A Representation Of Deformable Objects For Motion Planning With No Physical Simulation

Calder Phillips-Grafflin and Dmitry Berenson
Worcester Polytechnic Institute
{cnphillipsgraffl, dberenson}@wpi.edu

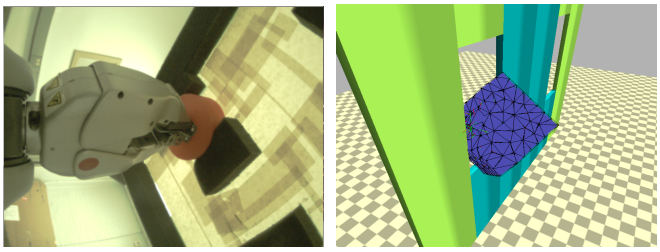
***Abstract*— We propose a new method of representing deformable objects that allows both physical and qualitative properties to be captured in an efficient representation. We show how to use this representation with two types of motion planners: 1) optimal discrete planners, which are suitable for low-dimensional problems, 2) sampling-based planners that plan in high-dimensional cost spaces. In both cases, our representation allows us to formulate a cost function that directly assesses the cost of deformation without expensive physical simulation or computation of deformed geometry. We show that our methods can generate paths that minimize deformation in both simulated and physical environments with either hard and soft robots in either hard and soft environments. The efficiency of our representation allows these paths to be computed in under 20s for 3-DOF problems. For more complicated 6-DOF problems, low-deformation paths can be computed in under 120s. Additionally, using feedback from simulated and physical test environments, we demonstrate methods for calibrating models based on our representation.**

I. Introduction

This paper addresses the problem of computing motion plans for scenarios where the object being manipulated by the robot and/or the environment are deformable. This important class of problems arises in everyday environments, such as putting on clothing or cooking food, as well as in surgical and industrial settings.

The primary challenge of motion planning for deformable objects/environments is that deformability is very difficult to simulate accurately. Unlike rigid objects, whose dynamics are well-understood, the motion of a deformable object depends on a large and complex set of parameters that define its stiffness, friction, and volume preservation. Computing the geometry of a deformable object in contact with another object is particularly challenging, especially if both objects are deformable. Many methods exist for deformable object simulation, including mass-spring model simulation [1] and the more general finite element method (FEM) [2], [3]. Simulation methods for meshless models also exist [4]. FEM simulation is generally regarded to be the most accurate, although it is highly sensitive to the discretization of the deformable object and, for fine discretizations, is very time-consuming to compute.

Given the difficulties of deformable object simulation, we seek to explore the practicality of performing useful motion planning for deformable objects without explicitly simulating them. To accomplish this, we model the environment and moving object using voxel grids. Each contains two values: *deformability* and *sensitivity*. The deformability represents how compressible the voxel is. The sensitivity represents the penalty for deforming that particular voxel. Sensitivity is used to allow our motion planner to avoid some objects more than others in the planning process, which is useful if different objects have different sensitivity to deformation. For instance, in a surgical setting, one organ may be more sensitive to compression than another, even though both are equally deformable.



(a) (b)

Fig. 1: Execution of paths planned using our representation: (a) with a PR2, (b) in the Bullet physics simulator.

Our representation is designed to represent objects that deform elastically, meaning that while the object may change its surface geometry when in contact, it exhibits volume restoration and will return to its original shape when the colliding object is removed (e.g. a sponge). Note that our representation currently assumes that environment objects, whether deformable or rigid, do not move as a result of deformation or other forces. While restrictive, we believe that these limitations are consistent with a range of real-world problems where deformable objects are constrained by the environment (e.g. inside the body) or by the robot itself (e.g. a robot with deformable components). In cases that do not completely meet these constraints, we believe our representation will be conservative – it will over-estimate the severity of deformation. Certain objects such as clothing or rope that do not obey these limitations may instead be represented as an articulated series of these voxel-based representations, though this is not within the scope of this paper.

Once the moving object and environment have been defined using our voxel-based representation, we can evaluate the cost of a given configuration of the object by computing a novel cost function that combines deformability and sensitivity into a single value. This value represents the *deformation cost* of that configuration. Using this cost function, the cost of

a given configuration in a motion planner may be computed. As we demonstrate in this paper, this cost function is suitable for both discrete (demonstrated using a variant of A*) and sampling-based motion planning algorithms (demonstrated using T-RRT [5] and GradienT-RRT [6]).

In our experiments we show that our method is effective at finding paths for rigid objects moving in deformable environments, deformable objects moving in rigid environments, and deformable objects moving in deformable environments. Computing paths for these scenarios would involve radically different simulation methods, however, using our approach, we need only to adjust the deformability of the object and environment. We verify the efficacy of our method in virtual environments using the Bullet physics simulator, and in physical experiments using the PR2 robot with a custom deformation-tracking camera system.

The remainder of the paper is structured as follows: Section II gives a background on previous work on simulation and motion planning of deformable objects, Section III describes our voxel-based representation, cost function, and planning algorithms. Section IV presents results in simulation and on PR2. Finally, Section V concludes the paper.

II. Background

Deformable objects are important for a variety of fields ranging from computer graphics to robotics to medicine. As a result, extensive work has been done on the modelling and representation of deformable objects for computer graphics [1] and medicine [7]. More recently, this work has been adapted to robotics to allow the manipulation of real-world objects such as clothing, rope, and human tissue. Unlike a range of work focusing on visual servoing [8], [9] and haptics [10], [11] with deformable objects, we focus on motion planning for deformable objects – i.e. we seek to compute a path that minimizes deformation.

While a wide variety of modelling approaches for deformable objects exist for computer graphics and physical simulation purposes, we are primarily concerned with representations suitable for motion planning purposes. We wish to avoid the problem of directly computing the geometry of a deformed object, as doing so is an expensive intermediate step to assessing the severity of deformation.

Existing representations of deformable objects fall into two main groups, those using meshed volumetric models and those using meshless models.

Meshed - Often tetrahedral meshes, these models preserve volumetric constraints and allow the use of numerical simulation to compute the effects of collision. Existing work uses Mass-Spring (M-S) [1] and Finite-Element (FEM) [2],[3] methods to simulate changes to object geometry resulting from collisions. These methods allow the inclusion of volume preservation and restoration. However, as noted in [12], M-S models are inaccurate beyond low-deformation cases, and both M-S and FEM are expensive to compute.

Meshless - Two variants of this approach exist – models that represent only the surface of the object such as [12], and models such as [4], which use a discretized representation that allows for heterogeneous objects with varying internal properties. In the former case, the surface model was used to compute the penetration into the object, ignoring internal forces. In the latter, discrete information on material properties allows the efficient computation of object deformation with comparable accuracy to established FEM methods [4].

Some deformable objects, such as rope and thread, can change shape with limited (or no) restoring forces. Existing models for these objects assume that the object itself is incompressible [13] (i.e., a rope cannot be crushed), which prevents the application of these techniques to problems such as ours, in which the objects modelled must be compressible.

Building from these established representations and techniques, a range of motion planning approaches have been developed to find paths in deformable environments. Extensive work has been done applying Probabilistic Roadmap planners (PRMs [14]) and Rapidly-exploring Random Tree planners (RRTs [15]) to deformable objects, including [16], [17], [18], [19], [20], [21]. Other work in the area includes planning for rope and thread such as [13].

However, the above work is marked by a trade-off between the desires for accuracy and performance. Accurate methods based on FEM models are slow to compute, prompting a range of simpler models such as [17], [12] which are designed to provide "good-enough" simulation of deformation. Furthermore, most previous work is concentrated on finding *feasible* deformations using volume preservation and penetration distance to evaluate feasibility, whereas we seek to minimize deformation.

Limited work has been done to account for the severity of the deformation, such as [12] and [21], which assess a cost of deformation. In the former, the problem of motion planning itself is replaced with an optimization problem of reducing the cost for motion along a parametrized Bezier curve below a preset threshold. Instead of a local optimization approach like this, our approach is to use global planners.

The most similar existing works to ours are [4] and [21], the former in terms of representation, and the latter in terms of planning. Using an underlying discrete representation similar to ours, [4] simulates the geometry and kinematics of heterogeneous deformable objects. While this technique could be used as a stepping-stone to compute a cost of deformation, our method skips the simulation step to directly compute a cost value. In addition, neither approach incorporates an element comparable to the *sensitivity* used in our representation.

In terms of planning, [21], like our method, attempts to generate optimal paths, taking into account both costs incurred in deformation and path length. Unlike our approach, [21] relies on extensive and time-consuming pre-computation of a meshed environment using FEM methods.

III. Methods

We have developed an efficient representation for deformable objects and a cost function for assessing the cost of collisions. Building from this representation, we have

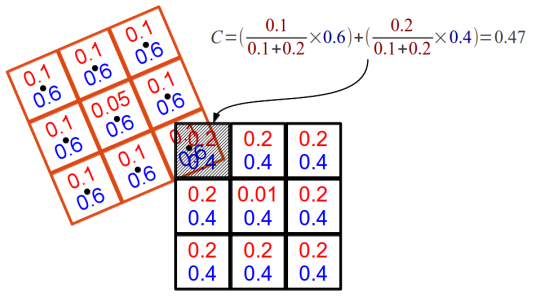


Fig. 2: An example of our representation and cost function – deformability values are shown in red, sensitivity values are shown in blue. The voxel centers of the moving object (orange) are shown as black points. Cost is computed for the shaded voxel in collision.

developed a cost function that allows discrete and sampling-based planners to compute paths that minimize deformation.

A. Representation

Object geometry is inherently captured in our voxel-based representation. Unlike triangle meshes and other methods optimized for surface representation, this discretization preserves information about the interior of the object. As with all discrete representations, the resolution of our representation is limited by the size on an individual voxel. Arbitrarily high resolution can be achieved by increasing the number and decreasing the size of the voxels, at the cost of increased memory usage and processing time. To address the well-known problem of rotating voxels, only the planning environment is directly modelled using voxels; objects being moved are represented by a set of points that shares the same discretization, in which the points correspond to voxel centers. An example representation is shown in Figure 2.

Our representation captures physical properties through the use of two parameters per voxel, *Sensitivity* and *Deformability*. These parameters represent the cost incurred by deforming the voxel, and the ability of the voxel to be deformed (similar to the "stiffness" in [4]). Intuitively, a completely rigid object has a deformability of zero, while empty space has a deformability of one. Sensitivity is a user-assigned parameter that allows a range of object qualities to be represented. In general, sensitivity may be used to differentiate between two deformable objects with similar physical *properties* but different desired *qualities*.

Increasing sensitivities from the surface to the center of an object can be used to represent a greater severity of deformation (e.g. the tissue inside an organ may be much more prone to damage than the exterior) or, by increasing sensitivity to infinity, effectively prevent the planner from producing paths that result in any penetration. While deformability parameters can be derived from physical properties of an object, tuning sensitivity parameters is more complicated. In future work, we plan to investigate the automatic generation of sensitivity parameters.

B. Cost Function

Using our voxel-based representation, we develop a cost function to assess the cost of collision between two objects. As noted already, previous work with deformable objects requires the expensive calculation of the deformed geometry; our method, however, directly assesses the costs resulting from this deformation by observing the intersecting volume of objects in collision. Cost is computed for each voxel of the object (or objects) being moved in collision, and the sum of these per-voxel costs is the total cost of deformation for a given state.

Per-voxel cost is evaluated using Equation 1. Let C_i be the total deformation cost of voxel i, while $S_i(A)$ and $S_j(B)$ are the sensitivity parameters of voxel i in object A and j in B, respectively. Similarly, $D_i(A)$ and $D_j(B)$ are the deformability parameters of voxels i and j.

$$C_i(A,B) = \frac{D_i(A)}{D_i(A)+D_j(B)} * S_i(A) + \frac{D_j(B)}{D_i(A)+D_j(B)} * S_j(B) \quad (1)$$

Intuitively, this cost function assigns cost based on the weighted combination of costs incurred by both objects. If both objects have the same deformability, each will contribute equally to the total cost (if $S_i(A) = S_j(B)$), while in cases of varying deformability, the "softer" object with higher deformability contributes more to the total cost. In cases of hard-on-soft or soft-on-hard collision where one object is completely rigid, only the soft object being deformed contributes to the total cost.

A notable special case of Equation 1 exists if both $D_i(A)$ and $D_j(B)$ are zero (meaning that voxels i in A and j in B are both rigid), in which case $C_i(A,B)$ becomes undefined. This property is used for implicit collision detection in our planner, as our implementation of the cost function returns NaN in these cases, which allows states resulting in rigid body collisions to be eliminated. Similarly, using NaN for sensitivity values can also be used to make certain states explicitly infeasible.[1]

C. Discrete Path Planning

For low-dimensional problems, we use a planning algorithm based on the well-known A* search algorithm [22]. A* is both complete, meaning that it will always find a solution if it exists, and optimal, meaning that it will always find the minimum-cost path to the solution.

In its conventional form, A* orders states based on the combined value of $g(s)$, the *cost* of moving to state s, and $h(s)$, the *heuristic* value of s. A* alone is sufficient for use in rigid environments in which states are either feasible and free of collision, or infeasible due to collision; however, A* is insufficient to handle planning in deformable environments with feasible collisions.

Thus, we adapt the A* algorithm to account for the *deformation cost* in addition to the path length cost, as seen in the DEFORM and FVALUE functions in Algorithm 1. The DEFORM function computes the total cost of deformation

[1] As defined in the IEEE Floating Point standard, NaN "poisons" calculations, so a single voxel cost of NaN results in a total cost of NaN.

Algorithm 1 Cost function for our A* planning algorithm

procedure FVALUE(s, p)
  return $(1-p) \times (h(s) + g(s)) + p \times$ DEFORM(s)
procedure DEFORM(s)
  $cost \leftarrow s.parent.cost$
  for each point A in $s.shape$ **do**
    $B \leftarrow$ LOOKUP(A)
    $cost \leftarrow cost + \mathrm{C}(A, B)$
  return $cost$

in a given state using $C(A, B)$, our cost function shown in Equation 1 for all voxels in the moving object. LOOKUP transforms a given point in the object being manipulated into the planning environment and returns the corresponding voxel. FVALUE, which implements the classical $f(s) = g(s)+h(s)$ in A*, is extended to incorporate the deformation cost of the path to s, which is returned by DEFORM. Here, $g(s)$ is the path length from the start to state s, and $h(s)$ is the euclidean distance from s to the goal.

It is important to point out that we have not simply appended deformation cost to the overall state cost. Instead, we have incorporated the concept of Pareto-optimality for paths discussed in [23], which allows us to compute paths with varying definitions of optimality. This control is introduced with the parameter p, which weighs the deformation against path length. Intuitively, low values of p induce greater deformation if doing so will result in a shorter path, as they increase the relative penalty of path length. High values of p may result in lower deformation at the expense of longer paths. The two boundary cases of p, namely $p = 0$ and $p = 1$, result in conventional A* (ignoring deformation) and best-first search considering only deformation, respectively. Note that using $1 - p$ to scale the heuristic is necessary to ensure that the heuristic remains admissible.

In practice, selecting values for p close to 0 may result in undesirably high deformation and values close to 1 may produce unnecessarily long paths. In cases where no deformation-free path exists, the effect of p is dependent on the total deformation encountered and the cost parameters of the deformable objects. In practice, the paths for the range of p values can be presented to the user, allowing the user to select from the Pareto front.

D. Sampling-based Planning

For higher-dimensional problems, we evaluate both the T-RRT algorithm [5] and the GradienT-RRT algorithm [6] with our representation. Both are probabilistically-complete sampling-based planners suitable for cost-space planning in high-dimensional spaces. Our choice of these planners instead of the better-known RRT* [24] is because T-RRT has been shown to outperform RRT* [25], [26] in higher dimensions and because GradienT-RRT is an extension of T-RRT specifically intended for narrow low-cost regions such as those encountered in many scenarios where objects need to be deformed to complete a task.

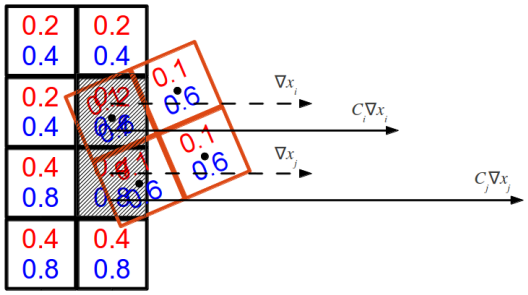


Fig. 3: Workspace gradient computation: initial per-voxel gradients $\nabla x_i, \nabla x_j$ (dashed arrows) are computed using a signed distance field in GradienT-RRT. These gradients are scaled by the cost function evaluated at the voxel of intersection (shaded) to produce the final $C_i \nabla x_i, C_j \nabla x_j$ (solid arrows).

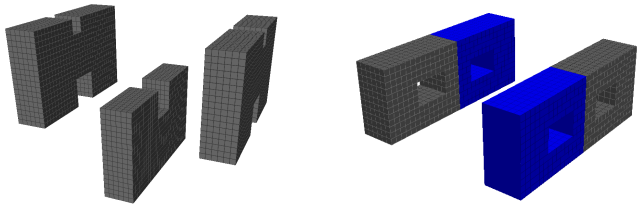
The T-RRT algorithm, unlike the basic RRT, uses cost to control the addition of nodes to the tree. Addition of nodes is a function of the cost of the new node, the cost of its parent, and the distance between them (see Algorithm 2 in [5] for details). New nodes of lower cost than their parents are automatically added. Higher-cost nodes are added to the tree with probability dependent on the cost increase and the current *temperature*. Expansion behavior is primarily controlled with the $nFailMax$ parameter, which specifies the number of unsuccessful extensions required to increase temperature. Effectively, $nFailMax$ can be used to trade between cost and planning time – lower values will result in more rapid expansion (and thus faster planning), while higher values will result in lower cost solutions, usually at the expense of longer planning time.

The GradienT-RRT algorithm is designed to address particular shortcomings of T-RRT, namely in the inability of T-RRT to follow narrow valleys in the cost-space. Instead of simply rejecting higher-cost nodes, GradienT-RRT adjusts them using the gradient of the cost function. If these new nodes result in lower cost, they are then added to the tree. GradienT-RRT has been previously applied to a range of cost-space problems including those with workspace, task-space, and configuration-space costs. However, GradienT-RRT requires a function that computes the gradient ∇q of the cost function at configuration q in addition to the cost of that configuration. Our approach to computing the gradient derives from previous work planning for workspace uncertainty [6].

The gradient for a given state is computed as shown in Equation 2 in a similar manner to that used in [6].

$$\nabla q = \mathbf{J}(q, x_1, x_2, ...)^T [C_1 \nabla x_1^T, C_2 \nabla x_2^T, ...]^T \qquad (2)$$

Here, a workspace gradient ∇x_i is computed for each voxel x_i of the robot that intersects an obstacle at configuration q. As this workspace gradient only reflects penetration of filled voxels, we multiply the magnitude of the gradient for each voxel with the cost computed from our aforementioned cost function C_i as shown in Figure 3. We use the Jacobian $\mathbf{J}(q, x_1, x_2, ...)$, a composition of the Jacobians for each point in the intersection, to convert this workspace gradient to the



(a) Triple-wall (b) Multi-cost

Fig. 4: Simulation environments (a) three walls with a deformation-free path, (b) two walls with multiple sensitivity values – blue sections have half the sensitivity of gray sections.

C-space gradient ∇q needed for GradienT-RRT. For both T-RRT and GradienT-RRT planners, edge cost is simply the change in cost between a node and its parent. Note that we do not currently have an equivalent to p for our sampling-based planners, as the paths produced by T-RRT and GradienT-RRT are not guaranteed to be optimal (though they have low cost in practice).

IV. Results

We have applied our methods to both simulation-only environments and simpler physical environments manipulated by a PR2 robot. For low-dimensional problems, we have implemented a planner integrated with ROS [27], which provides both visualization for planning and testing, control of the PR2, and the interface to our custom deformation tracking system. For higher-dimensional problems, we have modified the existing GradienT-RRT planner in the OpenRAVE [28] planning environment and implemented a validation environment using the Bullet physics simulation engine [29]. We show the performance of these planners on several problems and report time and cost results. We also present a way to calibrate our model.

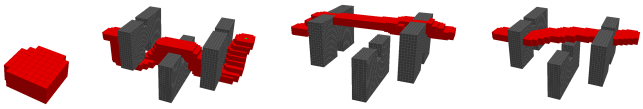
A. Low-dimensional Planning

We first demonstrate the capabilities of our representation in low-dimensional environments using our A*-derived planner. We use a set of simulated environments and objects with different combinations of hard and soft material properties. The environments are modelled at a resolution of 8mm, for a total size of 54000 voxels. Due to the well-known performance problems of applying A* directly to high-dimensional problems, we limit planner control to 3D translation of the object.

1) Environment: We have built two simulation environments, shown in Figure 4, that demonstrate the capabilities of our low-dimensional planning method. The first of these environments provides a set of distinct pareto-optimal paths with decreasing path lengths and increasing cost, while the second environment illustrates the control provided by the sensitivity parameter (e.g. $S_i(A)$) of our representation. With both of these environments, the behavior of the planner can be controlled using p. Additionally, due to the simplicity of our representation, it is trivial to change these environments to reflect all four combinations of hard/soft obstacles and robot, and we report results for all of these cases.

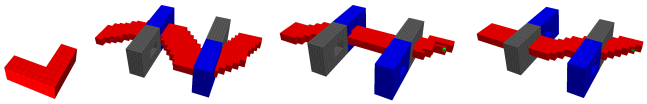
The first environment, ("triple-wall"), allows both deformation-free and deformed paths. By varying p, we are able to control the balance between deformation and path length. Additionally, due to the existence of a deformation-free solution, it can be simulated as a fully rigid environment instead, and we provide this for comparison.

The second environment, ("multi-cost"), illustrates the capability of our deformability and sensitivity representation. Both black and blue obstacles in this environment exhibit the same nominal physical properties (captured by deformability). However, we assign lower *sensitivity* to the blue obstacles to indicate that deformation of them is less severe. As before, using p we can tune the behavior of the planner to produce paths that deform either or both blue and black obstacles. While simple, this environment illustrates the advantage of our representation, namely that it combines both physical and qualitative properties of objects that are difficult to capture using purely mechanical models.



(a) (b) (c) (d)

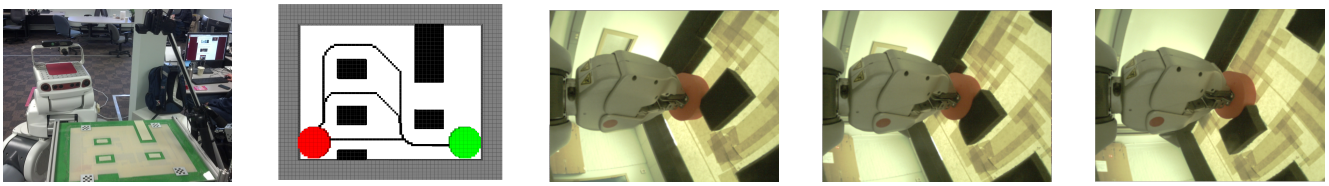
Fig. 5: Path classes for the Triple-wall simulation environment (soft) using the robot (hard) shown (a) with the swept volume of the robot shown in red (b) deformation free path: length = 94, p = 0.7, deformation = 0, (c) medium deformation path: length = 65, p = 0.01, deformation = 683, and (d) highest deformation path: length = 61, p = 0.0, deformation = 1062.



(a) (b) (c) (d)

Fig. 6: Path classes for the Multi-cost simulation environment (soft) using the robot (hard) shown (a) with the swept volume of the robot shown in red (b) lowest deformation path: length = 73, p = 0.7, deformation = 81, (c) medium deformation path: length = 58, p = 0.01, deformation = 159, and (d) highest deformation path: length = 57, p = 0.0, deformation = 310.

2) Testing: Using the triple-wall environment, we have run the planner for all combinations of hard and soft properties and values of p ranging between 0 and 1. Three distinct examples of these paths can be seen in Figure 5. Notably, only very low values of p, such as $p = 0.01$ result in *any* incurred deformation for this test environment. This is due to the imbalance between cost incurred due to path length and cost of deformation – the optimal non-deformation path through the environment has a length of approximately 94, while a deformation-causing path of length 61 incurs a deformation cost of 1062. In contrast, the multi-cost environment lacks a deformation-free path. As before, we have tested the planner with a range of p values. Several examples of paths with corresponding values of p used to produce them can be seen in Figure 6.



(a) (b) (c) (d) (e)

Fig. 7: (a) Vision system for deformation tracking, (b) the three Pareto-optimal path classes, (c),(d),(e) execution of the shortest path through the physical test environment, as seen by the forearm camera of the PR2.

3) Performance: For each combination of hard and soft environment and robot, we ran our discrete planner 101 times on each environment, increasing p from 0.0 to 1.0 in 0.01 increments. Planning times for the two simulation environments (triple-wall and multi-cost) average approximately 14.4 seconds and 2.3 seconds, respectively, for these experiments. A notable outlier in terms of run time exists for very low values of p in cases with soft environments and/or soft robots, in which a solution is found in under 2 and 1 seconds, respectively, because the cost of deformation is effectively ignored in these cases. In general, while computation of deformation adds comparatively little overhead to the state evaluation process, planner performance is worse with fully deformable environments or fully deformable robots with p values above zero, as there are no longer any *infeasible* states that can be eliminated as would be the case with rigid environments. As a result, there are more states to consider.

B. PR2 Testing

We have also built a test environment to evaluate our cost function and demonstrate the performance of our planner when applied to a physical environment. This test environment is similar in concept to the triple-wall simulation environment discussed already, containing a hard robot and soft obstacles, which has been simplified to allow assessment of deformation. As full tracking of deformation in the physical world is currently very difficult to accomplish, our test environment allows us to sense physical deformation.

1) PR2 Implementation: Our test environment consists of obstacles built out of deformable foam blocks on a rigid backing. These blocks are attached to the backing to prevent rotation or movement without affecting the deformable behavior of the blocks' exterior (see Figure 7(a)). For planning purposes, the test environment is represented at 5mm resolution, as this offers a balance between accuracy and fast planning times. Planning times for the test environment average 1.2 seconds with a maximum of 1.7 seconds, requiring the evaluation of at most 4818 unique states.

To represent the simulated robot in our test environment, we use the red cylinder shown in Figure 7(b) which is moved through the environment by a PR2 robot. For our tests, paths planned in the simulator are converted into pose-space trajectories in the PR2's reference frame. Using the provided inverse-kinematics software, we convert these pose-space trajectories to joint-space trajectories which are then executed using the PR2's provided joint trajectory controllers, while the base of the robot remains at a fixed location.

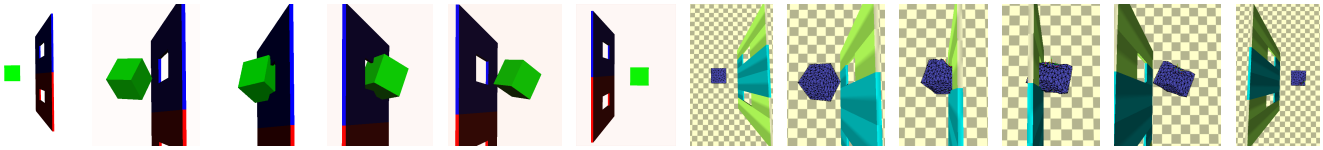
2) Deformation Tracking: To assess the accuracy of our cost function, we use a vision-based tracking system shown in Figure 7(a) to provide ground-truth values for deformation. This vision system consists of a camera mounted above the test environment which measures the visible deformation of the foam environment. Our test environment is specifically designed so that only the areas made up of deformable foam are visible to the camera, as these are the only areas in which deformation may occur.

3) Testing: Using three Pareto-optimal paths produced by the planner shown in Figure 7(b), we executed each path 12 times through the environment with six executions in either direction; snapshots from the execution are shown in figures 7(c-e). Notably, the deformation-free path of p = 0.5 caused a non-zero measured deformation – this was due to A* planning paths that closely follow the shape of obstacles, which results in interaction between the surface of the foam environment and the plastic object. Overall, however, measured deformation and observed behavior of the foam strongly correlates to that expected from the planner.

4) Calibration: An additional role of the deformation tracking system is to calibrate the costs returned by the planner's cost function to the costs measured by the tracking system. To calibrate, we calculated the ratio between planned and measured deformation (in pixels) for each point of the three paths, and used the mean of this data to scale the cost computed by our cost function. For the two Pareto-optimal paths in Figure 7(b) with planned deformation, the planner computed cumulative path costs of 581 and 1240 in units of our cost function. We measured cumulative path deformation of 19700 and 56200 in units of pixels. Applying our calibration, the planner costs are 23000 and 51000 pixels, respectively, which are within 17% and 9% of the measured values. Inaccuracy in the calibration occurs at points of starting and ending deformation; we believe this results from the discretization of the planner and the material properties of our foam test environment.

C. Higher-dimensional Planning

To demonstrate the suitability of our representation for higher-dimensional problems that cannot be reasonably addressed with discrete planning approaches, we use the T-RRT and GradienT-RRT planners in the environment shown in Figure 8.



(a) (b) (c) (d) (e) (f) (g) (h) (i) (j) (k) (l)

Fig. 8: 6-dimensional planning: (a-f) Execution of a planned trajectory in the OpenRAVE environment used for planning, (g-l) execution of the same trajectory in the Bullet physics simulator.

1) Environment: In this environment, the vertical wall is rigid and the cube robot is deformable. The vertical wall is largely symmetric; however, the hole in the blue half is 12.5% larger and thus presents a lower-cost path. The robot has 6 degrees of freedom (translation and rotation), but is neither capable of moving around the wall nor passing completely through the wall, as the very center of the robot is rigid. Rather, a valid solution must pass through one of the two holes, both of which are smaller than the robot. These holes pose two particular challenges to the planning algorithm as each presents both a "narrow passage" and a "cost-space chasm" [6] – i.e. a narrow area of low cost.

2) Testing: Using the aforementioned testing environment, we can produce paths such as those shown in Figure 8. As already discussed, the GradienT-RRT planning algorithm is especially designed to navigate the low-cost C-space region resulting from the hole in the wall, however, neither it nor T-RRT is particularly designed to address the problem of *entering* this region.

In our initial tests, both T-RRT and GradienT-RRT failed to compute any paths given a reasonable time limit. As a workaround, we added virtual padding to the obstacle when creating the discretized cost-space. This effectively increases the volume of the obstacle and produces a smoother gradient instead of a sharp boundary. Counter-intuitively, while this padding makes the "narrow passage" of entry narrower, it makes the *volume* of the cost-space chasm larger, which increases the probability of sampling within it. Notably, it also increases the area of "useful gradients" that push the algorithm towards the hole. For paths such as that shown in Figure 8, padding increased the length of the cost-space chasm from 0.1m to 0.7m for both holes.

	T-RRT		GradienT-RRT	
nFailMax	10	20	10	20
Planning time (s)	112(45.6)	736(260)	20.6(10.7)	175(69.4)
Planned cost	515(82.3)	478(56.9)	714(326)	563(164)
Calib. cost (m^3)	3.04(0.48)	2.82(0.33)	4.21(1.90)	3.32(0.96)
Sim. cost (m^3)	2.97(1.46)	2.63(1.28)	4.10(2.04)	3.48(1.29)

TABLE I: Performance data [mean(std. dev.)] for T-RRT and GradienT-RRT planners. Cost given is the integral of costs incurred at each state in a trajectory.

3) Performance: We ran both planners 30 times in our test environment, each with $nFailMax = 10$ and $nFailMax = 20$. The results of these trials are shown in Table I. GradienT-RRT produced solutions in all 30 trials for both values of $nFailMax$, while T-RRT failed to find a solution in 1200 seconds for 8 of 30 trials with $nFailMax = 20$. GradienT-RRT is significantly faster than T-RRT with the same parameters, however, T-RRT produces lower-cost paths and always traverses the lower-cost hole when it returns a solution. At $nFailMax = 10$ and $nFailMax = 20$, GradienT-RRT planned paths through the higher-cost hole seven and one times, respectively. The high standard deviation for the cost produced by GradienT-RRT is the result of these paths through the higher-cost hole.

As is clearly visible in Table I, T-RRT and GradienT-RRT work best with different values of $nFailMax$. GradienT-RRT, by virtue of "greedily" following the cost-space gradient, requires a higher value (e.g. 20) to discourage planning through the higher-cost hole. T-RRT, on the other hand, requires longer planning times for a given $nFailMax$ but produces lower-cost paths than GradienT-RRT with the same parameters. Notably, both planners produce "corner-first" trajectories for the cube, demonstrating that our planners take advantage of the rotational degrees of freedom to reduce deformation.

D. Simulator Validation

Given the difficulty of assessing real-world deformations, our simulation environment provides an alternative to real-world testing – albeit one limited by the accuracy of the simulator. To assess the paths produced with the T-RRT and GradienT-RRT planning algorithms, we have implemented a validation environment using the Bullet physics simulation engine to match the OpenRAVE planning environment shown in Figure 8. This validation environment provides soft-body physics to simulate the physical interactions between the deformable cube and the wall. The deformable robot is modelled using a tetrahedral mesh anchored to a small rigid body. This setup allows the soft robot to be "towed" along the path produced by the planner. Deformation is assessed by computing the current volume of each tetrahedron in the tetrahedral mesh (in m^3) and comparing the total volume against that of an undeformed reference mesh.

Our calibration strategy is similar to that used previously in Section IV-B.4. We calibrated the raw costs returned by the planner for each of the 112 trajectories to produce the calibrated cost values shown in Table I. Discrepancies between the calibrated planner cost and the simulator cost are largely due to "mangling" of the deformable cube – i.e. it does not return to its original shape after deformation, which is an artifact of the Bullet simulator. This effect can be seen in Figure 8(l).

E. Representation Performance

Cost assessment using our cost function offers significant performance improvements over cost assessment using physical simulation. A single cost assessment for the deformable cube discussed in Section IV-C, modelled with 1000 voxels in our representation, takes an average of 84 microseconds, regardless of the state of the deformable object. In comparison, the same cost assessment done using the Bullet physics simulator (see Section IV-D), with the cube modelled with 400 tetrahedrals, takes an average of 4 milliseconds when the cube is in contact with a rigid obstacle, and 16 milliseconds when the cube is in contact with a deformable obstacle. Not only is using our representation significantly faster (almost 50 times so in hard-on-soft and 200 times so in soft-on-soft), but Bullet at these settings–tuned for a balance between simulation quality and speed–exhibits severe mangling and distortion of the deformable object during and after deformation. Tuning parameters in favor of higher simulation quality results in an even greater performance gap, while tuning them in favor of faster simulation results in prohibitively poor simulation quality.

V. Conclusions

We have proposed a new method of representing deformable objects that allows both physical and qualitative properties to be captured in a voxel-based representation. Using this representation, we have designed a cost function that directly assesses the severity of deformation without expensive physical simulation or computation of deformed geometry. This cost function is particularly suitable for motion planning, and we have demonstrated its application to both discrete motion planning in low dimensions and sampling-based motion planning in higher dimensions. We show that our methods can generate paths that minimize deformation in both simulated and physical environments with either hard and soft robots in either hard and soft environments. In addition, using both a physical test environment and a soft-body simulation environment, we have demonstrated methods for calibrating our object representation to match observed object behavior.

References

[1] S. F. F. Gibson and B. Mirtich, "A survey of deformable modeling in computer graphics," Mitsubishi Electric Research Laboratories, Tech. Rep., 1997.

[2] M. Müller, J. Dorsey, L. McMillan, R. Jagnow, and B. Cutler, "Stable real-time deformations," in *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ser. SCA '02, New York, New York, USA, 2002.

[3] G. Irving, J. Teran, and R. Fedkiw, "Invertible finite elements for robust simulation of large deformation," in *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ser. SCA '04, New York, New York, USA, 2004.

[4] F. Faure, B. Gilles, G. Bousquet, and D. K. Pai, "Sparse meshless models of complex deformable solids," *ACM Transactions on Graphics*, pp. 73–73, 2011.

[5] L. Jaillet, J. Cortes, and T. Simeon, "Sampling-Based Path Planning on Configuration-Space Costmaps," *IEEE Transactions on Robotics*, vol. 26, no. 4, pp. 635–646, Aug. 2010.

[6] D. Berenson, T. Simeon, and S. S. Srinivasa, "Addressing cost-space chasms in manipulation planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[7] R. Alterovitz and K. Goldberg, *Motion Planning in Medicine: Optimization and Simulation Algorithms for Image-Guided Procedures (Springer Tracts in Advanced Robotics)*. Springer, 2008.

[8] P. Couvignou, N. Papanikolopoulos, and P. Khosla, "Model-based robotic visual servoing," in *American Control Conference (ACC)*, vol. 1, 1995.

[9] P. A. Couvignou, N. P. Papanikolopoulos, M. Sullivan, and P. K. Khosla, "The use of active deformable models in model-based robotic visual servoing," *Journal of Intelligent Robotic Systems*, vol. 17, no. 2, pp. 195–221, 1996.

[10] F. Barbagli, K. Salisbury, and D. Prattichizzo, "Dynamic local models for stable multi-contact haptic interaction with deformable objects," in *11th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, 2003.

[11] V. S. Arikatla and S. De, "A two-grid iterative approach for real time haptics mediated interactive simulation of deformable objects," in *IEEE Haptics Symposium*, Mar. 2010.

[12] B. Maris, D. Botturi, and P. Fiorini, "Trajectory planning with task constraints in densely filled environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct. 2010.

[13] M. Saha and P. Isto, "Motion planning for robotic manipulation of deformable linear objects," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2006.

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

[15] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," Tech. Rep., 1998.

[16] E. Anshelevich, S. Owens, F. Lamiraux, and L. Kavraki, "Deformable volumes in path planning applications," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 3, 2000.

[17] O. Burchan Bayazit and N. Amato, "Probabilistic roadmap motion planning for deformable objects," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2002.

[18] R. Gayle, M. Lin, and D. Manocha, "Constraint-Based Motion Planning of Deformable Robots," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2005.

[19] M. Moll and L. Kavraki, "Path planning for deformable linear objects," *IEEE Transactions on Robotics*, vol. 22, no. 4, pp. 625–636, Aug. 2006.

[20] S. Rodriguez and N. Amato, "Planning motion in completely deformable environments," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2006.

[21] B. Frank, C. Stachniss, N. Abdo, and W. Burgard, "Efficient motion planning for manipulation robots in environments with deformable objects," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sep. 2011.

[22] P. Hart, N. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[23] C. Hallam, K. Harrison, and J. Ward, "A Multiobjective Optimal Path Algorithm," *Digital Signal Processing*, vol. 11, no. 2, pp. 133–143, Apr. 2001.

[24] S. Karaman and E. Frazzoli, "Sampling-based Algorithms for Optimal Motion Planning," *The International Journal of Robotics Research*, vol. 30, no. 7, p. 20, 2010.

[25] R. Iehl, J. Cortés, and T. Siméon, "Costmap planning in high dimensional configuration spaces," in *IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, no. Section II, 2012.

[26] D. Devaurs, T. Siméon, and J. Cortés, "Enhancing the Transition-based RRT to Deal with Complex Cost Spaces," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2013.

[27] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2, 2009.

[28] R. Diankov and J. Kuffner, "Openrave: A planning architecture for autonomous robotics," Robotics Institute, Pittsburgh, PA, Tech. Rep., 2008.

[29] E. Coumans, "Bullet 2.73 Physics SDK Manual," p. 47, 2010.