Space-time Area Coverage Control for Robot Motion Synthesis

In: Proc. 17th IEEE International Conf. on Advanced Robotics (ICAR 2015), Istanbul, Turkey (2015).

Space-time Area Coverage Control for Robot Motion Synthesis

Vladimir Ivan, Sethu Vijayakumar
School of Informatics, University of Edinburgh, Edinburgh, UK
Email:{v.ivan, sethu.vijayakumar}@ed.ac.uk

***Abstract*—We propose a novel method for representing the interaction of a robot and an object. We create a virtual surface by taking a chain of linear segments attached to the robot links, we spatially extrude them in time, and we then compute the coverage of this surface around the object. Our approach uses a technique based on computation of electric flux, borrowed from electro dynamics. The advantage of using this method is that it is invariant to the relative transformations of the virtual surface, which makes it suitable as a complementary term in a cost function when constructing a multi-objective problem. We demonstrate the different types of interactions this method can represent, and how it can be integrated into trajectory optimisation based motion planners. We also demonstrate a practical application of such representation on a real robot.**

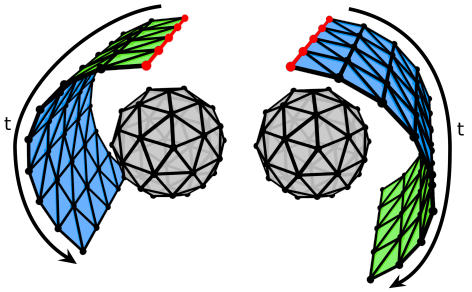
I. Introduction

Robotic systems are gaining popularity as tools in academia, industry, and every day life, with applications such as manufacturing, assembly, construction, mining, health monitoring, packaging, and agriculture. The range of tasks that robots in these areas are programmed to do is wide, but the tools and techniques that are available to define such tasks is fairly limited, i.e. tracking pre-programmed end-effector position and orientation, or minimising control effort. In this paper, we present a method which we use to define a kinematic task based on controlling the amount of wrapping of the surface that the moving links of the robot sweep along during the motion around an object of interest. Figure 1 illustrates what would such a surface look like and what types of motion can be achieved by deforming this surface into different shapes. This method is complementary to the existing tools and it can be used as an additional constraint or as a cost term in a control algorithm.

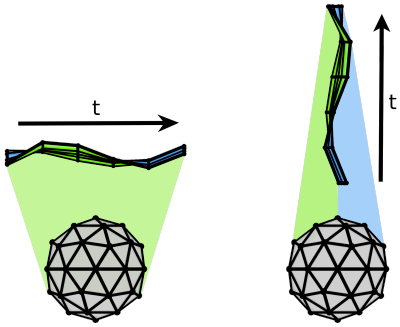
Controlling the amount of wrapping of the surface that the robot's links move across has several applications, such as: wiping the surface of an object (polishing), covering the surface area with a spray tool (painting or spraying cement), scanning complex object (checking structural fidelity) and wrapping objects with flexible materials (packaging using cling film or assisting people with putting clothes on [1]). Some of these tasks can be defined by end-effector trajectories or relative force profiles [2]. The area that the surface in Figure 1 covers is a measure that generalises over finer geometric detail. This allows us to transfer and adapt the planned motion to objects of different shapes, or to exploit this invariance to improve robustness with respect to geometric inaccuracies that may arise from imprecise sensing.

We propose to use an analogy to the well studied physical

978-1-4673-7509-2/15/$31.00 ©2015 Crown



(a) Negative side coverage (b) Positive side coverage



(c) Large amount of positive coverage (d) Zero coverage

Figure 1: Different configurations of a virtual surface with a positive side (green) and negative side (blue) constructed by spatially extruding a chain of linear segments (red line) in time. (a) and (b) show negative and positive side covering. To visualise the concept of coverage in (c) the large positive (green) area dominates the interaction, while in (d) the positive and negative contributions negate each other.

property of electric flux [3] to compute the area that an open or closed surface covers around an object of interest. The method proposed here exploits this parametrisation in the context of computing coverage through flux of a chain of linear segments attached to the robot links spatially extruded in time. We use the kinematics of the robot to define virtual points attached to the robot links, e.g. joint positions. We then track the position of these points over time and create a virtual surface constructed through triangulation. Our previous work [4] has shown that the electric flux space has very few local minima, which makes it a suitable task space for local optimisation methods. To synthesise robot motion, we use the Approximate Inference Control, a local trajectory optimisation

method, proposed in [5], which we use to optimise the shape of the virtual surface to achieve a prescribed amount of electric flux through it.

II. Related work

Coverage has been studied in the literature using tools from both geometry and topology. In [6], the authors proposed a geometrical methods for computing areas on the curved surface of an object which are suitable for grasping by caging the object using the gripper. Assuming that a caging grasp is possible only within a sphere of a certain radius, by varying this radius the curvature of the surface is analysed at different scales, and the stability of the grasps is verified using physics simulation. The resulting surface map is then used for motion planning. A different method for controlling the movement of flexible objects (such as clothes and ropes) on the surface of a rigid object using geodesics has been proposed in [1]. As opposed to these geometric methods, topological tools have been used to compute coverage of sensor networks by [7]. These tools allowed the authors to compute the coverage without the explicit knowledge of the geometry of sensor network and to provide guarantees about the sensor coverage over a particular area. Our method aims to compute coverage without the reliance on the fine geometric detail of the environment, same as the approach used for computing the sensor network coverage, but at the same time, our representation has to capture enough geometric information to allow us to control the motion of the robot.

A class of representations which are invariant to geometrical detail but capture the topology of punctured euclidean spaces has been studied in [8]. The representations discussed here have been also applied to solve robotics and computer animation problems, such as path planning with winding constraints using the winding numbers [9], character motion control preserving spatial relationships using the Gauss Linking Integral [10], and capturing the relationship between two objects by parametrizing the space around one of these objects using the analogy of the electric field and computing position within this field [4] and its integral through the surface of the second object [3]. These methods capture the relationship between objects by describing how much one object wraps around another while ignoring the finer geometric detail. Our method builds on top of this work but we redefine the way we construct the manipulated objects.

In our previous work [11], we have shown the effects of choosing an alternate space for motion planning. It is often possible to exploit a task representation, such as the one we propose in this paper, which renders a complex motion in the joint space as a simple, linear motion in the alternate space. As a result, using local motion planning methods is then sufficient for solving complex problems that would otherwise require global planning methods (such as RRTs [12]). Local trajectory optimisation techniques, such as iLQG [13] can therefore be employed to solve complex tasks by exploiting the alternate representations. The author of [5] proposed to formulate the trajectory optimisation as an probabilistic inference problem and solve it using a message passing algorithm. This formulation introduces the idea of treating the alternate representation as a Bayesian prior which expresses our belief that successful trajectories will be smoother and more linear in this space.

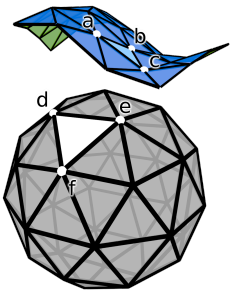


Figure 2: Virtual surface (blue/green) with a triangle $\triangledown abc$ and a virtually charged object (grey) with a triangle $\triangledown def$.

We exploit this idea and argue that trajectories that involve wrapping and scanning motion are more likely to be successful in the space of space-time area coverage.

III. Computing area coverage using analogy to electric flux

We propose to measure the area that a 2D surface covers around an object in 3D. To achieve this, we use an analogy to physical property of electric flux. Given a charged object surrounded by an electric field, the electric flux is defined as the rate of flow of this field through a given area of a virtual surface [14]. When the virtual surface is a closed Gaussian surface, the electric flux Φ_E can be expressed as an integral over the virtual closed surface S:

$$\Phi_E = \oiint E \cdot dS = \frac{Q}{\epsilon_0}, \tag{1}$$

where E is the electric field, Q is the total electric charge inside of the surface S, and ϵ_0 is the permitivity of free space. This relation is also known as the *Gauss' law*. The key property of the Gauss' law for the purpose of computing the area coverage is that the surface surrounding the charged object will always have constant flux, regardless of its deformation or transformation. This allows us to control the coverage independently of the distance from the virtually charged object. For brevity, we will refer to the virtual electric flux only as *flux*.

In practice, we are interested in computing the flux of arbitrary 2D surfaces modeled using triangulation. Such method was first proposed in [15] and later used in [3] to compute the flux of a deformable surface. Given a triangulated model of the virtually charged object and the triangulated open or closed surface in the vicinity of the charged object, we define the approximate electric flux through triangle $\triangledown abc$ (see Figure 2) due to the uniformly charged triangle $\triangledown def$ as:

$$\Phi_E(\triangledown abc, \triangledown def) = \frac{|\vec{de} \times \vec{df}|}{2} \sum_{i=1}^{4} g(\vec{x}_i, \triangledown abc), \tag{2}$$

$$\vec{x}_1 = \frac{4\vec{d} + \vec{e} + \vec{f}}{6}, \vec{x}_2 = \frac{\vec{d} + 4\vec{e} + \vec{f}}{6},$$

$$\vec{x}_3 = \frac{\vec{d} + \vec{e} + 4\vec{f}}{6}, \vec{x}_4 = \frac{\vec{d} + \vec{e} + \vec{f}}{3}$$

where $g(\vec{x}, \triangledown abc)$ is the electric flux through triangle $\triangledown abc$ due to charged point $\vec{x}$ defined as[1]:

$$\begin{aligned} g(\vec{x}, \triangledown abc) =& 2\,\mathrm{atan2}(J, K), \\ J =& (\vec{ax} \times \vec{bx}) \cdot \vec{cx}, \\ K =& |\vec{ax}||\vec{bx}||\vec{cx}| + \vec{ax} \cdot \vec{bx}|\vec{cx}| \\ &+ \vec{ax} \cdot \vec{cx}|\vec{bx}| + \vec{cx} \cdot \vec{bx}|\vec{ax}|. \end{aligned} \tag{3}$$

Each triangle ($\triangledown def$) of the virtually charged object contributes to generating the electric field around the object, and each triangle of the virtual surface contributes to the total flux. We compute the total flux using superposition. Because of this, the flux computation has the complexity of $O(lm)$, where l is the number of triangles of the virtually charged object and m is the number of triangles of the virtual surface. The flux formula and its Jacobian are ideal for implementation on parallelized systems, such as GPUs, due to the independent contribution of each triangle to the total flux.

We refer to the object from which we draw triangles $\triangledown def$ as virtually charged object. This analogy to objects generating an electric field is useful for visualising the concept of the virtual electric flux, but notice that we do not compute any actual electrical charges on the the surface of the object, apart from assuming that each triangle is charged uniformly with an unit charge.

If the points a, b, c are attached to a kinematic structure of the robot and controlled via joint angles $q \in \mathbb{R}^n$, then the analytical Jacobian of the flux with respect to the joint angles can be obtained using the chain rule:

$$\frac{\partial \Phi_E(\triangledown abc, \triangledown def)}{\partial q} = \frac{|\vec{de} \times \vec{df}|}{2} \sum_{i=1}^{4} \frac{\partial g(\vec{x}_i, \triangledown abc)}{\partial q}, \tag{4}$$

$$\frac{\partial g(\vec{x}, \triangledown abc)}{\partial q} = 2\frac{\frac{\partial J}{\partial q}K - J\frac{\partial K}{\partial q}}{J^2 + K^2}, \tag{5}$$

$$\begin{aligned} \frac{\partial J}{\partial q} =& (\frac{\partial \vec{ax}}{\partial q} \times \vec{bx} + \vec{ax} \times \frac{\partial \vec{bx}}{\partial q}) \cdot \vec{cx} \\ &+ (\vec{ax} \times \vec{bx}) \cdot \frac{\partial \vec{cx}}{\partial q}, \end{aligned} \tag{6}$$

$$\begin{aligned} \frac{\partial K}{\partial q} =& \frac{\partial |\vec{ax}|}{\partial q}|\vec{bx}||\vec{cx}| + |\vec{ax}|\frac{\partial |\vec{bx}|}{\partial q}|\vec{cx}| + |\vec{ax}||\vec{bx}|\frac{\partial |\vec{cx}|}{\partial q} \\ &+ (\frac{\partial \vec{ax}}{\partial q} \cdot \vec{bx} + \vec{ax} \cdot \frac{\partial \vec{bx}}{\partial q})|\vec{cx}| + \vec{ax} \cdot \vec{bx}\frac{\partial |\vec{cx}|}{\partial q} \\ &+ (\frac{\partial \vec{ax}}{\partial q} \cdot \vec{cx} + \vec{ax} \cdot \frac{\partial \vec{cx}}{\partial q})|\vec{bx}| + \vec{ax} \cdot \vec{cx}\frac{\partial |\vec{bx}|}{\partial q} \\ &+ (\frac{\partial \vec{cx}}{\partial q} \cdot \vec{bx} + \vec{cx} \cdot \frac{\partial \vec{bx}}{\partial q})|\vec{ax}| + \vec{cx} \cdot \vec{bx}\frac{\partial |\vec{ax}|}{\partial q}, \end{aligned} \tag{7}$$

$$\frac{\partial \vec{ax}}{\partial q} = -\frac{\partial \vec{a}}{\partial q}, \frac{\partial \vec{bx}}{\partial q} = -\frac{\partial \vec{b}}{\partial q}, \frac{\partial \vec{cx}}{\partial q} = -\frac{\partial \vec{c}}{\partial q}, \tag{8}$$

$$\frac{\partial |\vec{ax}|}{\partial q} = \frac{\vec{ax} \cdot \frac{\partial \vec{a}}{\partial q}}{|\vec{ax}|}, \frac{\partial |\vec{bx}|}{\partial q} = \frac{\vec{bx} \cdot \frac{\partial \vec{b}}{\partial q}}{|\vec{bx}|}, \frac{\partial |\vec{cx}|}{\partial q} = \frac{\vec{cx} \cdot \frac{\partial \vec{c}}{\partial q}}{|\vec{cx}|}, \tag{9}$$

[1]We denote a triangle defined by edges a, b, and c as $\triangledown abc$, a 3D vector as $\vec{x}$, and a vector between two points as $\vec{ax} = \vec{x} - \vec{a}$.

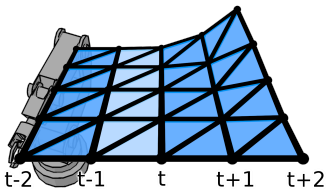


Figure 3: The virtual surface constructed from vertices attached to the robot's links at time steps $t-2$ to $t+2$ respectively (the columns). The highlighted set of triangles is used for flux computation at time t.

where $\frac{\partial \vec{a}}{\partial q}$, $\frac{\partial \vec{b}}{\partial q}$, and $\frac{\partial \vec{c}}{\partial q}$ are the 3D position Jacobians of frames attached to the kinematic structure of the robot that can be obtained using standard kinematics software. The complexity of computing the Jacobian of the flux is $O(lmn)$.

IV. Area Coverage of Space-Time Surfaces

One of the key contributions of this paper is how we compute the flux through the surface that the robot parts sweep along during the motion. We generate this surface on the fly from a series of $\widetilde{n}$ number of points attached to the kinematic structure of the robot. We do this by constructing triangles between points from two consecutive time steps (see Figure 3). Given T number of time steps, we create $2\widetilde{n}(T-1)$ triangles, when we consider the swiping surface of one moving kinematic chain. If multiple chains are considered, such as the arms of a bi-manual robot, we have the choice between defining multiple surfaces or glueing the surfaces manually with additional triangles. We decide on an order in which we connect the triangles, to ensure consistency, and also to define an orientation that the triangles are facing (see Figure 2). This orientation is important for distinguishing between the positive and the negative flux generating side of the surface. In our figures and illustrations, the positive flux is measured through the green side of the surface and negative flux is measured through the blue side.

The virtual surface constructed in this way consists of patches of triangles that connect consecutive time steps, where we assume that at the time step t only the vertices corresponding to this time step can be controlled. For this reason, we only compute flux through the patch constructed from vertices at $t-1$ and t (the highlighted patch in Figure 3). We ignore all previous and future patches because the flux through these will be accounted for when we update the subsequent time steps. We store the positions of all the vertices for all the time steps in memory, and access these by indexing into the memory storage. When computing the flux, we update the vertex positions at time t and we use the stored positions from the previous time step.

The approximation of the flux through the surface of the triangle is most accurate when the triangle equilateral does not intersect other triangles, and covers only a small area relative to the distance of the triangle to the object's surface. We therefore aim to create triangulations as dense and as uniform as possible while keeping the number of triangles as small as possible due to the computational overhead.

Having an oriented surface through which we measure the flux, we can now deform this surface to produce three kinds

of behaviour: 1) increase the flux in a positive direction (see Figure 1b), 2) increase the flux in a negative direction (see Figure 1a), and 3) maintain zero flux (see Figure 1d). The positive and negative flux changes can be interpreted as opposite wrapping motions, i.e. wrapping and unwrapping an object with the virtual surface or wrapping the object in clockwise and in the anticlockwise directions. Maintaining zero flux is a bit harder to visualise, since this involves deforming the virtual surface in such way, that it remains perpendicular to the surface of the charged object (see Figure 1d). We will demonstrate this in the third scenario of the experiment in Section V-A.

The flux through the virtual surface does not depend on transformations and deformations of the surface which don't cause wrapping (increasing the area covered). Because of this, the flux, as a task representation for motion planning, has a null space which we can utilise to satisfy additional constraints, such as joint limits, collision avoidance, or end-effector position and orientation. Additionally, the proximity of the virtual surface to the object has to be controlled using another task term, otherwise the surface may collapse to the surface of the objects. How the flux task gets combined with the other tasks depends on the choice of motion planning algorithm.

A. Trajectory optimisation with electric flux

We have described a method for computing flux through a virtual sweeping surface defined by the poses the robot passes through during its motion. To deform this surface into a shape that will achieve the desired amount of flux, we will use motion planning to compute a trajectory, or a series of commands for the robot. The space of all possible configurations is called the *configuration space*:

$$x \in C \subseteq \mathbb{R}^n, \tag{10}$$

where x is an n-dimensional robot configuration. Here, we consider continuous configuration spaces which are subsets of $\mathbb{R}^n$ because they are suitable to represent real world properties such as joint angles. For each configuration x_t, we compute the flux y_t through the virtual surface patch at time t as:

$$\Phi_E(x_t) : x_t \rightarrow y_t, \tag{11}$$

where $\Phi_E(x_t)$ is the map between the configuration space and the flux space defined in Equation 2, with the addition of computing the positions of the edges of triangles $\triangledown abc$ using forward kinematics. We store these edge position as we discussed in Section IV, which is why we omit recomputing the forward kinematics of vertices from configuration x_{t-1}. We now want to compute a trajectory in configuration space that will minimize the squared error between the desired flux y_t^* and the flux generated by the robot at time t:

$$c_t = \left|\Phi_E(x_t) - y_t^*\right|^2, \tag{12}$$

where c_t is the task cost we minimize. The cost function may contain other terms such as, collision cost, joint limits violation, e.t.c. (see Equations 13-16 for concrete examples).

A multitude of algorithms have been proposed to solve the motion planning problem [12]. Methods for optimising robot motion with respect to a cost function have been proposed, ranging from sampling based approaches [16] to optimal control [13]. In our previous work [11], we discuss how the task spaces y can be constructed in a way that improves the convergence of local optimisation methods. In other terms, we construct spaces where successful trajectories are easier to find (in case of sampling methods), shorter or local (in case of optimisation methods) in an appropriate space, thus avoiding the need for more expensive global planning methods. For this reason, we choose a trajectory optimisation method called Approximate Inference Control (AICO). AICO uses approximate probabilistic inference on a graphical model representing the dynamical system to compute successful maximum aposteriori trajectories. For more detail about this method, we refer the reader to [5] or a reinterpretation of the formalism as a generalisation of stochastic optimal control [17] or path integral control [18]. The flux representation we described in this section, and in the previous section, provides the cost term (Equation 12) and the gradient of the cost (Equation 4), which are the necessary inputs for the optimisation algorithm.

Within the AICO framework, we represent the cost term from Equation 12 using a *motion prior*. Motion prior is a term borrowed from machine learning, which we use here to express our prior beliefs about successful trajectories. This allows us to picture the choice of the flux cost term as a way of expressing our belief that the task we intend to solve will be a motion that increases or decreases the area the robot swipes along when moving around the object of interest. This belief can be expressed in different ways and virtually any motion planning algorithm could be chosen here to solve this planning problem.

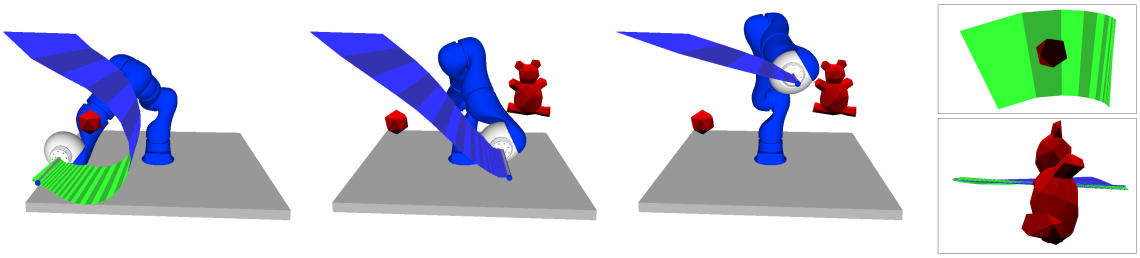
V. Experiments

A. Maximising area coverage of robot motion

The flux representation enables us to interact with objects through shaping the virtual surface we generate during the motion. In this experiment, we demonstrate several interactions that we can capture and control using the flux representation and a 7 degree-of-freedom (DoF) robotic arm. In all our experiments, we use the AICO planning framework we discussed in Section IV-A. We implement the flux task using a motion prior. How this motion prior gets computed and used for motion planning is described in [5]. However, the motion priors reflect the effects of a particular term in a cost function. Therefore, we will describe the setup of our problems using the cost function we optimise. We initialise AICO with the robot starting at a predefined starting position[2] and the initial series of commands will keep the robot in this position (zero commands). Both the choice of initial pose and the subsequent trajectory initialisation have impact on quality of the solution and the convergence rate. This is because AICO is a local optimisation method.

In the first scenario, we place a sphere in the working envelope and we define a problem of covering the area around the sphere. This could be interpreted as a task that requires scanning the object from multiple views. We define a single line segment between the penultimate joint position and the tip of the end-effector. We then construct the virtual surface by extruding this line over time. Figure 4 shows the virtual surface with the green positive side and blue negative side.

[2]We have chosen the starting pose so that the resulting virtual surface would start in a visually clear area.



(a) Single flux target (b) Opposite flux targets (c) Positive and zero targets (d) Object projection

Figure 4: Planning motion using the flux representation with: (a) positive flux target of 1.5 for a sphere, (b) positive flux target of 1.0 for the sphere and negative target of -1.0 for the teddy bear, and (c) positive flux target of 1.0 for for the sphere and zero target for the teddy bear. The robot arm starts at the top left of each figure and moves around the objects while optimised the flux values. (d) shows the projections of the large positive and close zero coverage produced in the experiment in (c).

We then set up the AICO motion priors using the following cost function:

$$c_t = \rho_E^{\text{sphere}} \left| \Phi_E^{\text{sphere}}(x_t) - 1.5 \right|^2 + \rho_o \left| \Phi_o(x_t) \right|^2 + \rho_d \left| \Phi_d(x_t) - 0.2 \right|^2, \quad (13)$$

where $\Phi_E^{\text{sphere}}(x_t)$ is the flux cost term with the desired flux value of 1.5, $\Phi_o(x_t)$ is the end-effector alignment term (keeping the end-effector parallel to the x axis), and $\Phi_d(x_t)$ is the distance terms which keeps the robot from colliding with the sphere by keeping the end-effector at a distance of 0.2m from the centre of the sphere. ρ_E^{sphere}, ρ_o, and ρ_d are the respective task precision constants, which we have manually tuned to give high precision to the flux and alignment terms but low precision to the distance term to achieve a consistent motion that is not affected too much by the distance term. Figure 4a shows the resulting shape of the virtual surface wrapped around the sphere.

In the second scenario, we add a second object and a secondary flux term with the desired flux value of -1.0, also changing the desired flux for the sphere to 1.0. We use the following cost function:

$$c_t = \rho_E^{\text{sphere}} \left| \Phi_E^{\text{sphere}}(x_t) - 1 \right|^2 + \rho_E^{\text{teddy}} \left| \Phi_E^{\text{teddy}}(x_t) + 1 \right|^2 + \rho_o \left| \Phi_o(x_t) \right|^2 + \rho_d \left| \Phi_d(x_t) - 0.2 \right|^2, \quad (14)$$

where $\Phi_E^{\text{teddy}}(x_t)$ is flux cost term for the second object. Figure 4b shows how the virtual surface passes between the two objects. The surface ends below the object on the right because both of the flux cost terms have the same weight and target value, which means that moving any further would increase the flux for one of the objects while decreasing it for the other resulting in a suboptimal solution. Additionally notice that the object on the right has a very different shape than the object on the left. This does not affect the motion because the flux representation is invariant to the shape of the object.

In the third scenario, we keep the same setup as in the second scenario but we set the desired flux target for the object on the right to zero. We use the following cost function:

$$c_t = \rho_E^{\text{sphere}} \left| \Phi_E^{\text{sphere}}(x_t) - 1 \right|^2 + \rho_E^{\text{teddy}} \left| \Phi_E^{\text{teddy}}(x_t) \right|^2 + \rho_o \left| \Phi_o(x_t) \right|^2 + \rho_d \left| \Phi_d(x_t) - 0.2 \right|^2. \quad (15)$$

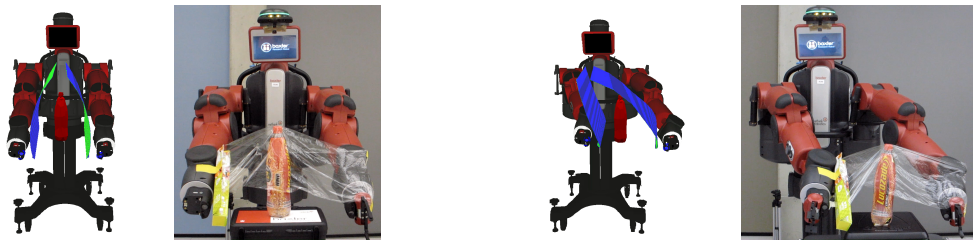
Figure 4c shows the shape of the virtual surface which maximises the positive flux around the sphere but remains flat to minimizes the flux w.r.t. the teddy bear. Figure 4d demonstrates this difference in coverage visually by viewing the virtual surface from the point of view of the objects. The top figure shows a large positive (green) area being covered, while the bottom figure shows a narrow strip with equal amount of positive (green) and negative (blue) area.

The sphere model consists of $l^{\text{sphere}} = 20$ triangles and the teddy bear model consists of $l^{\text{sphere}} = 200$ triangles. We have planned a trajectory of 50 time steps with a single line segment extruded to create the virtual surface. The virtual surface therefore consists of $m = 100$ triangles (2 triangles per linear segment). The robot arm has $n = 7$ DoF. The computational complexity of updating the flux representation is therefore $O(l^{\text{sphere}}mn + l^{\text{sphere}}mn) = O(154000)$. On a PC running a 3GHz Intel Core 2 Quad CPU the average computation time required to update the flux for both tasks for the whole trajectory is 3.5ms, when computing the flux sequentially.

B. Wrapping an object with packaging material using the flux representation

In Section V-A we showed different interactions we can represent using this representation. We will now demonstrate how flux can be applied to real world problems. For this experiment, we use the Baxter robot with two 7 DoF arms (14 controllable DoF in total). The task is to wrap an object in cling film. We attach the cling film to the arms of the robot (see Figure 5). To avoid modelling the deformation of the cling film, we assume that we can achieve successful wrapping by maximizing the area its edges cover around the bottle.

To solve this problem, we penalise for deviations of the orientations of the robot arm segments which have the cling film attached to them. We also penalise for distance from the bottle to avoid using too much wrapping material. The wrapping motion is then realised by optimising the amount of flux with a negative target for the right arm segments and



(a) Wrapping the bottle

(b) Wrapping the bottle at a new location

Figure 5: Wrapping a bottle with the cling film using flux to control the coverage. We displace the bottle between (a) and (b) to show that the spatial transformations don't affect the flux computation.

positive target for the left arm segments. Based on this, we construct the following cost function:

$$\begin{aligned} c_t = &\rho_E^{\text{right}} \left|\Phi_E^{\text{right}}(x_t)\right|^2 + \rho_E^{\text{left}} \left|\Phi_E^{\text{left}}(x_t) - 1\right|^2 \\ &+ \rho_o^{\text{right}} \left|\Phi_o^{\text{right}}(x_t)\right|^2 + \rho_d^{\text{right}} \left|\Phi_d^{\text{right}}(x_t) - 0.3\right|^2 \\ &+ \rho_o^{\text{left}} \left|\Phi_o^{\text{left}}(x_t)\right|^2 + \rho_d^{\text{left}} \left|\Phi_d^{\text{left}}(x_t) - 0.3\right|^2, \end{aligned} \tag{16}$$

where $\Phi_E^{\text{right}}(x_t)$, $\Phi_o^{\text{right}}(x_t)$, and $\Phi_d^{\text{right}}(x_t)$ are the flux, orientation and distance terms respectively for the right arm segments (the left arm segment cost terms are analogously $\Phi_E^{\text{left}}(x_t)$, $\Phi_o^{\text{left}}(x_t)$, and $\Phi_d^{\text{left}}(x_t)$).

We have optimised the motion using AICO and realised it on the Baxter robot (see Figure 5a). The robot has wrapped the bottle successfully. We then displaced the bottle and optimised the motion again. Figure 5b shows the resulting motion computed using the same cost function. This result demonstrates that to achieve the same amount of wrapping as in Figure 5a, the joint space trajectory has to be modified but the flux space trajectory remains the same. Although, there may be other ways to achieve similar motion, the flux captures the relationship of the robot motion with the bottle directly, therefore there is no need to change the task definition in the flux space when the environment changes. This experiment highlights robustness of the representation to the displacement of an object with a simple shape. However, this result generalises to more complex shapes as we have demonstrated in Section V-A.

VI. Conclusion

We have presented a method for describing the relationship between an object and a virtual surface constructed by taking chain of linear segments attached to the robot and spatially extruding them in time. We have extended the techniques used for computing the electric flux to compute the coverage of the object by the virtual surface. This method is invariant to relative transformations and deformations of the virtual surface. This property makes the representation challenging to use on its own but ideal as a complementary term in a problem definition that already constraints the distance between the robot and the object, and the orientation of the robot links. We demonstrated the capabilities of this representation as well as its practical application. In the future, we intend to extend this method for use in real time applications, where we'll exploit the flux invariants to track moving objects. We also intend to use the flux for consistent constraint learning as proposed in [2].

References

[1] H. Wang and T. Komura, "Manipulation of Flexible Objects by Geodesic Control," *Computer Graphics Forum*, vol. 31, pp. 499–508, May 2012.

[2] M. Howard, S. Klanke, M. Gienger, C. Goerick, and S. Vijayakumar, "A novel method for learning policies from variable constraint data," *Autonomous Robots*, vol. 27, no. 2, pp. 105–121, 2009.

[3] H. Wang, K. Sidorov, P. Sandilands, and T. Komura, "Harmonic Parameterization by Electrostatics," *ACM Transactions on Graphics (TOG)*, 2013.

[4] P. Sandilands, V. Ivan, T. Komura, and S. Vijayakumar, "Dexterous Reaching, Grasp Transfer and Planning Using Electrostatic Representations," in *Proc. of Humanoids*, 2013.

[5] M. Toussaint, "Robot Trajectory Optimization using Approximate Inference," in *Proc. of ICML*, pp. 1049–1056, 2009.

[6] D. Zarubin, F. T. Pokorny, M. Toussaint, and D. Kragic, "Caging complex objects with geodesic balls," in *Proc. of IROS*, pp. 2999–3006, 2013.

[7] P. Dłotko, R. Ghrist, M. Juda, and M. Mrozek, "Distributed Computation of Coverage in Sensor Networks by Homological Methods," *AAECC*, vol. 23, no. 1, pp. 29–58, 2012.

[8] S. Bhattacharya and et al., "Invariants for homology classes with application to optimal search and planning problem in robotics," *AMAI*, vol. 67, no. 3-4, pp. 251–281, 2013.

[9] P. Vernaza, V. Narayanan, and M. Likhachev, "Efficiently finding optimal winding-constrained loops in the plane," in *Proc. of R:SS*, (Sydney, Australia), 2012.

[10] E. S. L. Ho, T. Komura, and C.-L. Tai, "Spatial Relationship Preserving Character Motion Adaptation," *ACM Transactions on Graphics*, vol. 29, no. 4, pp. 33:1–33:8, 2010.

[11] V. Ivan, D. Zarubin, M. Toussaint, T. Komura, and S. Vijayakumar, "Topology-based Representations for Motion Planning and Generalisation in Dynamic Environments with Interactions," *IJRR*, vol. 32, no. 9-10, pp. 1151–1163, 2013.

[12] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[13] E. Todorov and W. Li, "A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems," in *Proc. of ACC*, pp. 300–306, 2005.

[14] D. J. Griffiths, *Introduction to Electrodynamics*. Always learning, Pearson, 2013.

[15] A. Van Oosterom and J. Strackee, "The Solid Angle of a Plane Triangle," *IEEE Transactions on Biomedical Engineering*, vol. BME-30, no. 2, pp. 125–126, 1983.

[16] O. Salzman and D. Halperin, "Asymptotically near-optimal RRT for fast, high-quality, motion planning." CoRR, 2013.

[17] K. Rawlik, M. Toussaint, and S. Vijayakumar, "On Stochastic Optimal Control and Reinforcement Learning by Approximate Inference," in *Proc.of R:SS*, (Sydney, Australia), 2012.

[18] E. Theodorou, J. Buchli, and S. Schaal, "A Generalized Path Integral Control Approach to Reinforcement Learning," *JMLR*, vol. 11, pp. 3137–3181, 2010.