, as demonstrated by the case *b* in Fig. 5. That means the segment is guaranteed to be collision free, and the function returns True for that segment. Otherwise, the bubble will intersect with the line segment in points $\mathbf{q}_{m1}$ and $\mathbf{q}_{m2}$, as demonstrated in Fig. 5 (case *a*). The computation of points $\mathbf{q}_{m1}$ and $\mathbf{q}_{m2}$ is straightforward. By the definition of the bubble, the border of $\mathcal{B}(\mathbf{q}_{mid})$ is given by:

$$\partial\mathcal{B}(\mathbf{q}_{mid}) = \{\mathbf{y} \mid \textstyle\sum_{i=1}^{n} r_i |y_i - q_{mid,i}| = d_c\}. \quad (10)$$

On the other hand, the set of points $\mathbf{Y} = [y_1\ y_2\ \ldots\ y_n]^T$ that belong to a straight line $\overline{\mathbf{q}_{m1}\mathbf{q}_{m2}}$ is given by:

$$y_i = q_{mid,i} + (q_{2i} - q_{1i})\, t, \quad i = 1, \ldots, n, \quad t \in \mathbb{R}. \quad (11)$$

Combining (10) and (11), to compute the intersection of $\overline{\mathbf{q}_1\mathbf{q}_2}$ with the bubble $\mathcal{B}(\mathbf{q}_{mid})$, the following is obtained:

$$\textstyle\sum_{i=1}^{n} r_i |y_i - q_{mi}| = |t| \sum_{i=1}^{n} r_i |q_{2i} - q_{1i}| = d_c. \quad (12)$$

The two solutions for parameter t, and hence points $\mathbf{q}_{m1}$ and $\mathbf{q}_{m2}$ (see equation (11)) follow directly from:

$$|t| = \frac{d_c}{\sum_{i=1}^{n} r_i |q_{2i} - q_{1i}|}. \quad (13)$$

If the distance l_{check} between points $\mathbf{q}_{m1}$ and $\mathbf{q}_{m2}$ is too small, that means that bubbles have become too small to be of use, and it is better to switch to collision checking for nested calls to ConnectPoints. In case of a collision, l_{check} is equal to 0. After that, ConnectPoints is called for the $\mathbf{q}_1$–$\mathbf{q}_{m1}$ segment, collision of the midpoint is checked, and ConnectPoints is called again for $\mathbf{q}_{m2}$–$\mathbf{q}_2$. If either of those checks fail, ConnectPoints returns False. Any bubble that is managed to be reached collision free is added to the tree,

function CONNECTPOINTS($\tau, \mathcal{B}_{par}, \mathbf{q}_1, \mathbf{q}_2, use\_bubbles$)

```
q_mid ← (q_1 + q_2)/2
if use_bubbles then
    B_mid ← Bubble(q_mid)
    if B_mid.Contains(q_2) then
        τ.AddVertex(B_mid)
        τ.AddEdge(B_par, B_mid)
        return True
    q_m1 ← B_mid.IntersectionTowards(q_1)
    q_m2 ← B_mid.IntersectionTowards(q_2)
    l_check ← ||q_m1 − q_m2||
    u ← l_check > l_min_bubble
    if ¬ConnectPoints(τ, B_par, q_1, q_m1, u) then
        return False
    if l_check = 0 then
        return False
    τ.AddVertex(B_mid)
    τ.AddEdge(B_par, B_mid)
    if ¬ConnectPoints(τ, B_mid, q_m2, q_2, u) then
        return False
else
    if ||q_1 − q_2|| < ε_0 then
        return True
    if IsCollision(q_mid) then
        return False
    if ¬ConnectPoints(τ, B_par, q_1, q_mid, False) then
        return False
    if ¬ConnectPoints(τ, B_par, q_mid, q_2, False) then
        return False
return True
```

regardless of the final outcome of ConnectPoints. The $2n$ vertices of an added n-dimensional bubble are added as nodes for nearest neighbor checks. On top of ConnectPoints, we build functions Extend and Connect. The Connect function finds the nearest node, and the corresponding bubble $\mathcal{B}_{near}$, in the tree. If the bubble contains the target $\mathbf{q}$, the connect succeeded. Otherwise, we find the point $\mathbf{q}_{near}$ at which the line between the center of $\mathcal{B}_{near}$ and $\mathbf{q}$ intersects the border of $\mathcal{B}_{near}$, and attempt a ConnectPoints between $\mathcal{B}_{near}$ and $\mathbf{q}$. The Extend function performs almost in identical way, except it limits the length of the segment to a certain ε, disallowing the procedure to be too greedy.

The BubbleRRT algorithm operates in a similar way to the RRT-Connect planner. The trees store bubbles that have collision-free paths to them. And while RRT-Connect calls each tree sequentially with a new random configuration $\mathbf{q}_{rand}$, the BubbleRRT planner performs the same procedures simultaneously, with the same $\mathbf{q}_{rand}$. First, Extend procedure is called on both trees, performing random exploration toward $\mathbf{q}_{rand}$. The bubble that was last added to a tree is the final bubble in the newly formed branch. That is why we use the newest bubble's position to perform Connect on the other tree, for each tree respectively. If either of those connection attempts succeeds, the trees are able to connect, and the path is retrieved by connecting the starting and final point.

IV. EXPERIMENTS

This section presents numerical results for the proposed algorithm. Simulation study includes several scenarios involving three different robotic manipulators. The algorithm

function EXTEND($\tau, \mathbf{q}$)

$\mathcal{B}_{near} \leftarrow NearestNeighbor(\mathbf{q}, \tau)$
$\mathbf{q}_\Delta \leftarrow \mathbf{q} - \mathcal{B}_{near}.\mathbf{q}$
$\hat{\mathbf{q}}_\Delta \leftarrow \frac{\mathbf{q}_\Delta}{\|\mathbf{q}_\Delta\|}$
$\mathbf{q}_{new} \leftarrow \mathcal{B}_{near}.\mathbf{q} + \hat{\mathbf{q}}_\Delta \cdot \min(\varepsilon, \|\mathbf{q}_\Delta\|)$
if $\mathcal{B}_{near}.Contains(\mathbf{q}_{new})$ **then**
  return True
$\mathbf{q}_{near} \leftarrow \mathcal{B}_{near}.IntersectionTowards(\mathbf{q}_{new})$
return $ConnectPoints(\tau, \mathcal{B}_{near}, \mathbf{q}_{near}, \mathbf{q}_{new}, \text{True})$

function CONNECT($\tau, \mathbf{q}$)

$\mathcal{B}_{near} \leftarrow NearestNeighbor(\mathbf{q}, \tau)$
if $\mathcal{B}_{near}.Contains(\mathbf{q})$ **then**
  return True
$\mathbf{q}_{near} \leftarrow \mathcal{B}_{near}.IntersectionTowards(\mathbf{q})$
return $ConnectPoints(\tau, \mathcal{B}_{near}, \mathbf{q}_{near}, \mathbf{q}, \text{True})$

function BUBBLERRT($\mathbf{q}_{init}, \mathbf{q}_{goal}$)

$\tau_a.init(Bubble(\mathbf{q}_{init}))$
$\tau_b.init(Bubble(\mathbf{q}_{goal}))$
for $k \leftarrow 1, K$ **do**
  $\mathbf{q}_{rand} \leftarrow RandomConfig()$
  $Extend(\tau_a, \mathbf{q}_{rand})$
  $Extend(\tau_b, \mathbf{q}_{rand})$
  $\mathbf{q}_{new,a} \leftarrow \tau_a.GetNewestBubble().\mathbf{q}$
  $\mathbf{q}_{new,b} \leftarrow \tau_b.GetNewestBubble().\mathbf{q}$
  $c_a \leftarrow Connect(\tau_a, \mathbf{q}_{new,b})$
  $c_b \leftarrow Connect(\tau_b, \mathbf{q}_{new,a})$
  if $c_a \vee c_b$ **then**
    return $Path(\tau_a, \tau_b)$
return Failure

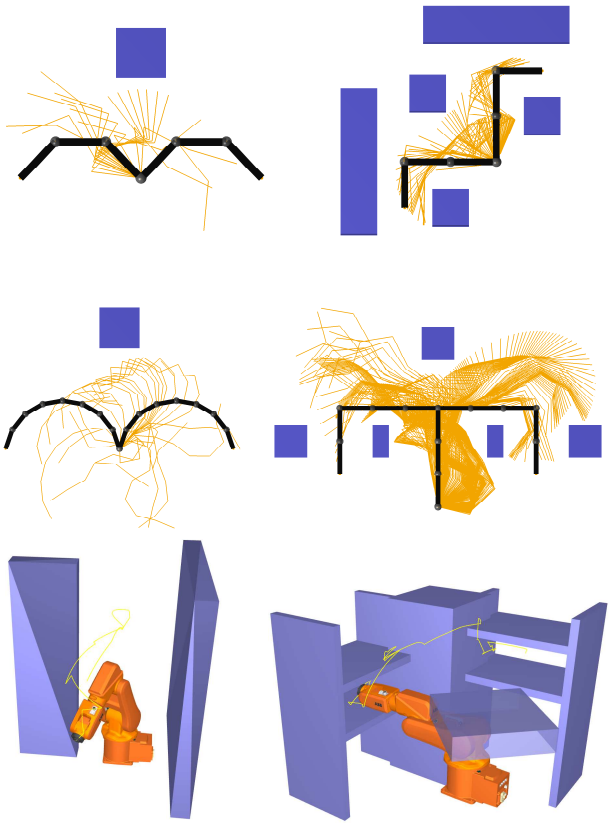

Fig. 6. Scenarios used in simulation study. 3 DOF planar robot (top), 8 DOF planar robot (middle), and 6 DOF industrial robot (bottom). The first column shows easier scenarios and the second column shows harder ones.

is compared to classical RRT-CONNECT planner [4]. Moreover, an RRT-based planning algorithm from [9] that uses bubbles for tree extension is implemented as well, with slight modifications for the sake of fair comparison (the algorithm will be referred to as RRT-CONNECT_B). Unlike in [9], our implementation includes bidirectional search with the CONNECT procedure and considers the specific goal configuration, rather than a goal region. A similar algorithm that uses bubbles for validating local paths is proposed in [10]. However, its performance has already been shown to be inferior to that of the classical RRT [10]. Therefore, we do not include it in the simulation study. Moreover, algorithms from [11], [12], [13] are currently out of the scope for comparison since they do not fall into RRT paradigm.

Fig. 6 shows the scenarios used for testing the algorithms, along with the examples of solution paths. Three types of manipulators are considered: 3 DOF planar manipulator, 8 DOF planar manipulator, and a 6 DOF ABB IRB120 industrial manipulator. Each manipulator is subjected to two test scenarios with two levels of difficulty: easy and hard. Distance/collision queries have been performed using FCL library [8]. All the simulations have been performed on a PC with Intel i5 3.3 GHz processor, and with 8GB RAM. For a single scenario, each algorithm is executed 100 times. It is worth pointing out that, prior to performing experiments, the step size for RRT_CONNECT algorithm has been optimized (for each type of manipulator) to achieve the fastest execution without collisions occurring along the computed path. During this optimization phase, the feasibility of solution paths was established using continuous collision detection. Note that continuous collision detection has not been used within experiments. The obtained step size is also used within bubble-based planners during the phases when they switch to collision checking mode.

For computing expanded bubbles, the distance profile $d(s)$ has been approximated by the function $d_N(s)$ using the partition with $N = n$ and $s_k = L_k$. Hence $d_N(s)$ takes constant values for each separate link. Thus, we avoided the extra computational cost regarding the distance query, when compared to computation of the original bubbles. Preliminary attempts to further increase the resolution N and better approximate the real distance profile $d(s)$, by denser "slicing" of the robot links, did not result in measurable performance improvement. We attribute this to the obvious fact that the distance query needs to be called for each slice, i.e., more frequently. Nevertheless, we believe that a careful partition, along with a proper simplification of the model geometry, could yield further improvements. Clearly, more research needs to be done to support this.

Table I shows the performance indicators for the tested algorithms. All the tested planners achieve a success rate of 100% for all scenarios. The experimental results show that

TABLE I

NUMERICAL RESULTS

			3 DOF easy				**3 DOF hard**	
Algorithm	Time (s)	Nodes	Collision/Distance queries	Iterations	Time (s)	Nodes	Collision/Distance queries	Iterations
RRT-CONNECT	0.4733	9068.6	260592/0	2908.4	2.2387	7600.2	275840.8/0	8055.6
RRT-CONNECT_B	4.2648	10070	423017/17663	7594	215.51	434542	27520132/1026187	591646
BUBBLE-RRT	0.0172	763.5	3716.3/174.95	31.9	0.2942	6459	22925/1943	460.1
E-BUBBLE-RRT	0.0134	641.7	3618.7/154.35	31.2	0.2748	6018	21519.5/1755.1	407.8
			8 DOF easy				**8 DOF hard**	
Algorithm	Time (s)	Nodes	Collision/Distance queries	Iterations	Time (s)	Nodes	Collision/Distance queries	Iterations
RRT-CONNECT	10.7707	107326	2886450.8/0	16245.2	87.6072	372029	10531419.6/0	101636
RRT-CONNECT_B	47.0035	108556	4759291/172417	63862	720.4806	761941	35691378/1039826	277886
BUBBLE-RRT	0.2194	8463.2	23281.6/688.3	115.6	14.1788	354745	897440.6/28702.1	3739.4
E-BUBBLE-RRT	0.0947	3417.6	8652.5/293.7	55.4	10.2419	224900	646848.4/19642.8	3035.9
			ABB IRB120 easy				**ABB IRB120 hard**	
Algorithm	Time (s)	Nodes	Collision/Distance queries	Iterations	Time (s)	Nodes	Collision/Distance queries	Iterations
RRT-CONNECT	18.5217	17511.2	505426/0	5511.6	35.0032	54968	1670984.4/0	26139.6
RRT-CONNECT_B	87.7714	28838	1315318/45443	16606	267.8443	102265	5651482/186564	84300
BUBBLE-RRT	0.6316	6225	16549.9/634.8	98.9	14.4243	111470	327074.5/15291.9	3253.2
E-BUBBLE-RRT	0.6085	5641.2	17423/605.2	106.2	11.9966	78442	226930.7/10768.4	2293

the BUBBLE-RRT algorithms are computationally more efficient than other planners used for comparison. In particular, the version that uses expanded bubbles (E-BUBBLE-RRT) enables noticeable improvements with respect to BUBBLE-RRT algorithm that uses original bubbles for local path validation. The main reason behind the efficiency of the proposed algorithm is clearly a relatively small number of iterations necessary to compute the solution path (see Table I). When possible, the ConnectPoints routine enables greedy extensions using rather bulky bubbles, resulting in quick exploration of larger portions of free C-space. If the robot gets very close to obstacles, the bubbles get smaller and the algorithm switches to collision checking, thus emulating a classic RRT mode, to avoid the expensive distance computations. Nevertheless, fairly explored regions of C-space facilitate the exploitation phase, thus improving the chance for the CONNECT routine to succeed.

V. CONCLUSIONS

We described a simple algorithm that uses bubbles of free C-space to compute collision free paths for robotic manipulators. The algorithm mimics a classical RRT-Connect, yet uses modified procedures for extending and connecting the trees via bubbles. To increase the exploration capabilities of the algorithm, an expanded version of bubbles was introduced. Experimental results show that our algorithm outperforms both the classical planner and another bubble-based method.

Future work will focus on gaining more knowledge about the free C-space, based on a limited distance information from the workspace. This includes more elaborate geometric representation and slicing of the robot to obtain good, yet inexpensive distance profile approximations. Moreover, the proposed collision checking routine will be applied within other planning methods and thus compared to other collision detection methods independently from RRT context. Finally, an interesting option is to tackle the problem of optimal path planning using expanded bubbles of free C-space.

REFERENCES

[1] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *Robotics and Automation, IEEE Transactions on*, vol. 12, no. 4, pp. 566 –580, aug 1996.

[2] R. Bohlin and L. Kavraki, "Path planning using lazy PRM," in *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, vol. 1, 2000, pp. 521–528.

[3] S. M. LaValle and J. J. Kuffner, "Rapidly-exploring random trees: Progress and prospects," in *Proceedings Workshop on the Algorithmic Foundations of Robotics*, 2000.

[4] J. Kuffner, J.J. and S. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, vol. 2, 2000, pp. 995–1001.

[5] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, June 2011.

[6] S. Quinlan, "Real-time modification of collision-free paths," Ph.D. dissertation, Stanford University, 1995.

[7] E. Larsen, S. Gottschalk, M. Lin, and D. Manocha, "Fast distance queries with rectangular swept sphere volumes," in *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, vol. 4, 2000, pp. 3719–3726 vol.4.

[8] J. Pan, S. Chitta, and D. Manocha, "FCL: A general purpose library for collision and proximity queries," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, May 2012, pp. 3859–3866.

[9] D. Bertram, J. Kuffner, R. Dillmann, and T. Asfour, "An integrated approach to inverse kinematics and path planning for redundant manipulators," in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, 15-19 2006, pp. 1874 –1879.

[10] N. Vahrenkamp, T. Asfour, and R. Dillmann, "Efficient motion planning for humanoid robots using lazy collision checking and enlarged robot models," in *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, Oct 2007, pp. 3062–3067.

[11] B. Lacevic and P. Rocco, "Towards a complete safe path planning for robotic manipulators," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, Oct 2010, pp. 5366–5371.

[12] ——, "Safety-oriented path planning for articulated robots," *Robotica*, vol. 31, no. 06, pp. 861–874, 2013.

[13] A. Ademovic and B. Lacevic, "Path planning for robotic manipulators via bubbles of free configuration space: Evolutionary approach," in *Control and Automation (MED), 2014 22nd Mediterranean Conference of*. IEEE, 2014, pp. 1323–1328.

[14] A. C. Shkolnik and R. Tedrake, "Sample-based planning with volumes in configuration space," *CoRR*, vol. abs/1109.3145, 2011.

[15] J. Bialkowski, M. Otte, S. Karaman, and E. Frazzoli, "Efficient collision checking in sampling-based motion planning," in *Algorithmic Foundations of Robotics (WAFR), 10th International Workshop on*, 2013, pp. 365–380.

[16] B. Lacevic, D. Osmankovic, and A. Ademovic, "Burs of free C-space: a novel structure for path planning," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, 2016, accepted for publication.