Topological Methods for Motion Prediction and Caging

JOÃO FREDERICO CARVALHO

Doctoral Thesis
Stockholm, Sweden 2020

Division of Robotics, Perception and Learning
School of Electrical Engineering and Computer Science
KTH Royal Institute of Technology
SE-100 44 Stockholm, Sweden

TRITA-EECS-AVL-2020:11
ISBN 978-91-7873-450-4

Akademisk avhandling som med tillstånd av Kungl Tekniska Högskolan framlägges till offentlig granskning för avläggande av teknologie doktorsexamen i datalogi tisdagen den 17 mars 2020, klockan 10:00 i sal F3, Lindstedtsvägen 26, 114 28 Stockholm.

Tryck: Universitetsservice US AB

Abstract

To fulfill the requirements of automation in unstructured environments it will be necessary to endow robots with the ability to plan actions that can handle the dynamic nature of changing environments and are robust to perceptual errors. This thesis focuses on the design of algorithms to facilitate motion planning in human environments and rigid object manipulation.

Understanding human motion is a necessary first step to be able to perform motion planning in spaces that are inhabited by humans. Specifically through long-term prediction a robot should be able to plan collision-avoiding paths to carry out whatever tasks are required of it. In this thesis we present a method to classify motions by clustering paths, together with a method to translate the resulting clusters into motion patterns that can be used to predict motion.

Another challenge of robotics is the manipulation of everyday objects. Even in the realm of rigid objects, safe object-manipulation by either grippers or dexterous robotic hands requires complex physical parameter estimation. Such estimations are often error-prone and misestimations may cause complete failure to execute the desired task. Caging is presented as an alternative approach to classical manipulation by employing topological invariants to determine whether an object is secured with only bounded mobility. We present a method to decide whether a rigid object is in fact caged by a given grasp or not, relying only on a rough approximation of the object and the gripper.

Sammanfattning

För att uppfylla kraven för automatisering i ostrukturerade miljöer är det nödvändigt att förse robotar med förmågan att planera i föränderliga miljöer. Denna avhandling fokuserar på design av algoritmer för att underlätta rörelseplanering i mänskliga miljöer och manipulering av rigida objekt.

För att planera handlingar i utrymmen där människor rör sig är det nödvändigt, som ett första steg, att förstå hur människor rör sig. Genom långsiktiga prognoser om människors rörelser kan en robot planera undvikande av kollisioner, samtidigt som en given uppgift kan planeras. Den här avhandlingen presenterar både metoder för klassificering av rörelser, samt metoder för att använda dessa klasser för förutsägelse av rörelser.

En annan stor utmaning för robotik är manipulering av vardagliga objekt. För att manipulera rigida objekt med enkla gripdon, så väl som avancerade robothänder, är det nödvändigt att uppskatta komplexa fysiska parametrar. Sådana uppskattningar innehar ofta fel som kan leda till misslyckande med att utföra den givna uppgiften. *Caging* är ett alternativ till klassisk manipulering, där topologiska invarianter används för att avgöra om ett objekt är säkrat med endast begränsad rörlighet. Vi presenterar en metod för att bestämma om en konfiguration kan hålla ett objekt fast eller inte, som bara förlitar sig på en förenklad modell av objekt och gripdon.

Apology

It is customary to put forward an acknowledgment of all the people that have directly and indirectly contributed to the thesis work, and here I will deviate slightly from tradition, as I think an apology is both more appropriate, if not more in character. This is not that I don't acknowledge the deep contributions of those around me which culminated in the present document, but rather because I want to acknowledge that above and beyond that, this was not an easy process. I will therefore presently try to individually acknowledge those who I may have offended or inconvenienced over this period. Those I fail to acknowledge directly may consider this my first apology.

First and foremost, I have to apologize to and thank my advisor Danica Kragic as I admit that I was not an ideal student to communicate with, much less supervise. This led to the exasperation of many failed and postponed deadlines. To my co-advisors Florian and Mikael much of the same applies.

I would be remiss if I didn't now apologize to my Parents, Sisters and extended family for my silent absence. Hopefully past the end of this somewhat chaotic period I have learned to be a more present son, brother, uncle, etc...

Among all my colleagues at RPL, Anastasiia stands out as the person to most contribute directly and indirectly to the actual content of this thesis. I should therefore apologize for the quality of my code and tendency to rephrase whole paragraphs without highlighting the changes, among many other annoying habits which I hope to have slowly mitigated over the years.

To my friends, in and out of RPL (Peeter, Angela, Diogo, Silvia, Anastasiia again, Hang Kaiyu, Joshua, Mia, Francisco, Cheng, Judith, Alessandro, Sergio and Michael, you may consider yourselves explicitly named here), I must apologize for the general indecisiveness by which I threw so many a monkey wrench into so many a plan.

Finally I should extend an apology to my old flatmate Frederick, who had the patience to put up with me for three years, I owe an apology for my maybe-a-bit-too-loud protests over laundry planning.

List of Papers

This thesis is based on the following which are included in Part II:

[A] **J. Frederico Carvalho**, Mikael Vejdemo-Johansson, Danica Kragic, Florian T. Pokorny. (2018) *An Algorithm for Calculating Top-Dimensional Bounding Chains* PeerJ Computer Science.

[B] **J. Frederico Carvalho**, Mikael Vejdemo-Johansson, Danica Kragic, Florian T. Pokorny. (2018) *Path Clustering with Homology Area* Proceedings of the 2018 IEEE International Conference on Robotics and Automation.

[C] **J. Frederico Carvalho**, Mikael Vejdemo-Johansson, Florian T. Pokorny, Danica Kragic. (2019) *Long-term Prediction of Motion Trajectories Using Path Homology Clusters* Proceedings of the 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems.

[D] Anastasiia Varava*, **J. Frederico Carvalho***, Danica Kragic, Florian T. Pokorny. (in submission) *Free Space Of Rigid Objects: Caging, Path Non-Existence, and Narrow Passage Detection* International Journal of Robotics Research.

Additionally during this period, the author also contributed to the following papers which are superseeded by Paper D.

Anastasiia Varava, **J. Frederico Carvalho**, Florian T. Pokorny., Danica Kragic (2017) *Caging and Path Non-Existence: A deterministic Sampling Based Verification Algorithm* International Symposium on Robotics Research.

Anastasiia Varava*, **J. Frederico Carvalho***, Florian T. Pokorny., Danica Kragic (2018) *Free Space of Rigid Objects: Caging, Path Non-Existence and Narrow Passage Detection* Workshop on Algorithmic Foundations of Robotics.

(The $*$ indicates that the marked authors have contributed equally to every aspect of the paper.)

Contents

Part I

Introduction

Chapter 1

Introduction

The main purpose of robotics research is to develop systems capable of automating every-day tasksMuch progress has been achieved through automating factories by having large expensive actuators that are able to dextrously maneuver components. This depends on precisely modeling the environment and therefore requires that this environment be kept away from human interference in order to guard both the integrity of the process, and the safety of people. By precisely modelling every component and its placement within the assembly line, the robot's task can be greatly simplified and performed without supervision.

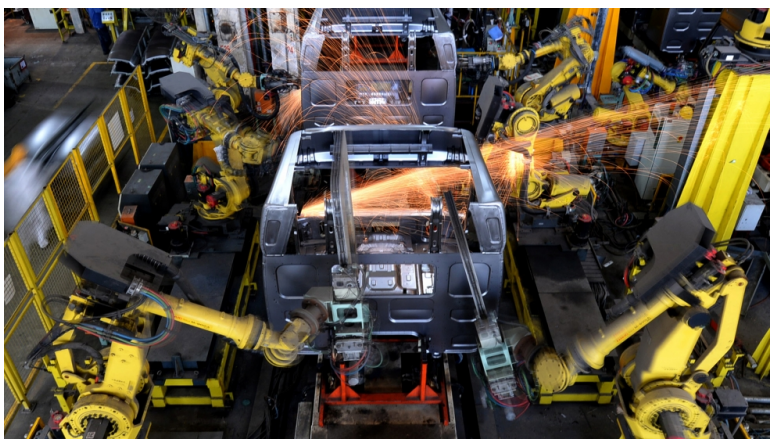


Figure 1.1: Typical robot deployment in a car factory. Due to the amounts of force employed by the actuators, human interaction with them is dangerous, and they should be kept isolated from people.

The next wave of automation will necessarily have to deal with automating tasks in environments that are both less structured, and more dynamic in nature. In

Figure 1.2 we present a high level schematic of how such a robotic system would operate in a constant feedback loop between the environment, and the planner. The robot receives input from its sensors (e.g. cameras) and devises a model of the world [77]. It then uses the model to plan its actions [52], which are then executed by the controller [76]. It has also been recognized that task should be in a feedback regime using sensory input to correct inaccuracies in the model [64–67].

Chief among the challenges in such environments are how to handle non-precise measurements from object perception pipelines, and handle cohabitation of the environment with other autonomous agents, namely humans.

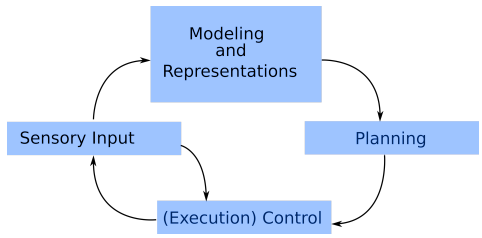


Figure 1.2: Typical robot task execution pipeline. To fulfill a task, a robot needs to be able to plan its task based on abstract models derived from its sensory inputs, and execute it while meeting constraints imposed by the environment, and the robot's own morphology. Throughout the execution process the robot may also be required to adapt to model inaccuracies.

A successful example of a robotic system that has found a wide deployment in the consumer market is the robotic vacuum cleaner. Due to the nature of its task (cover as much of the floor as possible) it does not need to reason efficiently about its environment. Furthermore, the size constraints imposed by low obstacles (such as furniture) mean that the robot needs to be small, which restricts it to being low-powered, and therefore to pose a small risk of injury to humans.

In order to overcome these limitations, and be able to deploy robots that are able to perform more complex tasks efficiently, robots need to be able to plan while taking into account the presence and movement of people in the environment. This suggests that a plan in such an environment should be reactive to the way the environment changes, and take into account the possible need for replanning during its execution [74, 75]. This replanning action due to changes in the environment, corresponds to going around the main loop in Figure 1.2 and can make the overall plan less efficient.

To mitigate the inefficiency associated to replanning it is necessary to minimize the amount of times the system replans. A natural way to do this is to predict how the environment will change and plan accordingly. However, there are limitations to how much this prediction can be modeled simply based on the layout of the environment. Therefore, it is sensible to create such models by collecting data from observations of paths in the environment.

Figure 1.3: A typical street, where pedestrians move in the sidewalks. A model that averages the movement of pedestrians on both sides of the sidewalk, may conclude that pedestrians are in fact most likely to walk in the middle of the road.

Such models cannot, however be derived from an indiscriminate set of paths. Since the possible variations on such motions are too great within the expected set of observations on any but the simplest environment, a model extracted from all possible observations does not necessarily approximate any of the actually observed paths as shown in Figure 1.3. Therefore, in order to properly model motion, it is necessary to classify such motions into coherent subsets of motions that can be modelled together. Doing so requires one to understand the space of paths in an environment. Understanding this space is a difficult task in itself, since it is arbitrarily high-dimensional, and due to the presence of obstacles lacks regularity properties such as linearity, that could potentially make it easier to study. For this reason, when studying the space of paths it is important to take into account its relation with the environment where the paths are drawn from. By using tools of geometry and topology it is possible to obtain insights into how the space of paths is laid-out, which can then be used to design algorithms for tasks such as motion prediction.

Thesis Contributions

One of the most basic properties to enquire about any space is its connectivity. Although it may be difficult to visualize how disconnectedness influences the space of paths on a base space, this can be reasoned about in terms of homotopy of paths [53]. Homotopy is difficult to calculate in practice, and therefore other tools like homology can be used as an "approximation" of homotopy. In Paper A ([1]) we describe how homology can be calculated efficiently.

Homology thus provides a way of finding the connected components of a space of paths, which can be seen as a way to classify paths that is intrinsic to the space where they reside. However, looking only at connected components, leads to too-coarse a classification, and therefore we need to adapt these methods in order to obtain a more fine-grained classification that is able to isolate classes of behaviours encoded in the path data. In Paper B ([2]) we describe how to obtain from homology a distance between paths that can be used to classify paths into categories that encode the observed behaviour.

By classifying paths it is then possible to use the path data to model the behaviour that gave rise to the paths in the first place. Such models can then be used to generate predictions of how, for example, a person who is currently traversing an environment, will be behaving in the near future. This is the problem we address in Paper C ([3]) where we provide a model of the classes of paths through a vector field representation.

Finally, a dual problem to that of understanding the space of paths, is understanding its complement. For example, if we consider the space of paths to be a subspace of a vector space[1], then the complement of the space of paths on a particular environment would be those paths that would traverse points that do not lie within the environment itself. This essentially describes the problem of path non-existence, which is addressed in Paper D ([4]) through the lens of caging and is an extension of previous works on the topic [5, 6].

[1] Consider for example that all paths may be parametrized by polynomials, which form a vector space.

Chapter 2

Background

This chapter expands on both the related work and background sections of the papers in Part II. It therefore tries to supplement the overview of the areas provided in the papers, and to give more details to the backgrounds, specifically of Papers A and B. We divide the chapter into four sections. Section 2.1 deals with the background specific to path clustering and points out issues with commonly used path distances which are alluded to in the introduction to Paper B. Sections 2.2 and 2.3 supplement the related works for Papers C and D respectively. Finally Section 2.4 expands the description of the theory of Homology presented in Papers A, B, and C with the results necessary to fully understand the proofs contained therein.

2.1 Path Clustering

Path clustering can be seen as a suite of methods for analyzing path-data with different intentions. At it's core, it can be interpreted as any other clustering mechanism, however, instead of having its data in some fixed-dimensional vector space, the dataset inhabit an infinite (read arbitrary-) dimensional vector space. Aside from dimensionality, when the paths being analyzed are based on real-world data collected from, e.g. pedestrians in a scene, further restrictions to the paths apply that make the vector space assumption into a local one at best.

Early applications of path clustering have focused on applications to surveillance and traffic analysis [7,10–13] where detection of anomalous behaviour can be reduced to the task of outlier detection [14]. Another possible field of application is learning from demonstration, where the focus is on extracting motion primitives from a set of demonstrations rather than explicit programming [15,67,71]. This is usually done by keeping a relatively small set of demonstrations that are intentionally similar to each other to allow for generalization [16,17]. Path clustering in this scenario may provide a way of recovering sets of demonstrations which can be generalized over. Finally, a related problem is that of motion prediction, which will be further explored in the next section and in Paper C where path clustering can be used as a

first step to extract motion primitives from previously observed path data.

In order to cope with the high dimensionality of the space of paths it is common to employ distance-based clustering methods. These methods simply cluster a data set of any sort by analyzing the distance matrix (that is a matrix where where entry (i, j) is the distance between data samples i and j) and therefore they only require the definition of a distance between data samples [40, 41]. A major concern about such methods is that calculating the distance between different samples should be as efficient as possible, since in order to run the clustering algorithm on a set containing N data samples, one would be required to perform N^2 distance calculations. A recent survey of path clustering using different path distances and clustering methods is available in [17]. In what follows, we provide a brief summary of commonly used path-distances.

A path distance, as the name indicates, purports to provide, given two paths, a measurement of "how far away they are from each other". Intuitively, this distance should be greater when paths are somehow more dissimilar, and if possible should satisfy the mathematical properties of a distance, i.e. if d is a distance function on a space X, then it is a binary function function $X \times X \to \mathbb{R}$ which satisfies:

1. Positivity $d(x, y) \geq 0$ for all $x, y \in X$.
2. Identity $d(x, y) = 0 \Rightarrow x = y$.
3. Symmetry $d(x, y) = d(y, x)$
4. Triangle Inequality $d(x, z) \leq d(x, y) + d(y, z)$.

The path distances here discussed, have the properties of positivity and any path distance can be made artificially symmetric (as is the case with three of the distances we will discuss briefly). However, not all path distances have identity property, or satisfy the triangle inequality. In the case of clustering, the triangle inequality guarantees a certain regularity in the spatial arrangement of clusters, and will therefore be in focus during the coming discussion.

It is important to note that we view a path P as an ordered list of points in $\mathbb{R}^d$, i.e. $P = \{p_1, p_2, \ldots, p_n\}$ (where n may vary from path to path). In the analysis we focus on whether or not individual path distances satisfy the axioms of a distance function, and the complexity of computing the distance function in terms of the length of the paths.

Euclidean Distance By far the simplest method to assign a distance to a pair of paths is to view them as finite dimensional vectors. If we take, a path $P = \{p_0, \ldots, p_n\}$ and choose a number of samples N, so that is interpolated to have the same length. The distance of the paths is then the distance between the associated vectors in $\mathbb{R}^{(d+1)N}$ where $p_i \in \mathbb{R}^d$. More concretely, given a fixed number of samples N and a path P we can define the interpolation operator $I_N(P)$ that produces an interpolated (with respect to some interpolation criteria, the most simple being

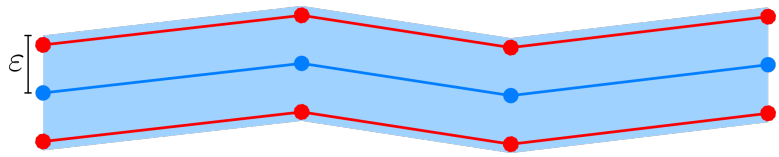


Figure 2.1: Let the paths in red be labelled r_1, r_2, and the path in blue be labeled b. The $LCSS_{\varepsilon,\delta}$ distance between b and r_i is 0 for both $i = 1$ and $i = 2$, which shows that the distance does not satisfy the identity axiom. Note also that since r_1 and r_2 run parallel to each other, and their vertical distance is larger than ε, the distance between them is $e - 1$ (the maximum distance), which shows that the $LCSS_{\varepsilon,\delta}$ does not satisfy the triangle inequality.

using a linear spline). We can then define the euclidean distance between two paths P and Q as being:

$$d_N^{Eucl}(P,Q) = \sqrt{\sum_{i=1}^{N} \|I(P)_i - I(Q)_i\|^2}$$

Note that in general, due to the use of the interpolation I, this distances does not necessarily satisfy the identity axiom.

Longest Common Subsequence (LCSS) The longest common subsequence is derived as a similarity measure rather than a distance measure. It uses two parameters ε, δ which are distance thresholds in space and time respectively. Given two paths P and Q, the similarity $\text{LCSS}_{\varepsilon,\delta}(P,Q)$ is the ratio of points in P that are in a neighborhood of Q as specified by the parameters ε, δ.

Concretely, let $N_{\varepsilon,\delta}(P,Q) = \{p \in P : \exists q \in Q, \|\Pi_{\mathbb{R}^d}(q-p)\| < \varepsilon, |\Pi_t(q-p)| < \delta\}$, where $\Pi_{\mathbb{R}^d}$ and Π_t represent the projections onto the spatial and time components of the path, respectively. Then the path similarity $\text{LCSS}_{\varepsilon,\delta} = \frac{|N_{\varepsilon,\delta}(P,Q)|}{|P|}$.

To obtain a symmetric distance function, a possibility is simply letting:

$$d_{\varepsilon,\delta}^{LCSS}(P,Q) = \exp(1 - \min\{\text{LCSS}_{\varepsilon,\delta}(P,Q), \text{LCSS}_{\varepsilon,\delta}(Q,P)\}) - 1$$

Note that to account for the asymmetry of the similarity function $LCSS_{\varepsilon,\delta}$ we take the minimum over the two ways the similarity could be calculated. Similarly to obtain a distance function from the similarity we employ a strictly decreasing function ($x \mapsto \exp(1-x) - 1$). Since the input is bounded to be between 0 and 1, then the output is between 0 and $e - 1$.

Note also that this distance function does not satisfy the axioms of identity, nor triangle inequality, and it is very simple to derive counter examples to both as shown on Figure 2.1

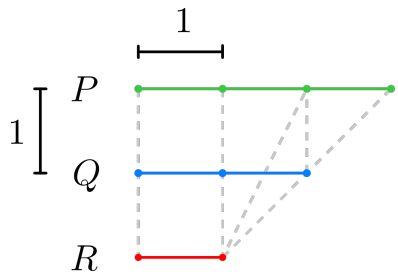


Figure 2.2: Example illustrating how the DTW path distance does not satisfy the triangle inequality. The dashed lines show the optimal time pairwise warping between paths P, Q and R. Note that $d(P,Q) = 3 + \sqrt{2}$, $d(Q,R) = 2 + \sqrt{2}$ so that $d(P,Q) + d(Q,R) = 5 + 2\sqrt{2}$. Finally the distance between P and R is $4 + 2\sqrt{2} + \sqrt{5} > d(P,Q) + d(Q,R)$.

Dynamic Time Warping (DTW) Another way to assign a distance to a pair of paths is to compute a time-warping, that is a (not-necessarily unique) order-preserving correspondence between the points in one path, to the points in the other path. Concretely, when considering a pair of paths $P = P_1, \ldots, P_K$ and $Q = Q_1, \ldots, Q_L$, a warping $w \in W(P,Q)$ is a list of pairs $(i_0, j_0), (i_1, j_1), \ldots, (i_M, j_M)$ satisfying $0 \leq i_l - i_{l-1} \leq 1$, $0 \leq j_l - j_{l-1} \leq 1$, $i_0 = 1, i_M = K, j_0 = 1, j_M = L$. Each pair of indices on a warping can be associated to the distance of the respective points in the two paths, and the Dynamic Time Warping Distance problem consists of computing the minimum sum of these point-to-point distances, i.e.:

$$d^{DTW}(P,Q) = \min_{w \in W(P,Q)} \sum_{(i,j) \in w} \|P_i - Q_j\|$$

However, note that the DTW distance does not satisfy the triangle inequality (as exemplified in Figure 2.2).

Fréchet The Fréchet distance, sometimes called the "dog-walking distance", can be summarily described in the same terms as DTW, by simply changing the objective. More concretely instead of where minimizing the sum of the point-to-point distances along a warping, it focuses on minimizing the maximum point-to-point distance within the warping, i.e.:

$$d^{Fr}(P,Q) = \min_{w \in W(P,Q)} \max_{(i,j) \in w} \|P_i - Q_j\|$$

Hausdorff The Hausdorff distance is a classical distance for sets of points, often used in topology as a measure of how disconnected two sets are. The intuition behind this distance is that two sets are close if every point in one set is close to some point in the other set:

$$H(P,Q) = \max_{p \in P} \min_{q \in Q} \|p - q\|$$

Just as we did in the case of the LCSS distance, this definition is non-symmetric, and can be made symmetric by simply considering the maximum between the alternatives:

$$d^H(P,Q) = \max\{H(P,Q), H(Q,P)\}$$

Modified Hausdorff (M-Hausd) While the Hausdorff distance provides a good measure of distance between two sets, it is generally sensitive to outliers and it does not take into account the structure of the underlying paths at all. For these reasons the modified Hausdorff distance function was introduced in [7] to cope with path data.

Computing this distance for a pair of paths $P = \{p_1, \ldots, p_n\}, Q = \{q_1, \ldots, q_n\}$ can be described in a 4-step process, and has two free parameters α and w.

- Define the path length $l_P(p_i, p_j) = \sum_{k=i+1}^{j} \|p_k - p_{k-1}\|$ and $L(P) = l(p_1, p_n)$ (with similar definitions for Q).
- For each point $p \in P$ define $C(p_i) = q_j$ so that j minimizes $|\frac{l_P(p_1,p_i)}{L(P)} - \frac{l_Q(q_1,q_j)}{L(Q)}|$, and define $N_w(p)$ to be those $q \in Q$ satisfying $\frac{l(q,C(p))}{L(Q)} < w$.
- This defines a non-symmetric distance as:

  $$\tilde{d}^{ModH}_{\alpha,w}(P,Q) = \text{ord}^{\alpha} \left\{ \min_{q \in N_w(C(p))} d(p,q) \ : \ p \in P \right\}$$

  Where ord^{α} is the α)-th quantile (i.e. the number r so that 100α percent of the elements in the set are smaller than it).
- Finally the distance is made symmetric

  $$d^{ModH}_{\alpha,w}(P,Q) = \max\{\tilde{d}^{ModH}_{\alpha,w}(P,Q), \tilde{d}^{ModH}_{\alpha,w}(Q,P)\}$$

The use of ord^{α} in the third step is what makes this distance function robust to outliers, however it also makes it not satisfy the triangle inequality as exemplified in the example in Figure 2.3.

Summary Aside from the Euclidean distance, which can be computed in a rather straight-forward manner and has essentially linear complexity, the remaining distances require some form of dynamic programming to be efficiently computed, which gives them quadratic complexity. To conclude this analysis, we can summarize the properties of different path distances in Table 2.1.

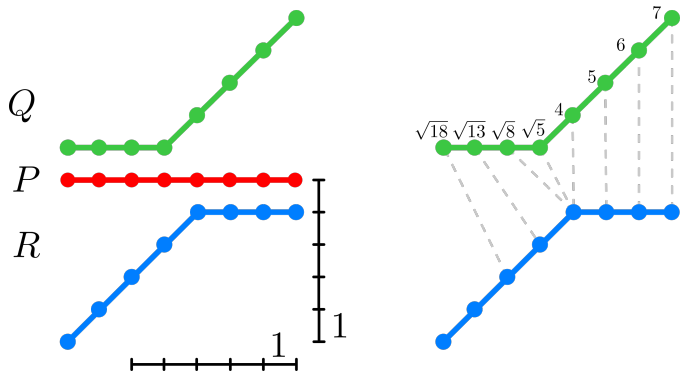


Figure 2.3: This is an example of how the modified Hausdorff distance does not satisfy the triangle inequality. Choose the parameters $w = 1$ and $\alpha = .5$ (which amounts to disregarding the ordering of the points in the computation, and considering the median distance). We can see that the closest point to any point in Q in P or R is vertically across. With that observation we can then see that $d_{\alpha,w}^{ModH}(P,Q) = d_{\alpha,w}^{ModH}(P,R) = 1$ (half the points in Q or R are at distance 1 from some point in P). Finally it can be easily calculated (by calculating the distance of every point in Q to its closest point in R and taking the median) that in these conditions $d_{\alpha,w}^{ModH}(Q,R) = 4$.

Distance	**param.**	**tri. ineq.**	**complex.**
Euclid.	N	YES	$O(\max\{N, n+m\})$
DTW		NO	$O(nm)$
LCSS	ϵ, δ	NO	$O(nm)$
Fréchet		YES	$O(nm)$
Hausd.		YES	$O(nm)$
M-Hausd.	w, α	NO	$O(nm)$

Table 2.1: Properties of classical path distances. The complexity is the worst-case runtime in number of elementary operations to calculate the distance between a pair of paths P, Q with lengths m and n respectively.

Other methods for path clustering Another avenue for performing path clustering is by employing global features, e.g. in [18] the authors apply a clustering method which clusters together paths which are in the same homology class. In a similar flavour, in [14], the authors cluster together paths that have the same beginning and end region, and lie within a certain envelope defined by the paths.

2.2 Motion Prediction

Motion prediction is an essential aspect to safe robotic navigation in dynamic and unstructured environments. Due to its importance, many different methods have been developed to perform motion prediction with application domains ranging from vehicle motion prediction [30] to predicting the full range human motion [27, 68].

In the present thesis we focus on the problem of predicting human trajectories in a 2D environment and a recent survey of the area is available at [28]. Broadly speaking, when predicting human motion there can be two types of scenes depending on how crowded they are. In crowded scenes models tend to assume that people try to move in straight lines and any deviations to this primitive occur due to obstacle avoidance [31, 32] via social forces. In less crowded scenes however, different assumptions have to be made to model path-curvature.

Typical models to deal with less crowded scenes often employ a descretization of the space and use a Hidden Markov Model (HMM) to model the motion, by employing first some form of path clustering. Examples of such methods have been used in [38] and [37] to predict human motion in an office environment and in a construction site. In [13] the authors encode motion patterns using Gaussian Mixture Models (GMM), and motion prediction is carried out using an HMM where the GMM motion patterns are are explicitly used to build the discretization. A hybrid approach is proposed by the authors of [29] by employing a Markov Decision Process (MDP) together with a social force model to account for obstacle avoidance in the model.

A perhaps simpler model was opted for in [36] using what is essentially a prediction consisting of a nearest neighbours query on a reduced dimensionality dataset.

Recent works have attempted to extract pedestrian dynamics by employing neural network architectures, such as Recurrent Neural Networks (RNN) and Graph Neural Networks [34, 35]. A related approach was taken in [39] by employing a Convolutional Neural Network (CNN) to decide which parts of a map are frequently traversed.

2.3 Path Non-Existence & Caging

Motion planning is often concerned with the problem of finding a path from a robot's starting position to some desired goal position [52]. A popular strategy to find such paths is through employing random sampling-based planners. A sampling based-planner generally works by trying to cover a part of the configuration space with a graph where the nodes are sample configurations, which have edges between them when it is possible to employ a local controller to move the robot between the two configurations. Such algorithms are often guaranteed eventually find a path if one exists, however, when one doesn't exist they will often rely on heuristics to decide when further search will be fruitless [73].

In the problem of path non-exsitence, one tries to find whether there is a disconnection in an object's collision-free space to decide whether or not it is feasible to find a motion plan from one configuration to another. This problem, often requires taking the whole configuration space into account. Due to the difficulty associated to obtaining useful approximations of the complete collision space, approaches for proving the disconnectedness of the collision-free space in the past have often focused on specific instances of the problem. For example in [59] the authors prove that a space is disconnected when the object is too big to navigate through a narrow passage. In the case of 2D rigid objects, approaches to this problem using cell decompositions of the configuration space have been proposed in [62] and [60]. By providing a cell decomposition of the collision-free space it is thus possible to verify if two configurations are disconnected or not.

An important application of the problem of path non-existence is caging [25]. Caging is the problem of partially immobilizing an object, by restricting its movement to a compact connected component of its collision-free space. Since both objects and manipulators used for manipulation tasks are bounded, all but one of the connected components of the objects collision-free space induced by the geometry of the manipulator are bounded. Therefore an object in a given configuration is caged in a given if there does not exist a path to any point in the unbounded connected component.

Relative to more classical approaches to grasping and manipulation, caging is akin to form-closure grasping. It is however less strict, in the sense that it doesn't require the object to be immobilised [19, 20]. It has thus been recognized as an intermediate step to attain a form-closure grasp [24].

In classical grasping one generally relies on estimations of friction coefficients and surface normals of the object which one intends to grasp, in order to decide whether or not a grasp is secure [21–23, 69, 70]. In contrast, caging can be formulated as a geometric problem, i.e. depending only on the shape of the object [49, 51]. Such shape features are often more robust to sensor noise and easier to estimate than for example surface normals, friction coefficients or mass density distribution which are required in classical grasping. Another advantage is that since an object that is caged cannot escape by virtue of there being no path to escape from the grasp, once such a grasp is obtained, there is no need for regrasping to react to external disturbances [72].

Historically, caging was first introduced in [42] as the problem of finding configurations of n points in the complement of a polygon and preventing it from escaping arbitrarily far from its initial position. This was later introduced in the robotics community as 2D point-based caging in many works including [43, 44, 46]. Generalizations of point-based caging to 3D were also made, for example in [45] where the authors propose caging non-convex polytopes using two fingers. And in [47] and [48] the authors use 2D-cages to drag caged objects to a given position, using mobile robots as actuators.

Generalizations of the problem to different types of actuators, or caging tools were proposed in works such as [19, 49–51]. These works take advantage of shape

features of both the caging tool and the object being caged to ensure that it is indeed caged. However, it was noted in [Makapunyo et al. (2013)] that it may not be necessary to prove that an object is caged for a given configuration to be useful for manipulation. The authors instead argue that in many circumstances it may be sufficient to use a notion of partial caging based on how unlikely it is for a motion planner to find an escaping path. A similar notion was proposed for gravity caging in [61] where an object is said to be caged if it is placed within a finite-depth gravity-well of its collision space.

Underlying both the problems of pat non-existence and caging, is the problem of efficiently approximating the collision-free configuration space. This problem has been studied a long time and a survey of early work can be consulted in [54]. An important idea for reconstructing the configuration space of rigid objects was that of decomposing the configuration space into slices along the rotational axis, which was introduced in [55] for the two-dimensional case. These slices could then be connected using the area swept by the object while moving from one orientation to an adjacent orientation. In [56] the authors extended this idea by using both outer and inner swept areas to construct a subset and a superset of the collision space of polygonal objects. These areas were represented by so-called generalized polygons, which are defined as the union or intersection of the base-polygon while rotating in a certain interval of orientation values. More recently, in [57] the authors proposed a method for computing the boundary of the collision space. Whereas in [58] the authors opted for a full reconstruction of the free space in order to perform complete motion planning.

2.4 Topological Spaces and their Simplicial Homology

To conclude this chapter we present the background in the topology of simplicial complexes that is necessary to fully understand the proofs in Papers A, B and C.

Simplicial complexes The main topological construction that we concern ourselves with is finite dimensional *simplicial complexes*. A finite dimensional simplicial complex embedded in $\mathbb{R}^n$ is a generalization of a triangulation. Formally, it consists of two components.

- A finite set of points $P \in \mathbb{R}^n$ (which are assumed to be numbered $p_1, \ldots, p_{|P|}$) which are called the *vertices* of the complex.
- A set S of subsets of $\{1, \ldots, |P|\}$ which are called the *simplices* or *faces* of the complex.

We identify a set $\sigma = \{\sigma_1, \ldots, \sigma_d\} \in S$ with the convex hull of $\{p_{\sigma_1}, \ldots, p_{\sigma_d}\}$ which we will denote $[p_\sigma] = [p_{\sigma_1}, \ldots, p_{\sigma_d}]$. For example, if $\sigma = \{1, 2\}$ then $[p_\sigma]$ is the convex hull of $\{p_1, p_2\}$, which forms a line segment, commonly referred to as an *edge* of the complex. These are required to satisfy the following axioms:

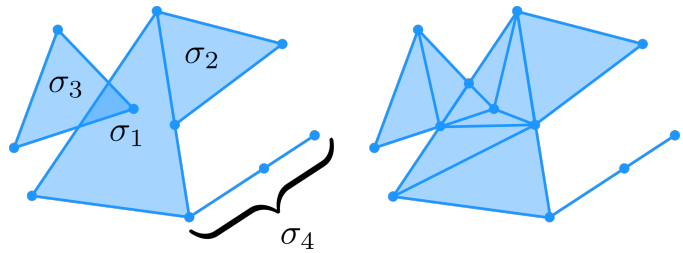


Figure 2.4: In the left we see a collection of simplices that do not form a complex because σ_1 intersects with σ_2 and σ_3 along parts of simplices instead of full simplices. Similarly the simplex σ_4 comprises 3 vertices but only has a 1-dimensional convex hull. On the right, we have an example of how the same area on the left can be covered by a simplicial complex by subdividing the simplices.

- If $\sigma \in S$ and $\tau \subseteq \sigma$, then $\tau \in S$ (all the faces of a simplex are themselves simplices).
- If $\tau, \sigma \in S$ and $[p_\sigma] \cap [p_\tau] \neq \emptyset$ then $\tau \cap \sigma \neq \emptyset$ and $[p_\sigma] \cap [p_\tau] = [p_{\sigma \cap \tau}]$ (two simplices only intersect along shared faces).
- If $\sigma = \{\sigma_0, \ldots, \sigma_k\} \in S$ then the vectors $p_{\sigma_i} - p_{\sigma_0}$ for $i = 1, \ldots, k$ are linearly independent (a simplex k points always spans a $k-1$ dimensional set).

Figure 2.4 exemplifies instances where these axioms are not satisfied. In particular the last axiom, guarantees that there is a direct correspondence between the number of vertices that span a face and the dimension of the associated convex hull. For this reason a face with $k+1$ vertices is said to have *dimension* k. A common way to decompose the set of simplices S is in *levels* $S^{(i)} = \{\sigma \in S \ : \ |\sigma| = i+1\}$ which comprise all faces the of dimension i.

These axioms make sure that the boundary of any simplex is comprised of simplices also in the complex, and therefore can be defined at every level $i > 0$ as a function:

$$\tilde{\partial}_i : S^{(i)} \to \left(S^{(i-1)}\right)^i$$

This is the main insight that is used to describe simplicial complexes algebraically via *chain complexes*.

Chain complexes By decomposing a simplicial complex $X = (P, S)$ into levels, we can concisely describe the associated *chain complex* $C_\bullet(X)$. The chain complex $C_\bullet(X)$ associated to the simplicial complex $X = (P, S)$ is a sequence of vector spaces and maps between them called *boundary maps*, which are arranged in a chain as in the following diagram:

$$C_0(X) \xleftarrow{\partial_1} C_1(X) \xleftarrow{\partial_2} C_2(X) \xleftarrow{\partial_3} \cdots$$

Where the $C_i(X) = \bigoplus_{\sigma \in S^{(i)}} \mathbb{R}$ (that is the dimension of C_i is the number of simplices in $S^{(i)}$), and the ∂_i maps are linear transformations satisfying $\partial_{i-1} \circ \partial_i = 0$. When no confusion arises we omit the index i from ∂_i.

The boundary maps represent, as the name indicates, taking the boundary of a simplex. Concretely, if we enumerate the simplices of S by a function $\iota_i : \mathbb{N} \to S^{(i)}$, denote by e_k the standard basis elements of $\mathbb{R}^n$ and by $\widehat{\sigma_{(l)}}$ the simplex $\sigma = \{\sigma_0 < \ldots < \sigma_i\}$ with the l^{th} element omitted, we can make this construction explicit via:

$$\partial_i(e_{\iota_i(k)}) = \sum_{j=0}^{i} (-1)^j e_{\iota_{i-1}^{-1}(\widehat{\sigma_{(j)}})}$$

Where we note that if we omit a vertex from a simplex σ of dimension d, the resulting set is still a simplex and it coincides with a $(d-1)$-dimensional boundary element[1]. It remains to be checked that this is a chain complex, i.e. that $\partial_{i-1} \circ \partial_i = 0$. To do so we introduce the notation $\widehat{\sigma_{(j,l)}} := \widehat{\left(\widehat{\sigma_{(j)}}\right)_{(l)}}$ for the sake of brevity, and we note that:

$$\widehat{\sigma_{(j,l)}} = \begin{cases} \widehat{\sigma_{(l+1,j)}} & \text{if } j \leq l \\ \widehat{\sigma_{(l,j-1)}} & \text{if } j > l \end{cases}$$

For conciseness denote e_τ to mean $e_{\iota_i^{-1}(\tau)}$ where i is the appropriate dimension. Now we note that the image of σ under $\partial \circ \partial$ is spanned by the elements $\widehat{\sigma_{(j,l)}}$ for $0 \leq j \leq l \leq i-1$, so we can use the usual inner product to get:

$$\begin{aligned} \left\langle (\partial \circ \partial)\, e_\sigma, e_{\widehat{\sigma_{(j,l)}}} \right\rangle &= \left\langle \partial \left((-1)^l e_{\widehat{\sigma_{(l+1)}}} + (-1)^j e_{\widehat{\sigma_{(j)}}} \right), e_{\widehat{\sigma_{(j,l)}}} \right\rangle \\ &= \left\langle (-1)^{l+1} \partial e_{\widehat{\sigma_{(l+1)}}} + (-1)^j \partial e_{\widehat{\sigma_{(j)}}}, e_{\widehat{\sigma_{(j,l)}}} \right\rangle \\ &= \left\langle (-1)^{l+j+1} e_{\widehat{\sigma_{(l+1,j)}}} + (-1)^{j+l} e_{\widehat{\sigma_{(j,l)}}}, e_{\widehat{\sigma_{(j,l)}}} \right\rangle = 0 \end{aligned}$$

Where in the last equality we use the fact that $j \leq l \Rightarrow \widehat{\sigma_{(j,l)}} = \widehat{\sigma_{(l+1,j)}}$.

Homology Since we require $\partial_{i-1} \circ \partial_i = 0$, it follows that $\text{im}(\partial_i) \subseteq \ker(\partial_{i-1})$. To measure the mismatch between $\ker(\partial_i)$ and $\text{im}(\partial_{i+1})$ we can find the classes of elements in $\ker(\partial_i)$ that differ, by an element in $\text{im}(\partial_{i+1})$, which are usually represented by:

$$B_i(X) = \text{im}(\partial_{i+1}) \quad Z_i(X) = \ker(\partial_i) \quad H_i(X) = Z_i(X)/B_i(X)$$

We call the vector spaces Z_i, B_i, H_i, the i^{th} *boundaries*, *cycles*, and *homology* of X. The main interest in homology lies in the fact that while operations that do not change the topology of X (i.e. do not create holes, or disconnect the space) may

[1] By convention, the empty set is considered a simplex of dimension -1.

change the associated group cycles, $Z_i(X)$, and the group of boundaries, $B_i(X)$, the underlying homology, $H_i(X)$, remains unchanged. Therefore homology provides a sequence of *topological invariants* of X which may be used to tell two spaces apart. Furthermore, since homology deals only with vector spaces, it is easy to implement programs that deal with it. However the reason to use homology as a tool in our work (Papers A, B and C) is its relative ease of computation (after translation, all operations are simple linear algebra), and its relation to homotopy.

Simplicial maps As with any mathematical structure, it is not just important to understand how they look, but also how they relate to other structures of the same nature via maps. The natural way of defining a map between a pair of simplicial complexes $(S, P), (S', P')$ is *simplicial maps*. These are functions $f : \{1, \ldots, |P|\} \to \{1, \ldots, |P'|\}$ such that $f(\sigma) \in S'$ for all $\sigma \in S$. Given this definition it is easy to see that $\partial(f(\sigma)) = f(\partial\sigma)$, and therefore the image of cycles and boundaries in S under the action of f, will be cycles and boundaries of S'.

Functoriality The property that the boundary operator ∂ commutes with a simplicial map f, i.e. $\partial_i \circ f = f \circ \partial_i$, makes the operation $X \to C_\bullet(X)$ into a *functor*. If we have a simplicial map $f : X \to Y$, we have a corresponding family of linear maps $f_i : C_i(X) \to C_i(Y)$ which satisfy $\partial_i f_i = f_{i-1}\partial_i$. This allows us to translate relations between simplicial complex into relations between the corresponding chain complexes.

Furthermore, the functoriality relation extends to homology, we can see this by taking an element of $c \in H_i(X)$ and considering a simplicial function $f : X \to Y$. Since $c \in H_i(X)$, it has a representative $\tilde{c} \in Z_i(X)$ and therefore $f_i(\tilde{c}) \in C_i(Y)$, furthermore because $\partial \circ f_i = f_{i+1} \circ \partial$ we have $f_i(\tilde{c}) \in Z_i(Y)$. Finally consider two representatives $\tilde{c}, \bar{c}$ of the same homology in $H_i(X)$, then they differ by a boundary $\tilde{c} - \bar{c} = b \in B_i(X)$ meaning. Let $\tilde{b} \in C_{i+1}(X)$ be such that $\partial(\tilde{b}) = b$, then $f_i(\tilde{c}) - f_i(\bar{c}) = f_i(\tilde{c} - \bar{c}) = f_i(\partial(\tilde{b})) = \partial f_{i+1}(\tilde{b})$. Which means that the f_i comprise a well defined map $H_i(X) \to H_i(Y)$.

2.5 Exactness and Chain Complex Arguments

As introduced in the previous section, a core concept of algebraic topology is sequences of spaces and maps of the form $A_1 \to A_2 \to A_3 \to \cdots$ where the kernel of a map contains the image of its predecessor. A particularly important property of such sequences is *exactness*.

Definition 1. *A sequence of maps:*

$$\cdots \leftarrow A_{i-1} \xleftarrow{g_i} A_i \xleftarrow{g_{i+1}} A_{i+1} \leftarrow \cdots$$

is said to be exact *at* A_i *if* $\ker g_i = \mathrm{im} g_{i+1}$. *Furthermore, the whole sequence is exact if it is exact at* A_i *for every* i.

Exactness allows us to intuit properties about certain spaces in the chain for example:

- If the sequence $0 \to A \xrightarrow{f} B$ is exact then f is injective, since this implies $\ker f = 0$.
- If the sequence $A \xrightarrow{f} B \to 0$ is exact then f is surjective since it implies that $\mathrm{im} f = \ker 0 = B$.
- If the sequence $0 \to A \xrightarrow{f} B \to 0$ is exact then f is an isomorphism since by the previous points, we know that f is both injective and surjective.

A particularly important type of exact sequence is the so-called *short exact sequence* which has the form $0 \to A \xrightarrow{f} B \xrightarrow{g} C \to 0$, where exactness implies f is injective and g is surjective. Such sequences figure prominently in the theory of homology, and we introduce them here to prove the existence of the "long exact-sequence of the pair" and the "Mayer-Vietoris Theorem". These theorems are extremely useful when trying to understand the homology of simplicial complexes by using simpler simplicial complexes.

Consider a simplicial complex X and a subcomplex $A \subseteq X$. This defines an inclusion map $\iota : A \to X$ which is necessarily a simplicial map. By functoriality of the chain complex operator $X \mapsto C_\bullet(X)$ it gives rise to a chain-map $\iota_* : C_\bullet(A) \to C_\bullet(X)$ which is injective at every level. In this situation, we can consider the *quotient* X/A which is formed from X by identifying every simplex fully contained in A with a single 0-simplex. Again, this is a simplicial map $j : X \to X/A$ and it induces a chain map $j_* : C_\bullet(X) \to C_\bullet(X/A)$ which is surjective at every level. With these two elements in place we see that we have a short exact sequence of chain complexes:

$$0 \to C_\bullet(A) \xrightarrow{\iota_*} C_\bullet(X) \xrightarrow{j_*} C_\bullet(X/A) \to 0$$

Meaning that for every i the associated sequence $0 \to C_i(A) \to C_i(X) \to C_i(X/A) \to 0$ is exact. In this condition, we have the following result.

Lemma 1 (Snake-lemma [9])**.** *Given a short exact sequence of chain complexes,* $0 \to A_\bullet \to B_\bullet \to C_\bullet \to 0$ *there exists a sequence of maps* $\tilde{\partial}_i : H_i(C) \to H_{i-1}(A)$ *which make the following long sequence exact:*

$$\cdots \to H_i(A) \to H_i(B) \to H_i(C) \to H_{i-1}(A) \to \cdots$$

This lemma can be used to construct a long-exact sequence from the pair of simplicial complexes $A \subseteq X$ and the quotient X/A.

Corollary 1 (Long-exact sequence of the pair)**.** *Given a simplicial complex* X *and a subcomplex* $A \subseteq X$*, we can define an exact sequence in the homology of the pair* (X, A)*:*

$$\cdots \leftarrow H_{i-1}(A) \leftarrow H_i(X/A) \leftarrow H_i(X) \leftarrow H_i(A) \leftarrow H_{i+1}(X/A) \leftarrow \cdots$$

A similar argument can be used to illuminate the structure of a union of spaces. To make this explicit consider a simplicial complex $Z = X \cup Y$ where X and Y are subcomplexes, then we can construct the following short-exact sequence of chain complexes:

$$0 \to C_\bullet(X \cap Y) \xrightarrow{\iota} C_\bullet(X) \oplus C_\bullet(Y) \xrightarrow{j} C_\bullet(X \cup Y) \to 0$$

Where the map j is constructed using the projections π_1, π_2 onto the individual components, as $j(c) = \pi_1(c) - \pi_2(c)$. And the map ι is constructed using the injections ι_1, ι_2 into the individual components as $\iota(c) = \iota_1(c) \oplus \iota_2(c)$.

To prove that this chain is exact, we just need to notice that ι is injective, which is the case since it is the sum of two injections into disjoint components of a direct sum. We need to verify that j is surjective, which again is true by construction and can be verified by noting that every basis element e of $C_\bullet(X \cup Y)$ is generated by a simplex $\sigma \in X$ or $\sigma \in Y$, in the former case it is the image of $j(e_\sigma, 0)$, and in the latter it is given by $j(0, -e_\sigma)$. Finally we need to see that the kernel of j is the image of ι, which can be directly verified by noting that $c \in \ker j$ implies that $\pi_1(c) = \pi_2(c)$ which only happens in the subspace spanned by the image of ι.

By applying the snake lemma to this short-exact sequence we prove the Mayer-Vietoris theorem:

Theorem 1 (Mayer-Vietoris Theorem)**.** *Given a simplicial complex $Z = X \cup Y$, where X, Y are subcomplexes, there is a long-exact sequence of the form:*

$$\begin{aligned} \cdots \to H_{i+1}(X \cup Y) \to H_i(X \cap Y) \to H_i(X) \oplus H_i(Y) \\ \to H_i(X \cup Y) \to H_{i-1}(X \cap Y) \to \cdots \end{aligned}$$

These results form the necessary background to fully understand the chain complex arguments used in the proofs in Papers A, B, and C.

Chapter 3

Summary of Papers

A An Algorithm for Calculating Top-Dimensional Bounding Chains

In this paper we address the problem of efficiently calculating an n-chain b that satisfies $\partial b = c$, when c is an $(n-1)$ dimensional chain, on a *manifold-like* simplicial complex of dimension n. The manifold-like restriction essentially ensures that every $(n-1)$-face of the simplicial complex has exactly (or at most) two cofaces, and that there are no d-faces with $d < n$ which aren't faces of some n-simplex.

We observe, and prove that in these conditions this problem has a solution with time complexity linear in the number of $(n-1)$-simplices. This is achieved by computing the bounding chain as a solution to N linear equations, instead of an N-dimensional linear equation (where N is the number of $(n-1)$-dimensional simplices). The main observation is that due to the manifold-like restriction, each of the N linear equations that need to be solved, involves only two variables, and

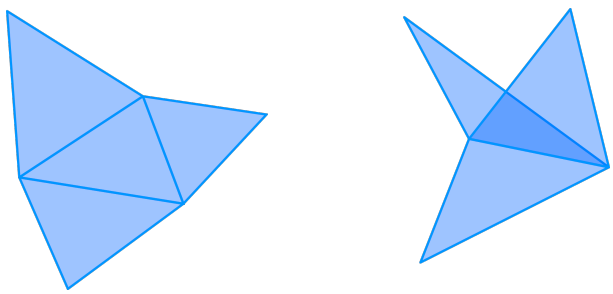


Figure 3.1: Illustration of a manifold-like simplicial complex (left), and a non-manifold-like simplicial complex. Note that the central edge on the right has three adjacent faces, while being only of dimension two (that is, the highest dimensional faces are 2D triangles).

therefore is trivially solvable if one of them is predetermined. We take into account that in certain circumstances any one simplex can be arbitrarily assigned a value, to derive the final algorithm.

Aside from the theoretical results we provide a baseline implementation of our algorithm in `C++` and compare the performance with that of computing the bounding chain as a solution to the associated sparse linear system using the implementation of *LSQR* from the `Eigen C++` library. The results of these tests empirically confirm our observations regarding the complexity of the algorithm.

B Path Clustering with Homology Area

We propose clustering paths in 2D using an unbounded path distance function between paths given by the area bounded between them. We calculate the area between them as the smallest chain whose boundary is the difference between the edge-paths associated to two paths. The complexity of the algorithm to calculate this distance for a pair of paths is relatively high (depending on the discretization of the space). However, despite this, by leveraging the properties of the distance function we can reduce the complexity of calculating a full distance matrix in two ways:

1. Since the distance function is unbounded, and satisfies the triangular inequality. Whenever we have p, q so that $d(p, q) = \infty$, we know that if $d(p, p') < \infty$, then $d(p', q) = \infty$. Essentially, this means that the distance function partitions the clustering problem into the problem of clustering smaller subsets of comparable paths within the original dataset.

2. To calculate the area enclosed by p, q we can keep track of the chains $c_{p,q}$ whose boundary is $q - p$ rather than just the area. This allows us to recover the area between q and q' as the area associated to $c_{p,q'} - c_{p,q}$. Which means we are required to compute the distance between much fewer pairs of paths, (A total of $O(Nk)$ where N is the number of paths in the dataset and k is the number of groups of comparable paths).

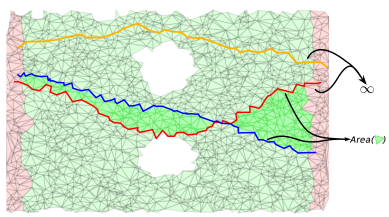


Figure 3.2: Example of how the topology of the space splits the set of paths into subsets of mutually incomparable paths. The yellow path above is incomparable to the red path, and therefore is at ∞ distance. By extension it is also at distance ∞ from the blue path. Because there are no obstacles between the red and blue paths however, they are at finite distance, given by the sum of the area of bright-green triangles.

So while this clustering method does not mitigate the complexity of calculating single distances between a pair of paths, the total amount of time is greatly reduced simply because the complexity of computing the linear combinations is much lower than of calculating the initial pair distance.

C Long-term Prediction of Motion Trajectories Using Path Homology Clusters

This paper continues the line of work started in Paper B by further refining the path-clustering method developed in it, and employing the underlying model to long-term human perform motion prediction based on the path clusters. The main ideas are as follows:

1. Take advantage of the mesh underlying the path-clustering method to create piecewise linear vector fields associated to each path cluster, based on the mesh.
2. Use the vector field to decide which cluster is most similar to a given path by verifying verifying which is the most parallel vector field to the path.
3. Employ the vector field to predict future motion by calculating an integral path to the vector field.

Further, since our method is based on simple dynamic primitives, it is conceptually simple to understand and allows one to interpret the predictions produced, as well as to extract a cost function that can be used for planning.

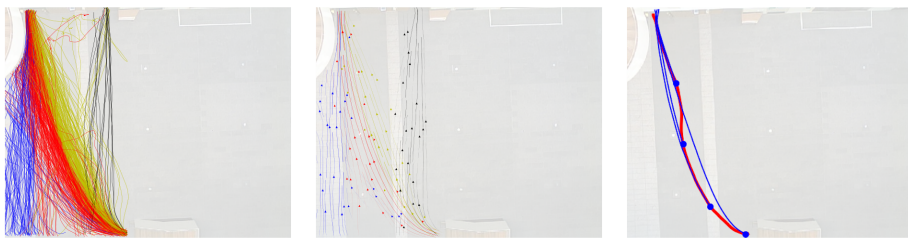


Figure 3.3: Brief overview of the motion prediction pipeline. On the left a set of paths have been clustered into different clusters, where the corresponding vector fields can be seen in the center, and the generated predictions for a given path are seen on the right.

We test our method on synthetic data and show its performance on the Edinburgh forum pedestrian long-term tracking dataset [1] where we were able to outperform a Gaussian Mixture Model tasked with extracting dynamics from the paths.

D Free Space of Rigid Objects: Caging, Path Non-Existence, and Narrow Passage Detection

In this paper we focus on the problem of finding caging grasps on rigid objects in 2D and 3D. We do so by computing an over-approximation of the collision-free space of the object and finding its connected components. By controlling how much of an over approximation we compute, we can employ the algorithm to find whether there are narrow escaping-paths from any given bounded connected component.

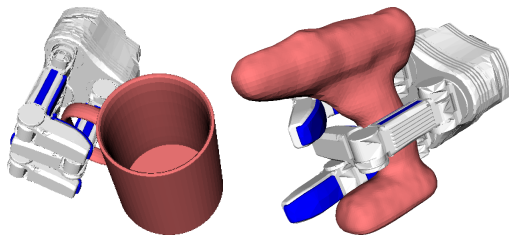


Figure 3.4: Examples of a caging grasp on a mug (left) and on a drill (right).

The algorithm employs a simple decomposition of the 3 and 6-dimensional configuration spaces of 2D and 3D objects respectively, into a Cartesian product of its rotational and translational part. Further it uses a representation of the object and obstacles as unions of balls. We then utilize a discretization of the rotational part, and at each orientation in the discretization we compute an under-approximation of the translational part of collision space. Using a diagram of balls dual to the under-approximation of the collision space as a representation of the translational free-space, we obtain an over approximation of the collision free space at a given fixed orientation. The approximations of the collision-free space at each orientation are then assembled over a neighborhood graph associated to the grid of orientations. We also describe a parallelized version of the algorithm and investigate the influence of parallelization in runtime through experiments.

We validate the performance of the algorithm in finding simple caging configurations in test objects (both synthetic and real-world), and also in applications to molecular caging.

Bibliography

[1] J. F. Carvalho, M. Vejdemo-Johansson, D. Kragic, and F. T. Pokorny. An Algorithm for Calculating Top-Dimensional Bounding Chains. In *PeerJ Computer Science*, 2018.

[2] J. F. Carvalho, M. Vejdemo-Johansson, D. Kragic, and F. T. Pokorny. Path Clustering with Homology Area. In *IEEE International Conference on Robotics and Automation*, 2018.

[3] J. F. Carvalho, M. Vejdemo-Johansson, F. T. Pokorny, and D. Kragic. Long-term Prediction of Motion Trajectories Using Path Homology Clusters. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems*.

[4] A. Varava*, J. F. Carvalho*, D. Kragic, and F. T. Pokorny. Free Space Of Rigid Objects: Caging, Path Non-Existence, and Narrow Passage Detection In *International Journal of Robotics Research* (in submission). Preprint: *arXiv:2002.02715*.

[5] A. Varava, J. F. Carvalho, F. T. Pokorny, and D. Kragic. Caging and Path Non-Existence: A deterministic Sampling Based Verification Algorithm In International Symposium on Robotics Research 2017.

[6] A. Varava*, J. F. Carvalho*, F. T. Pokorny, and D. Kragic. Free Space of Rigid Objects: Caging, Path Non-Existence and Narrow Passage Detection. In *Workshop on Algorithmic Foundations of Robotics* 2018.

[7] S. Atev, O. Masoud, and N. Papanikolopoulos. Learning Traffic Patterns at Intersections by Spectral Clustering of Motion Trajectories. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, October 2006.

[8] E. Ukkonen. Algorithms for approximate string matching. *Information and Control*, 1985, vol. 64(1-3), (pp. 100-118).

[9] S. Lang. *Algebra* Springer Verlag. 3rd Edition (2002) Graduate Texts in Mathematics, vol. 211, (pp. 157-159)

[10] B.T. Morris and M.M. Trivedi. A Survey of Vision-Based Trajectory Learning and Analysis for Surveillance. *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 18(8), August 2008.

[11] D. Makris and T. Ellis. Learning semantic scene models from observing activity in visual surveillance. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 35(3), June 2005.

[12] S. Atev, G. Miller, and N.P. Papanikolopoulos. Clustering of Vehicle Trajectories. *IEEE Transactions on Intelligent Transportation Systems*, 11(3), September 2010.

[13] L. Brun, A. Saggese, and M. Vento. Dynamic Scene Understanding for Behavior Analysis Based on String Kernels. *IEEE Transactions on Circuits and Systems for Video Technology*, 24(10), October 2014.

[14] W. Liu, X. Chong, P. Huang and N. I. Badler. Learning motion patterns in unstructured scene based on latent structural information. In *Journal of Visual Languages and Computing 25* (2014) (pp. 43-53)

[15] A. Billard, S. Calinon, Rüdiger Dillmann, and S. Schaal. Robot Programming by Demonstration. In *Springer Handbook of Robotics*. Springer Berlin Heidelberg, 2008.

[16] J. Aleotti and S. Caselli. Trajectory clustering and stochastic approximation for robot programming by demonstration. In 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, Edmonton, Alta., 2005, (pp. 1029-1034).

[17] C. Chuck, M. Laskey, S. Krishnan, R. Joshi, R. Fox, and K. Goldberg. Statistical data cleaning for deep learning of automation tasks from demonstrations. In *2017 13th IEEE Conference on Automation Science and Engineering (CASE)* 2017 (pp. 1142-1149)

[18] F. T. Pokorny, K. Goldberg, and D. Kragic. Topological trajectory clustering with relative persistent homology. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, May 2016.

[19] S. Makita, and Y. Maeda. 3D multifingered caging: Basic formulation and planning. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008. (pp. 2697-2702).

[20] S. Makita, K. Okita, and Y. Maeda. 3D two-fingered caging for two types of objects: sufficient conditions and planning. In *International Journal of Mechatronics and Automation* 2013, vol. 3(4), (pp. 263-277).

[21] J. A. Haustein, K. Hang, and D. Kragic. Integrating motion and hierarchical fingertip grasp planning. *2017 IEEE International Conference on Robotics and Automation (ICRA)* 2017, (pp. 3439-3446).

[22] K. Hang, J. A. Stork, N. S. Pollard, and D. Kragic. A Framework for Optimal Grasp Contact Planning. *IEEE Robotics and Automation Letters* 2017, vol. 2 (pp. 704-711).

[23] C. Borst, M. Fischer, and G. Hirzinger. Grasping the dice by dicing the grasp. *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)* , 2003 vol. 4 (pp. 3692-3697).

[24] A. Rodriguez, M. T. Mason, and S. Ferry From caging to grasping. In *International Journal of Robotics Research* 2012, vol. 31 (7), (pp. 886-900).

[25] A. Varava, J. F. Carvalho, F. T. Pokorny, and D. Kragic. Caging and Path Non-existence: A Deterministic Sampling-Based Verification Algorithm. In *Robotics Research. The 18th International Symposium ISRR 2017* Springer International Publishing 2020 (pp. 589-604)

[26] A. Varava, J. F. Carvalho, F. T. Pokorny, and D. Kragic. Free Space of Rigid Objects: Caging, Path Non-Existence and Narrow Passage Detection. In *The 13th International Workshop on the Algorithmic Foundations of Robotics 2018*. (to appear).

[27] J. Bütepage, H. Kjellström, and D. Kragic. Anticipating Many Futures: Online Human Motion Prediction and Generation for Human-Robot Interaction. In *IEEE 2018 IEEE International Conference on Robotics and Automation (ICRA)*.

[28] A. Rudenko, L. Palmieri, S. Herman, K. M. Kitani, D. M. Gavrila, and K. Oliver Arras. Human Motion Trajectory Prediction: A Survey. ArXiv 2019 abs/1905.06113

[29] A. Rudenko, L. Palmieri, and K. O. Arras. Joint Long-Term Prediction of Human Motion Using a Planning-Based Social Force Approach In *IEEE 2018 IEEE International Conference on Robotics and Automation (ICRA)*.

[30] A. Zyner, S. Worrall, and E. Nebot. A recurrent neural network solution for predicting driver intentions at unsignalized intersections. In *Robotics and Automation Letters*, vol. 3, July 2018.

[31] S. Kim, S. J. Guy, W. Liu, D. Wilkie, R. W. Lau, M. C. Lin, and D. Manocha. Brvo: Predicting pedestrian trajectories using velocity-space reasoning. In *The International Journal of Robotics Research* vol. 34, no. 2, pp. 201-217, 2015.

[32] A. Bera, S. Kim, T. Randhavane, S. Pratapa, and D. Manocha. GLMP - realtime pedestrian path prediction using global and local movement patterns. In *IEEE 2016 International Conference on Robotics and Automation*, IEEE, May 2016.

[33] M. Bennewitz, W. Burgard, G. Cielniak, and S. Thrun. Learning motion patterns of people for compliant robot motion. In *The International Journal of Robotics Research*, vol. 24, no. 1, pp. 31-48, 2005.

[34] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese. Social LSTM: Human Trajectory Prediction in Crowded Spaces. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, (pp. 961-971).

[35] Y. Sun, T. He, J. Hu, H. Huang, and B. Chen. Socially-Aware Graph Convolutional Network for Human Trajectory Prediction In *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, 2019 (pp. 325-333)

[36] S. Xiao, Z. Wang, and J. Folkesson. Unsupervised robot learning to predict person motion. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2015 (pp. 691-696).

[37] K. M. Rashid, and A. H. Behzadan Enhancing motion trajectory prediction for site safety by incorporating attitude toward risk In *Computing in Civil Engineering* 2017.

[38] Z. Wang, P. Jensfelt, and J. Folkesson Multi-scale conditional transition map: Modeling spatial-temporal dynamics of human movements with local and long-term correlations In *IEEE/RSJ International Conference on Intelegent Robots and Systems* 2015 (pp. 6244-6251).

[39] J. Doellinger, M. Spies, and W. Burgard. Predicting occupancy distributions of walking humans with convolutional neural networks. In *IEEE Robotics and Automation Letters*, 2018 vol. 3, (pp. 1522-1528).

[40] D. Müllner. Modern hierarchical, agglomerative clustering algorithms. In *arXiv:1109.2378*, September 2011.

[41] U. von Luxburg. A tutorial on spectral clustering. In *Statistics and Computing*, 2007 vol 17, (pp. 395-416)

[42] W. Kuperberg. Problems on polytopes and convex sets. In *DIMACS Workshop on polytopes*, 1990, (pp. 584-589).

[43] E. Rimon, and A. Blake. Caging planar bodies by one-parameter two-fingered gripping systems. In *International Journal of Robotics Research* 199,9 vol. 18(3) (pp. 299-318).

[44] P. Pipattanasomporn, and A. Sudsang. Two-finger caging of concave polygon. In *IEEE International Conference on Robotics and Automation* , 2006, (pp. 2137-2142).

[45] P. Pipattanasomporn, A. Sudsang. Two-finger caging of nonconvex polytopes. In *IEEE Transactions in Robotics and Automation*, 2011, vol. 27(2), (pp. 324-333).

[46] M. Vahedi, A. F. van der Stappen. Caging polygons with two and three fingers. In *The International Journal of Robotics Research*, 2008, vol. 27(11-12) (pp. 1308-1324).

[47] G. A. S. Pereira, M. F. M. Campos, and V. Kumar. Decentralized algorithms for multi-robot manipulation via caging. In *The International Journal of Robotics Research*, 2004, vol. 23(7-8), (pp. 783-795) (2004).

[48] Z. Wang, and V. Kumar. Object Closure and Manipulation by Multiple Cooperating Mobile Robots. In *IEEE International Conference on Robotics and Automation* , 2002, (pp. 394-399).

[49] A. Varava, D. Kragic, and F. T. Pokorny. Caging Grasps of Rigid and Partially Deformable 3-D Objects With Double Fork and Neck Features. In *IEEE Transactions in Robotics and Automation*, 2016, vol. 32(6), (1479–1497).

[50] J. A. Stork, F. T. Pokorny, and D. Kragic. Integrated Motion and Clasp Planning with Virtual Linking. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, (pp. 3007-3014).

[51] F. T. Pokorny, J. A. Stork, D. Kragic. Grasping objects with holes A topological approach. In *IEEE International Conference on Robotics and Automation* , 2013, (pp. 1100-1107).

[52] S. M. LaValle. Planning Alogrithms. Cambridge University Press, 2006.

[53] J. R. Munkres. Topology. Prentice Hall, 2000.

[54] K. Wise, and A. Bowyer A survey of global configuration-space mapping techniques for a single robot in a static environment. In *The International Journal of Robotics Research*, 2000, vol. 19(8), (pp. 762-779).

[55] T. Lozano-Perez. Spatial planning: A configuration space approach. IEEE Transactions on Computers 1983, vol. 2, (108-120).

[56] D. Zhu, and J. Latombe. New heuristic algorithms for efficient hierarchical path planning. In *IEEE Transactions in Robotics and Automation*, 1991, vol. 7(1), (pp. 9-20).

[57] E. Behar, and J. M. Lien. Mapping the configuration space of polygons using reduced convolution. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, (pp. 1242-1248).

[58] V. Milenkovic, E. Sacks, and S. Trac Robust complete path planning in the plane. In *Algorithmic Foundations of Robotics X* 2013, (pp. 37-52).

[59] J. Basch, L. J. Guibas, D. Hsu, and A. T. Nguyen. Disconnection proofs for motion planning. In *IEEE International Conference on Robotics and Automation* , 2001, (pp. 1765-1772).

[60] Z. McCarthy, T. Bretl, and S. Hutchinson. Proving path non-existence using sampling and alpha shapes. In *IEEE International Conference on Robotics and Automation* , 2012, (pp. 2563-2569).

[61] J. Mahler, F. T. Pokorny, Z. McCarthy, A. F. van der Stappen, and K. Goldberg. Energy-bounded caging: Formal definition and 2-D energy lower bound algorithm based on weighted alpha shapes. In *IEEE Robotics and Automation Letters* 2016, vol. 1(1), (pp. 508–515).

[62] L. Zhang, J. K. Young, and D. Manocha. Efficient cell labelling and path non-existence computation using C-obstacle query. In *The International Journal of Robotics Research*, 2008, vol. 27(11-12), (pp. 1246-1257).

[63] S. Salvador, and P. Chan. FastDTW: Toward Accurate Dynamic Time Warping in Linear Time and Space. In *Intelligent Data Analysis.*, 2007, vol. 11, (pp. 561-580).

[64] D. Kragic, L. Petersson, and H.I. Christensen. Visually guided manipulation tasks. In *Robotics and Autonomous Systems*, 2002, vol. 40(2) (pp. 193-203)

[65] V. Kyrki, D. Kragic, and H. I. Christensen. New shortest-path approaches to visual servoing. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2004 (pp. 349–354).

[66] D. Kragic, and H. I. Christensen. Survey on visual servoing for manipulation. Technical report, ISRN KTH/NA/P-02/01-SE, Computational Vision and Active Perception Laboratory, Royal Institute of Technology, Stockholm, Sweden . 2002.

[67] G. López-Nicolás, C. Sagüés, J. J. Guerrero, D. Kragic, and P. Jensfelt. Switching visual control based on epipoles for mobile robots. In *Robotics and Autonomous Systems*, 2008, vol. 56(7), (pp. 592-603).

[68] D. Aarno, and D. Kragic. Motion intention recognition in robot assisted applications. In *Robotics and Autonomous Systems*, 2008, vol. 56(8), (pp. 692-705).

[69] J. Bohg, A. Morales, T. Asfour, and D. Kragic. Data-Driven Grasp Synthesis–A Survey. In *IEEE Transactions on Robotics*, 2014, vol. 30(2), (pp. 289-309).

[70] A. Billard, and D. Kragic. Trends and challenges in robot manipulation. In *Science* 2019, vol. 364(6446).

[71] M. Vejdemo-Johansson, F. T. Pokorny, P. Skraba, and D. Kragic. Cohomological learning of periodic motion, In *Applicable Algebra in Engineering, Communication and Computing* 2015, vol. 26(1), (pp. 5-26)

[72] K. Hang, M. Li, J. A. Stork, Y. Bekiroglu, F. T. Pokorny, A. Billard, and D. Kragic. Hierarchical Fingertip Space: A Unified Framework for Grasp Planning and In-Hand Grasp Adaptation. In *IEEE Transactions on Robotics*, 2016, vol. 39, (pp. 960-972).

[73] J.-C. Latombe. *Robot Motion Planning*. Norwell, MA, USA: Kluwer Academic Publishers. 1991.

[74] M. Zucker, J. Kuffner, and M. Branicky. Multipartite RRTs for Rapid Replanning in Dynamic Environments. In *IEEE International Conference on Robotics and Automation*, 2007, (pp. 1603-1609).

[75] M. Phillips, and M. Likhachev. SIPP: Safe interval path planning for dynamic environments. In *IEEE International Conference on Robotics and Automation*, 2011, (pp. 5628-5635).

[76] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo. Robotics: modelling, planning and control. Springer Science & Business Media, 2010.

[77] Z. Zou, Z. Shi, Y. Guo, and J. Ye. Object detection in 20 years: A survey. In *arXiv preprint* 2019, arXiv:1905.05055.

Part II

Included Publications

Paper A

An Algorithm For Calculating Top-Dimensional Bounding Chains

J. Frederico Carvalho, Mikael Vejdemo-Johansson,
Danica Kragic and Florian T. Pokorny

Abstract

We describe the Coefficient-Flow algorithm for calculating the bounding chain of an $(n-1)$-boundary on an n-manifold-like simplicial complex S. We prove its correctness and show that it has a computational (time) complexity of $O(|S^{(n-1)}|)$ (where $S^{(n-1)}$ is the set of $(n-1)$-faces of S). We estimate the big-O coefficient which depends on the dimension of S and the implementation. We present an implementation, experimentally evaluate the complexity of our algorithm, and compare its performance with that of solving the underlying linear system.

1 Introduction

Topological spaces are by and large characterized by the cycles in them (i.e., closed paths and their higher dimensional analogues) and the ways in which they can or cannot be deformed into each other. This idea had been recognized by Poincaré from the beginning of the study of topology. Consequently much of the study of topological spaces has been dedicated to understanding cycles, and these are the key features studied by the topological data analysis community [12].

One key part of topological data analysis methods is to distinguish between different cycles, more precisely, to characterize different cycles according to their homology class. This can be done efficiently using cohomology [13, 17]. However, such methods only distinguish between non-homologous cycles, and do not quantify the difference between cycles. A possible way to quantify this difference is to solve the problem of finding the chain whose boundary is the union of the two cycles in

question, as was proposed in [5] by solving the underlying linear system, where the authors also take the question of optimality into account (with regards to the size of the resulting chain). This line of research ellaborates on [10] where the authors considered the problem of finding an optimal chain which is in the same homology class as a given chain. More recently, in [18] the authors have taken a similar approach to ours using a combinatorial method to compute bounding chains of 1-cycles on 3-dimensional simplicial complexes.

In this paper, we explore the geometric properties of simplicial n-manifolds to provide an algorithm that is able to calculate a chain whose boundary is some prescribed $(n-1)$-dimensional cycle, and we show that the proposed algorithm has a complexity which is linear in the number of $(n-1)$-faces of the complex. This is substantially faster than calculating a solution to the linear system as considered in [5], for which the complexity would be, at least, quadratic on the number of n-dimensional faces [15].

1.1 Background

In what follows we make extensive use of sequences. Therefore, for any $n \in \mathbb{N}$, we abbreviate $x_0, \ldots, x_n$ to $x_{0:n}$.

1.1.1 Simplicial complexes

Given a set of points $P \subseteq \mathbb{R}^n$, we define a *k-dimensional simplex*, or k-simplex, on points of P as the ordered set $[p_{0:k}]$, where $p_{0:k} \in P$ are $k+1$ affinely independent points and are called the *vertices* of $[p_{0:k}]$. We represent the simplex $[p_{0:k}]$ by the convex hull of the points $p_{0:k}$, and we say that two simplices are the same if they have the same points and the ordering of the points differs only by an even permutation. If the ordering differs by an odd permutation we say they have opposite *orientations*.

Since a convex hull of a finite set of points is a bounded closed set, it carries the notion of a *boundary* $\partial[p_{0:k}]$ which is defined as:

$$\partial[p_{0:k}] = [p_{1:k}] + (\sum_{i=1}^{k-1} (-1)^i [p_{0:i-1}, p_{i+1:k}]) + (-1)^k [p_{0:k-1}].$$

The above sum can be interpreted as a "union with orientation", and multiplying by 1 or -1 is the identity or a reversal of orientation, respectively. Note that if $p_{0:k}$ are affinely independent, then the boundary of the convex hull does indeed correspond to the union of the convex hulls of all subsets of $\{p_{0:k}\}$ with k distinct points.
For example, the boundary of the simplex $[p_0, p_1, p_2]$ is given by:

$$[p_0, p_1] - [p_0, p_2] + [p_1, p_2]$$

Applying the only possible orientation-reversing permutation to the simplex $[p_0, p_2]$ gives $[p_0, p_1] + [p_1, p_2] + [p_2, p_0]$. This corresponds to the union of the edges that

form the boundary of the triangle $[p_0, p_1, p_2]$ oriented in such a way as to form a *closed path.*

Definition 2. *A set of points $P \subseteq \mathbb{R}^n$ and a set of simplices $T = \{\sigma_{0:N}\}$ defines a* geometric simplicial complex *$S = (P, T)$ if any finite subset of a simplex in T is also in T, and given any two simplices $[p_{0:k}], [q_{0:k'}] \in T$, the intersection of the convex hulls $[p_{0:k}] \cap [q_{0:k'}]$ is the convex hull of $\{p_{0:k}\} \cap \{q_{0:k'}\}$ and is also a simplex in T.*

*For any d we define the d-*skeleton *of T by $T^d = \{\sigma \in T \mid \dim \sigma \leqslant d\}$ and the d-th* level *as $T^{(d)} = \{\sigma \in T \mid \dim \sigma = d\}$.*

Given two simplices σ, τ we write $\tau \lhd \sigma$ if $\tau \subset \sigma$ and $\dim \tau = \dim \sigma - 1$ which can be read as "τ is a top-dimensional face of σ". Note that $\lhd$ is not transitive, and therefore it is not a preorder. Its transitive closure, $\tau < \sigma$ however, defines a preorder. We can thus read $\tau < \sigma$ as "τ is contained in the boundary of σ" or simply "τ is a *face* of σ".

We say that a simplicial complex $S = (P, T)$ has *dimension* d if d is the largest integer such that $T^{(d)} \neq \emptyset$.

Definition 3. *A d-dimensional simplicial complex $S = (P, T)$ is called a* d-manifold-like *simplicial complex if*

- *for every $\tau \in T^{d-1}$ there exists some $\sigma \in T^{(d)}$ such that $\sigma > \tau$, and*
- *if $\dim \tau = (d-1)$ then there are at most two $\sigma \in T^{(d)}$ satisfying $\sigma > \tau$.*

Note that a triangulation of a d-manifold is a manifold-like simplicial complex, however the definition also includes other spaces like triangulations of manifolds with boundary and the pinched torus.

1.1.2 Algebraic description

We will focus on finite geometric simplicial complexes $S = (P, T)$ (where $|P|, |T| < \infty$). Since such an S has a finite number of simplices, we can define for each level $0 \leqslant k \leqslant \dim(S)$ an injective function $\iota_k : T^{(k)} \to \mathbb{N}$ such that $\iota_k(T^{(k)}) = \{1, \ldots, |T^{(k)}|\}$; we call ι an *enumeration* of $T^{(k)}$. From this we define the *chain complex* associated with S.

Definition 4. *Given a simplicial complex $S = (P, T)$ the* chain complex *associated with S is defined as the pair $\{(C_k(S), d_k)\}_{k=0}^{+\infty}$ where the $C_k(S)$ are vector spaces defined as $C_k(S) = \mathbb{R}^{|T^{(k)}|}$ and the d_k are linear maps $d_k : C_k(S) \to C_{k-1}(S)$ defined on basis elements as*

$$d_k(e_i) = \sum_{\tau \in \partial(\iota_k^{-1}(i))} o(i, \tau) e_{\iota_{k-1}(\tau)}$$

where $o(i, \tau)$ is the orientation of τ induced by the boundary of $\sigma = \iota_k^{-1}(i)$.

It can be shown that $d_k \circ d_{k+1} = 0$, which allows us to define, for each k, the k-th homology group of S as

$$H_k(S) = \ker(d_k)/\mathrm{im}(d_{k+1}).$$

By a slight abuse of notation, for a simplicial complex $S = (P, T)$ and a k-chain c, we write c_σ for the coefficient corresponding to σ, e_σ for the corresponding basis element and d for the appropriate boundary map whenever these are clear from their contexts.

We call the elements $p \in C_k(S)$ such that $dp = 0$, *k-cycles*. Two k-cycles p, p are said to be *homologous* if there exists a chain $c \in C_{k+1}(S)$ such that $dc = p - p'$, so that $p - p'$ is called a *k-boundary*. This defines k-homology as the group of k-cycles quotiented by the homology relation.

1.2 Problem description and contribution

We are interested in the bounding chain problem, that is, given a cycle p, we want to decide whether or not p is a boundary, and in case it is, provide a witness in the form of a chain c such that $\partial c = p$; we call c a *bounding chain* of p. To achieve this, we further specialize the problem to

$$\begin{aligned} \text{solve: } & \partial c = p \\ \text{subject to: } & c_\sigma = v, \end{aligned} \tag{A.1}$$

where p is a specified $(n-1)$-boundary in an n-manifold-like simplicial complex S, σ is an n-simplex, and v is a pre-specified real number. In general, the equation $\partial c = p$ has more than one solution c, therefore by adding the constraint $c_\sigma = v$ we are able to make this solution unique.

The Coefficient-Flow algorithm that we present solves this restricted form of the bounding chain problem (by providing one such bounding chain if it exists) and has computational time complexity of $O(|S^{(n-1)}|)$. Furthermore, we show how the parameters σ and v can be done away with in cases where the chain is unique, and we discuss how this algorithm can be used to find a minimal bounding chain.

1.3 Related Work

In [6] the authors address the problem of computing the area of a homotopy between two paths on 2-dimensional manifolds, which can be seen as a generalization of the same problem, for 2-dimensional meshes via the Hurewicz map [16]. In [5] the authors provide a method for calculating the minimum area bounding chain of a 1-cycle on a 2d mesh, that is the solution to the problem

$$\arg\min_c \mathrm{area}(c) = p, \qquad \text{where } \partial c = p \tag{A.2}$$

and p is a 1-chain on a given simplicial complex. This is done by using optimization methods for solving the associated linear system. These methods however have time

complexity lower-bounded by matrix multiplication time which is in $\Omega(\min(n,m)^2)$ where n, m are the number of rows and columns of the boundary matrix [1] [9]. This complexity quickly becomes prohibitive when we handle large complexes, such as one might find when dealing with meshes constructed from large pointclouds.

More recently, in [18] the authors proposed a method for computing bounding chains of 1-cycles in 3-dimensional complexes, using a spanning tree of the dual graph of the complex.

In [10] the authors address the related problem of efficiently computing an optimal cycle p' which is homologous to a given cycle p (with $\mathbb{Z}$ coefficients). This is a significant result given that in [7] the authors proved that this cannot be done efficiently (i.e., in polynomial time) for 1-cycles using $\mathbb{Z}_2$ coefficients, a result that was extended in [8] to cycles of any dimension.

2 Methodology

For any simplicial complex S, and any pair of simplices $\sigma, \tau \in S$ such that, $\tau \triangleleft \sigma$, we define the index of τ with respect to σ as $\langle \tau, \partial\sigma \rangle = \langle e_\tau, de_\sigma \rangle$ [14]. Note that the index corresponds to the orientation induced by the boundary of σ on τ and can be computed in $O(d)$ time by the following algorithm:

Algorithm A2: INDEX(σ, τ)

input : σ - k-simplex represented as a sorted list of indices of points.
τ - $(k-1)$-face of σ represented as a sorted list of indices.
output: The orientation of τ induced by σ

```
for i ← 0 ... dim τ do
    if τ_i ≠ σ_i then
        orientation ← (−1)^i
        break loop
return orientation
```

By inspecting the main loop we can see that INDEX(σ, τ) returns $(-1)^i$ where i is the index of the first element at which σ and τ differ. We assume τ is a top dimensional face of σ, so if $\sigma = [s_{0:d}]$, then by definition $\tau = [s_{0:(i-1)}, s_{(i+1):d}]$ for some i, and so the coefficient of τ in the boundary of σ is $(-1)^i$ as per the definition of the boundary operator. This is also the index of the first element at which τ and σ differ, since they are represented as sorted lists of indices.

The following is an intuitive result, that will be the basis for the COEFFICIENT-FLOW algorithm that we will present in the sequel.

[1] Which corresponds to the number of $(k-1)$- and k-faces of the complex, respectively.

Proposition 1. *Let S be a manifold-like simplicial complex, let c be an n-chain on S and p its boundary. Then for any pair of n-simplices $\sigma \neq \sigma'$ with $\partial\sigma \cap \partial\sigma' = \{\tau\}$ we have:*

$$c_\sigma = \langle \tau, \partial\sigma \rangle (p_\tau - \langle \tau, \partial\sigma' \rangle c_{\sigma'}).$$

Proof. If we expand the equation $\partial c = p$, we get $p_\tau = \sum_{\tau \lhd \omega} \langle e_\tau, d(c_\omega e_\omega) \rangle$, recall that by definition $d(c_\omega e_\omega) = \sum_{\nu \lhd \omega} \langle \nu, \partial\omega \rangle c_\omega e_\nu$; and so we get $p_\tau = \sum_{\tau \lhd \omega} \langle \tau, \partial\omega \rangle c_\omega e_\tau$.

Now since S is a manifold-like simplicial complex and $\tau = \partial\sigma \cap \partial\sigma'$, then σ, σ' are the *only* cofaces of τ, and hence we have:

$$p_\tau = \langle \tau, \partial\sigma \rangle c_\sigma + \langle \tau, \partial\sigma' \rangle c_{\sigma'}$$

which can be reorganized to $c_\sigma = \frac{p_\tau - \langle \tau, \partial\sigma' \rangle c_{\sigma'}}{\langle \tau, \partial\sigma \rangle}$. Finally, since the index $\langle \tau, \partial\sigma \rangle$ is either 1 or -1, we can rewrite this equation as:

$$c_\sigma = \langle \tau, \partial\sigma \rangle (p_\tau - \langle \tau, \partial\sigma' \rangle c_{\sigma'})$$

□

Next, we present an algorithm to calculate a bounding chain for a $(n-1)$-cycle in an n-manifold-like simplicial complex. The algorithm proceeds by checking every top-dimensional face σ, and calculating the value of the chain on adjacent top-dimensional faces, using Proposition 1.

In order to prove that the COEFFICIENT-FLOW algorithm solves problem (A.1), will use the fact that we can see the algorithm as a traversal of the *dual graph.*

Definition 5. *Given an n-dimensional simplicial complex S, recall that the* dual graph *is a graph $G(S) = (V, E)$ with set of vertices $V = S^{(n)}$ and $(\sigma, \sigma') \in E$ if $\dim(\sigma \cap \sigma') = n - 1$.*

Proposition 2. *If S is a manifold-like simplicial complex, where $G(S)$ is connected, and p is an $(n-1)$-boundary, then* COEFFICIENT-FLOW(p, σ, v) *returns a bounding chain c of p satisfying $c_\sigma = v$, if such a boundary exists. Furthermore, the main loop (*05-21*) is executed at most $O(|S^{(n-1)}|)$ times.*

Proof. We start by proving the bound on the number of executions of the main loop. This is guaranteed by the fact that the loop iterates while the queue is non-empty, and a triple (σ, τ, v) can only be inserted in line 24 if τ has not been marked as seen. Furthermore, since τ has at most two cofaces, say, σ, σ', we can only enqueue τ if we are analyzing σ or σ' and so for each τ, at most two elements are placed in the queue, and hence the main loop only gets executed at most $2|S^{(n-1)}|$ times.

To prove correctness of the algorithm, we have to prove that it outputs an error if and only if the problem has no solution, and otherwise it outputs a bounding chain of p with $c_{\sigma_0} = v$.

First, we note that if a face σ is marked as seen, the value of c_σ can never be reassigned. This is because the program branches on whether or not σ has been

Algorithm A3: COEFFICIENT-FLOW(p, σ_0, v_0)

input : p — an $n-1$ boundary of the simplicial complex S
σ_0 — an n–simplex from where the calculation will start
v_0 — the value to assign the bounding chain at sigma
output: c — a bounding chain of p satisfying $c_{\sigma_0} = v_0$ (if it exists)

```
1  initialize c and mark every n- and (n − 1)-cell of S as not seen.
2  initialize an empty queue Q
3  let τ_0 ∈ ∂σ_0
4  enqueue (σ_0, τ_0, v_0) into Q.
5  while Q is non-empty do
6      (σ, τ, v) ← pop first element from Q
7      if σ has been marked seen then
8          if v ≠ c_σ then
9              the problem has no solution
10     else
11         if τ has been marked seen then
12             skip
13         c_σ ← v
14         mark τ and σ as seen
15         for τ' ∈ ∂σ do
16             if σ is the only coface of τ' then
17                 mark τ' as seen
18                 if p_τ' ≠ ⟨∂σ, τ'⟩v then
19                     the problem has no solution
20             else
21                 if τ' has not been marked as seen then
22                     σ' ← other coface of τ'
23                     v' ← ⟨∂σ', τ⟩(p_τ − ⟨∂σ, τ⟩v)
24                     enqueue (σ', τ', v') into Q
25 return c
```

marked as seen, and c_σ can only be assigned on line 13 which is bypassed if σ has been previously marked as seen. From this fact we conclude that $c_{\sigma_0} = v_0$ as it is assigned on line 13 and σ_0 marked as seen in the first iteration of the main loop.

Second, note that there is an edge between two n-faces in the dual graph if and only if they share an $(n-1)$-face. This implies that as we execute the algorithm, analyze a new n-face σ and successfully add the other cofaces of elements of the boundary of σ, we add the vertices neighboring σ in the dual graph. Since the dual graph is connected all of the nodes in the graph are eventually added, and hence all

of the n-faces are analyzed.

Third, we note that for any pair (τ, σ) with $\dim \sigma = n$ and $\tau \triangleleft \sigma$, either σ is the only coface of τ, or τ has another coface, σ'. In the first case, if $p_\tau \neq c_\sigma \langle \tau, \partial\sigma \rangle$ an error is detected on line 19. In the second case, assuming that the triple (σ, τ, v) is enqueued before (σ', τ, v') we have $v' = \langle \partial\sigma', \tau \rangle (p_\tau - \langle \partial, \sigma \rangle v)$ as is assigned in line 23 then

$$\begin{aligned}(dc)_\tau &= \langle \partial\sigma, \tau \rangle v + \langle \partial\sigma', \tau \rangle v' \\ &= \langle \partial\sigma, \tau \rangle v + \langle \partial\sigma', \tau \rangle (\langle \partial\sigma', \tau \rangle (p_\tau - \langle \partial, \sigma \rangle v)) \\ &= \langle \partial\sigma, \tau \rangle v + p_\tau - \langle \partial\sigma, \tau \rangle v = p_\tau.\end{aligned}$$

Finally, since upon the successful return of the algorithm, this equation must be satisfied by every pair $\tau \triangleleft \sigma$, it must be the case that $dc = p$. If this is not the case, then there will be an error in line 9 and the algorithm will abort. □

Note that the connectivity condition can be removed if we instead require a value for one cell in each connected component of the graph $G(S)$ and, and throw an error in case there is an $(n-1)$-simplex τ with no cofaces, such that $p_\tau \neq 0$. Furthermore, the algorithm can be easily parallelized using a thread pool that iteratively processes elements from the queue.

Finally, in the case where it is known that S has an $(n-1)$-face τ with a single coface, we do not need to specify σ or v in Coefficient-Flow, and instead use the fact that we know the relationship between the coefficient p_τ and that of its coface in a bounding chain c of p, i.e., $p_\tau = \langle \partial\sigma, \tau \rangle c_\sigma$. This proves Corollary 2.

Algorithm A4: Bounding-Chain(p)

```
   input : p — an n − 1 boundary of the simplicial complex S
   output: c — a bounding chain of p
 1 for τ ∈ S^{n−1} do
 2     if τ has a single coface then
 3         σ ← the coface of τ
 4         v ← ⟨∂σ, τ⟩ p_τ
 5         break loop
 6 c ←Coefficient-Flow(p, σ, v)
 7 returnc
```

Corollary 2. *If S is a connected n-manifold-like simplicial complex with a connected dual graph, and has an $(n-1)$-face with a single coface, then given an $(n-1)$-cycle p on S,* Bounding-Chain(p) *returns a bounding chain of p if one exists.*

2.1 Implementation details

We will now discuss some of the choices made in our implementation of our algorithm[2]. Before we can even tackle the problem of considering chains on a simplicial complex we first need to have a model of a simplicial complex. For this, we decided to use a simplex tree model [4] provided by the GUDHI library [19] as it provides a compact representation of a simplicial complex (the number of nodes in the tree is in bijection with the number of simplices) which allows us to quickly get the enumeration of a given simplex σ. Indeed, the complexity of calculating $\iota_k(\sigma)$ is in $O(\dim \sigma)$.

It is important to compute the enumeration quickly because in our implementation of Coefficient-Flow we use arrays of booleans to keep track of which faces of the simplicial complex have been seen before, as well as numerical arrays to store the coefficients of the cycle and its bounding chain, which need to be consulted at every iteration of the loop.

However, finding the cofaces of the simplicial complex is not as easy in a simplex tree, since, if $\sigma = [p_{i_0:i_k}]$, this would require to search every child of the root node of the tree that has an index smaller than i_0, followed by every child of the node associated with p_{i_0} and so on, which in the worst case scenario is in $O(\dim \sigma |S^{(0)}|)$. Thus we need to adopt a different method. For this, we keep a Hasse diagram of the face relation which comprises a directed graph whose vertices are nodes in the simplex tree and has edges from each face to its codimension-1 cofaces (see for instance [11] for more details of this data-structure). This allows us to find the codimension-1 cofaces of a simplex of S in $O(1)$ with a space overhead in $O(|S|)$.

With these elements in place, we can perform a full analysis of the complexity of the Coefficient-Flow:

Lemma 2. *The* Coefficient-Flow *algorithm using a simplex tree and a Hasse diagram has computational (time) complexity* $O(d^2|S^{(d-1)}|)$ *where* $d = \dim S$.

Proof. In Proposition 2 we saw that the Coefficient-Flow algorithm executes the main loop at most $O(|S^{(d)}|)$ times, so we only need to measure the complexity of the main loop, in order to obtain its time complexity. This can be done by checking which are the costly steps:

- In lines 7 and 11 checking whether a face has been marked as seen requires first computing $\iota_k(\tau)$ which, as we stated above, has time complexity $O(k)$, with $k \leq d$.
- In line 15, computing the faces of a simplex σ requires $O(\dim \sigma^2)$ steps, and yields a list of size $\dim \sigma$, hence the inner loop in lines (15–24) is executed $\dim \sigma$ times, where $\dim \sigma = d$, for each element placed in the queue.
- The loop (15–24) requires once again, computing $\iota(\tau')$, and $\langle \partial\sigma, \tau \rangle$, each of these operations, as we explained before, carries a time complexity $O(d)$.

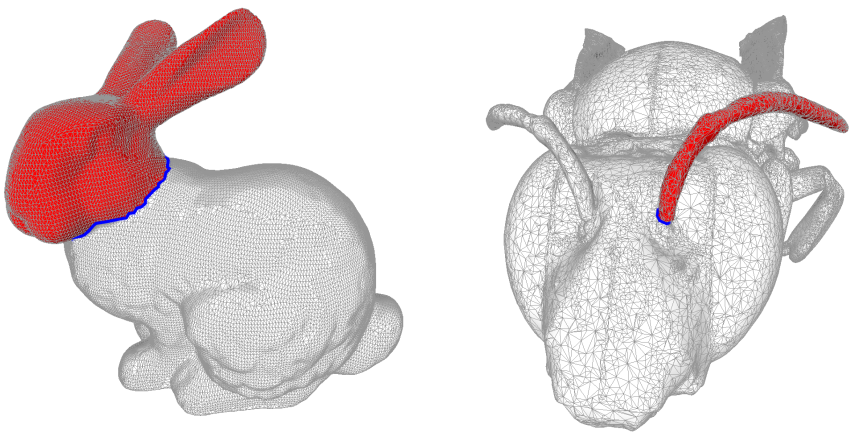
[2] available in `https://www.github.com/crvs/coeff-flow`

- All other operations have complexity $O(1)$.

Composing these elements, the total complexity of one iteration of the main loop is $O(\max\{d, d^2, d \cdot d\}) = O(d^2)$, which yields a final complexity for the proposed implementation, of $O(d^2|S^{(d-1)}|)$. □

2.2 Example runs and tests

In Figure A.1 we provide an example of the output of the COEFFICIENT-FLOW algorithm for the mesh of the Stanford bunny [3] and the *eulaema meriana* bee model from the Smithsonian 3D model library [2],



(a) (b)

Figure A.1: Examples of bounding chains: The edges in blue form cycles in the mesh, and the faces in red form the corresponding bounding chain as computed by the COEFFICIENT-FLOW algorithm. The presented bounding chains correspond to the optimal bounding chains (w.r.t. the number of faces), these can by obtained by choosing σ_0 and v_0 so as to yield the desired chain. In this case, since the two complexes are topological spheres and the cycles are simple cycles (meaning they are connected and do not self-intersect), there are only two possible bounding chains that do not include all the faces of the complex, which can be obtained by running the algorithm twice choosing σ_0 arbitrarily, and setting v_0 to be $0, n$ or $-n$ where $n = \max_{\tau \in S^{(1)}} |p_\tau|$. In the case of non-simple cycles, more alternatives would exist.

For comparison, we performed the same experiments using the Eigen linear algebra library [1] to solve the underlying linear system[3], and summarized the results in Table A.1. This allowed us to see that even though both approaches

[3]We use the least squares conjugate gradient descent method to solve the system.

remain feasible with relatively large meshes, solving the linear system consistently underperforms using the COEFFICIENT-FLOW algorithm.

mesh	faces	edges	vertices	Eigen	COEFFICIENT-FLOW
Bunny	69 663	104 496	34 834	2.073 (s)	**0.48911 (s)**
Bee (Sample)	499 932	749 898	249 926	116.553 (s)	**3.04668 (s)**
Bee (Full)	999 864	1 499 796	499 892	595.023 (s)	**7.15014 (s)**

Table A.1: Timing for computation of bounding chains using COEFFICIENT-FLOW, and using Eigen in several meshes. "Bee (Sample)/Bee (Full)" and "Bunny" refer to the meshes in Figure A.1 left and right, respectively. The mesh "Bee (Full)" is the one obtained from [2], whereas the one labeled "Bee (Sample)" is a sub-sampled version of it.

Even though COEFFICIENT-FLOW is expected to outperform a linear system solver (an exact solution to a linear system has $\Omega(n^2)$ time complexity), we wanted to test it against an approximate sparse system solver. Such solvers, (e.g., conjugate gradient descent [15]) rely on iterative matrix products, which in the case of boundary matrices of dimension d can be performed in $O((d+1)n)$ where n is the number of d-dimensional simplices, placing the complexity of the method in $\Omega((d+1)n)$. In order to observe the difference in complexity class we performed randomized tests on both implementations. In this scenario we made a mesh on 1-dimensional square from a random sample. By varying the number of points sampled from the square, we effectively varied the resolution of the mesh. Finally, at each resolution level we snapped a cycle onto the mesh, and computed its bounding chain using both COEFFICIENT-FLOW and by solving the sparse linear system as before. We plotted the timings in Figure A.2 from where we can experimentally observe the difference in the complexity class between our algorithm and the solution to the sparse linear system.

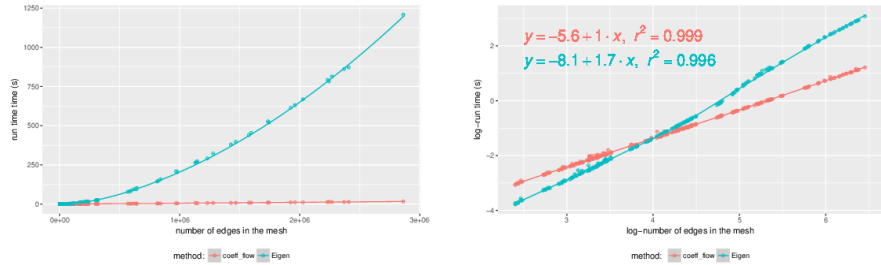
Furthermore by analyzing the Log-Log plot in Figure A.2 (right), we can confirm our complexity estimates by analyzing the slope of the lines where the samples are distributed, i.e., solving the sparse linear system is approximately $O(n^{1.7})$ complexity[4], whereas COEFFICIENT-FLOW displays linear complexity.

3 Conclusion and future work

While the problem of finding a bounding chain for a given cycle in a simplicial complex remains a challenging one for large complexes, we showed that this problem can be solved efficiently for codimension-1 boundaries. We implemented and tested our algorithm and have provided complexity bounds for its run-time.

However, this leaves open the question of finding bounding chains for boundaries of higher codimension, for which solving a large sparse linear system is still, to the

[4]Since boundary matrices are naturally sparse, and we are computing an approximate solution, the complexity can be improved beyond the aforementioned $\Omega(n^2)$



(a) (b)

Figure A.2: Running times for a calculating the bounding chain of a cycle as a function of the number of edges (a), and Log-log plot of the same data (b).

best of our knowledge, the only feasible approach, save for codimension 2 cycles in dimension 3 [18]. In the future we would like to generalize our algorithm to be able to work with cobounding cochains (i.e., in cohomology), as well as considering the optimality question (i.e., finding the minimal bounding chain w.r.t. some cost function).

Acknowledgements: This work has been supported by the Knut and Alice Wallenberg Foundation and the Swedish Research Council.

References

[1] Eigen. [Online] `http://eigen.tuxfamily.org/`

[2] Smithsonian X 3D: Eulaema meriana bee, Smithsonian gardens. [Online] `https://3d.si.edu`

[3] The Stanford 3D scanning repository. [Online] `https://graphics.stanford.edu/data/3Dscanrep/`

[4] Jean-Daniel Boissonnat and Clément Maria. The simplex tree: An efficient data structure for general simplicial complexes. *Algorithmica*, 70(3):406–427, Nov 2014.

[5] E. W. Chambers and M. Vejdemo-Johansson. Computing minimum area homologies. In *Computer Graphics Forum*, volume 34. Wiley Online Library, 2015.

[6] E. W. Chambers and Y. Wang. Measuring Similarity Between Curves on 2-manifolds via Homotopy Area. In *2013 Symposium on Computer Graphics*, SoCG '13, pages 425–434. ACM, may 2013.

[7] Erin W. Chambers, Jeff Erickson, and Amir Nayyeri. Minimum cuts and shortest homologous cycles. In *Symposium on Computational geometry*, 2009.

[8] Chao Chen and Daniel Freedman. Hardness Results for Homology Localization. *Discrete and Computational Geometry*, 45(3), 2011.

[9] A. M. Davie and A. J. Stothers. Improved bound for complexity of matrix multiplication. *Proceedings of the Royal Society of Edinburgh Section A: Mathematics*, (2), 2013.

[10] Tamal K. Dey, Anil N. Hirani, and Bala Krishnamoorthy. Optimal Homologous Cycles, Total Unimodularity, and Linear Programming. *SIAM Journal on Computing*, 40(4):1026–1044, jan 2011.

[11] Giuseppe Di Battista and Roberto Tamassia. Algorithms for plane representations of acyclic digraphs. *Theoretical Computer Science*, 61(2), 1988.

[12] H. Edelsbrunner and J. Harer. *Computational topology: an introduction*. American Mathematical Soc., 2010.

[13] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological Persistence and Simplification. *Discrete & Computational Geometry*, 28(4), November 2002.

[14] R. Forman. Morse Theory for Cell Complexes. *Advances in Mathematics*, 134(1), March 1998.

[15] Gene H. Gloub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, 3 edition, 1996.

[16] A. Hatcher. *Algebraic topology*. Cambridge University Press, 2 edition, 2002.

[17] Florian T. Pokorny, Majd Hawasly, and Subramanian Ramamoorthy. Topological trajectory classification with filtrations of simplicial complexes and persistent homology. *International Journal of Robotics Research (IJRR)*, 2016.

[18] Ana Alonso Rodríguez, Enrico Bertolazzi, Riccardo Ghiloni, and Ruben Specogna. Efficient Construction Of 2-Chains With a Prescribed Boundary. *Journal of Numerical Analysis*, 55(3):1159–1187, 2017.

[19] The GUDHI Project. *GUDHI User and Reference Manual*. GUDHI Editorial Board, 2015. [Online] `http://gudhi.gforge.inria.fr/doc/latest/`

Paper B

Path Clustering with Homology Area

J. Frederico Carvalho, Mikael Vejdemo-Johansson,
Danica Kragic and Florian T. Pokorny

Abstract

Path clustering has found many applications in recent years. Common approaches to this problem use aggregates of the distances between points to provide a measure of dissimilarity between paths which do not satisfy the triangle inequality. Furthermore, they do not take into account the topology of the space where the paths are embedded. To tackle this, we extend previous work in path clustering with relative homology, by employing minimum homology area as a measure of distance between homologous paths in a triangulated mesh. Further, we show that the resulting distance satisfies the triangle inequality, and how we can exploit the properties of homology to reduce the amount of pairwise distance calculations necessary to cluster a set of paths. We further compare the output of our algorithm with that of DTW on a toy dataset of paths, as well as on a dataset of real-world paths.

1 Introduction

Due to the prevalence of devices with access to positions sensors such as GPS, large datasets of paths have become increasingly available in the past few years [15]. Through path clustering, this data can be partitioned into sets of similar paths, from which motion primitives can be learned in order to control a robot without explicitly programming it [4, 17]. In this paper, we present a new method of computing a topologically inspired similarity measure for paths which can be applied to path clustering.

Clustering is a machine learning technique used to classify large sets of data into smaller subsets that are similar to each other [5]. However, the domain of "popular" clustering techniques, such as k–means and support vector machines, is that of data

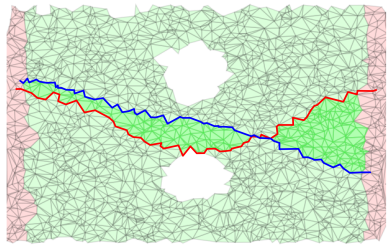


Figure B.1: Triangulated region of the plane X (green and red) with two holes (white). The region in red corresponds to a subset $A \subseteq X$ that are used as "goal regions" which contains the end points of the paths being analyzed. We also depict two edge-paths p, q (in red and blue) together with the minimum area bounding chain c in bright-green, which intuitively corresponds to the area that is delimited by the paths p and q. The area of the region c is interpreted as the distance between paths p and q.

points in a vector space of fixed dimension. Because paths have variable length, they cannot be easily represented in finite dimensions. Therefore, commonly used path clustering methods rely on calculating distances between paths and applying clustering methods that use distance alone [14].

These methods have been successfully applied to different problems, such as learning dynamic scene models, and car trajectories [3, 6, 11] to name a few. Comparative surveys of commonly used path distances can be consulted in [10, 13]. The aforementioned distances do not take into account the topology of the space within which the paths lie, and therefore cannot properly gauge the influence of obstacles on the clustering produced [18]. To do this [18] proposed a method for path clustering that only takes the obstacles into account.

In this paper, we propose an extension of the framework of clustering with relative homology proposed in [18], by using homology area [7]. Intuitively, this corresponds to defining the distance between two paths as the area enclosed between them (see an example in Figure B.1). This method allows us to obtain a more fine-grained control of how the clustering takes place as compared to only using relative homology. It also ensures that the obtained distance function is compatible with the results from homology clustering.

Further, we show how this path distance, even though computationally expensive to compute, can be leveraged when calculating a distance array to cluster a large set of paths, outperforming DTW[1] by an order of magnitude.

More details of our method can be consulted in `http://csc.kth.se/~jfpbdc/projects/homology_clustering`.

[1] Dynamic Time Warping (DTW) is a popular path distance which can be efficiently computed [19].

Main contributions

The main contributions of this paper is 4-fold (1) we define a distance for paths on a mesh using homology area and show that it satisfies the triangle inequality (Proposition 3). (2) Rather than using a reordering of the simplices in a simplicial complex to compute relative homology like in [18], we implement relative homology using simplicial complex quotient to speed-up relative homology area-computation (Algorithm B2). (3) We provide an algorithm that reduces the amount of homology area calculations on a dataset of N paths, from $O(N^2)$ to $O(N)$ (Algorithm B3, and Corollary 3). (4) We test our framework, both on a toy example, and on the dataset of paths on an indoor environment [8] and compare its performance to Dynamic Time Warping, a commonly used metric for path clustering.

1.1 Motivation and Related Work

The problem of learning by demonstration deals with extracting motion primitives for a robot from data collected from a human demonstration, rather than explicit programming [4]. However, if motion primitives are averaged from a diverse enough set of trajectories, the resulting motion may not be a good representative[2] of the demonstrated motions. Therefore, path clustering can be used to split the space of paths into cohesive sets of paths from which valid motion primitives can be extracted [17].

Path clustering has been extensively investigated, and and has found applications in surveillance, traffic analysis, among others [2, 3, 6, 11, 13]. In order to cope with the high dimensionality of the space of paths on a vector space, the employed methods use distance-based clustering rather than other methods that require a finite dimensional representation, such as k–means and support vector machines. An exhaustive review of path clustering combining different distances and clustering methods can be consulted in [1]. As mentioned before, the employed clustering methods rely only on distance between paths, to assign them to a cluster, therefore they need to compare each path to every other. This implies that the path distance needs to be evaluated $O(N^2)$ times where N is the number of paths. This fact puts a strain on path clustering methods, since path distances need to be evaluated quicly in order to cope with large databases of paths.

Recently [18] proposed a clustering method for paths based solely on their relative homology class. Intuitively, two paths belong to the same cluster (in which case they are said to be "homologous") if there are no obstacles in the area between them. The strength of this method, lies not only in the speed with which it can be evaluated, but also in the fact that it only needs to do a single pass through the data, outperforming distance-based methods. However, since this method only distinguishes paths based on topological features, the paths in each cluster present a wide range of variability. We therefore want to be able to cluster paths into

[2]If the distribution of paths is multimodal, then taking an average of the motions may yield behaviour inconsintent with the data.

more narrow categories in a manner that is compatible with the homology clusters. To this end, we use the results from [7] where an efficient method for computing the area of the region enclosed by two closed paths was presented. This notion of distance has the convenience of being compatible with the purely topological clustering in [18].

In this paper, we build upon the results in [7, 18] by extending the notion of minimum area homology to quotient complexes, and employing the area of the obtained region as a path distance. Further, we note that in clustering a set of N paths with k homology classes, the number of regions that need to be computed can be brought down from $O(N^2)$ to $O(kN)$ in the worst case. We show in the experiments section how this allows us to outperform DTW [19] when clustering even relatively small datasets (circa 2000 paths).

1.2 Mathematical Background

We now provide a brief description of the mathematical tools used in the remainder of the paper (an exhaustive treatment can be found in [9]). To do this, we begin by introducing Homology, which is a general algebraic tool that can be applied to understanding topological spaces, particularly through Simplicial Homology which we explain further on. Finally, we introduce relative homology and homology area which will use to define our path distance.

1.2.1 Homology

A chain complex is a pair $(C_\bullet, \partial_\bullet)$, where $C_\bullet = \{C_i\}_{i\in\mathbb{N}}$ is a sequence of vector spaces, and $\partial_\bullet = \{\partial_i : C_i \to C_{i-1}\}_{i\in\mathbb{N}}$ is a sequence of linear maps (matrices) called *boundary maps* (matrices) often depicted in a sequence as in (B.1).

$$\cdots \leftarrow C_i \xleftarrow{\partial_i} C_i \xleftarrow{\partial_{i+1}} C_{i+1} \leftarrow \cdots \tag{B.1}$$

Where the boundary matrices are required to satisfy $\partial_i\partial_{i+1} = 0$. We call an element of C_i an *i–chain*; and when no confusion arises, we denote all boundary maps as ∂, dropping the index.

Given a chain complex, we define two further sequences of vector spaces $Z_\bullet, B_\bullet$ by:

$$Z_i = \ker(\partial_i) \quad B_i = \operatorname{im}(\partial_{i+1})$$

We call elements of Z_i and B_i; *i–cycles* and *i–boundaries*, respectively. Further, since $\partial_i\partial_{i+1} = 0$ implies $B_i \subseteq Z_i$, we can define the quotient $H_i = Z_i/B_i$, which we call the i–th *homology* of the chain complex.

Given an i–boundary p and an $(i+1)$–chain c such that $p = \partial c$ we call c the *bounding chain* of p.

1.2.2 Simplicial Homology

Given $k+1$ affinely independent points in a vector space $\{x_0, \ldots, x_k\} \subseteq \mathbb{R}^n$ with $n > k$, we form the *k–simplex* $[x_{0:k}]$ (where $x_{0:k}$ is a shorthand for "$x_0, \ldots, x_k$"). Each such simplex has an orientation that is inherent to the ordering of its points or *vertices*. Two simplices with the same vertices are said to have opposite orientations if the ordering differs by an odd permutation.

Geometrically, we represent a simplex by the convex hull of its vertices, and its boundary by

$$\mathrm{bd}([x_{0:k}]) = \bigcup_{i=0}^{k} [x_{0:i-1}, x_{i+1:k}]$$

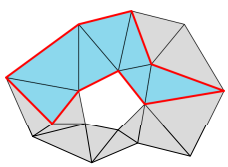
Given a topological space $X \subseteq \mathbb{R}^n$, we can obtain a discrete approximation of X by taking a discrete set $\tilde{X} \subseteq X$ and using a triangulation T of $\tilde{X}$ (in the remaining we let T be a subset of the Delaunay triangulation). From T, we can build a chain complex as follows:

In order to construct a chain complex out of $\tilde{X}$ and T, we denote the ordered[3] set of i–simplices in T by $T^{(i)} = \{\sigma_{1:b_i}\}$, and define $C_i = \mathbb{R}^{b_i}$. Furthermore, for each $j = 1, \ldots, b_i$ we set

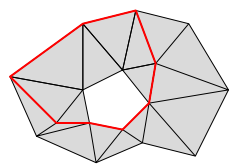
$$\partial_i e_j = \sum_{k=0}^{i} \pm e'_{l_{i,k}}$$

where $e_{1:b_i}$ and $e'_{1:b_{i-1}}$ form the canonical basis of C_i and C_{i-1}, respectively; and $l_{i,k}$ is the index of the k–th face of σ_j and the sign depends on the orientation.

Thus, we obtain a *simplicial complex* (see [9] for a proof), which we denote by $(C_\bullet(X), \partial_\bullet)$ and its homology by $H_\bullet(X)$.



(a) A cycle which is a boundary (in red) and its bounding chain (in blue).



(b) A cycle which is not a boundary (in red).

In this setting, it can be observed that 1–cycles correspond to sums of edges that form a closed path, and that 1–boundaries correspond to those 1–cycles that form the boundary of a closed region (see Figure B.2 for an example).

1.2.3 Relative Homology

Given a chain complex $(C_\bullet, \partial_\bullet)$ we define a *subcomplex* as a complex $(A_\bullet, \partial_\bullet)$, where for each i, A_i is a subspace of C_i and ∂_i is the boundary map of $(C_\bullet, \partial_\bullet)$ restricted

[3] This is an arbitrary ordering, as it is only needed for indexing purposes.

to A_i. From such a subcomplex we can define *relative chain complex* $(D_\bullet, \partial'_\bullet)$, as $D_i = C_i/A_i$ and ∂'_i as the map induced by ∂_i on the quotient.

In the case of simplicial complexes, as introduced in Section 1.2.2, we are interested in subcomplexes $A_\bullet$ such that the generators of each A_i are a subset of the generators of C_i. This ensures that $A_\bullet$ can be seen as a closed subspace $A \subseteq X$. In this case we denote the relative chain complex by $(C_\bullet(X, A), \partial_\bullet)$ and its homology by $H_\bullet(X, A)$.

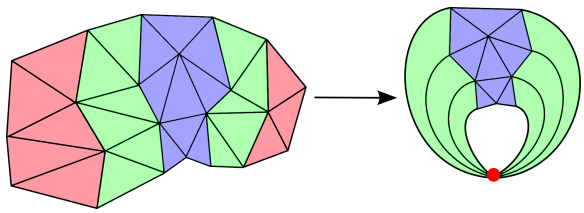


Figure B.3: Representation of the quotient map on a simplicial complex. The region in blue is unaffected whereas the region in red is collapsed onto a single point and the region in green is distorted. In this example, we have taken a space with no holes and have produced a space with one hole.

Now, taking the quotient $C_i(X)/C_i(A)$ corresponds to ignoring all topological features in A, and hence we expect to see a relationship between the relative complex and the quotient X/A (see Figure B.3) which is made explicit in the following Lemma:

Lemma 3 ([9]). *Let X be a simplicial complex and A be a subcomplex, then the relative homology $H_i(X, A)$ is isomorphic to $H_i(X/A)$*

1.2.4 Homology area

We say that a simplicial complex is k–dimensional if it has at least one simplex of dimension k and no simplices of dimension higher than k. Let S be a two-dimensional simplicial complex, and let $S^{(2)} = \{\sigma_{1:b_2}\}$, then for any 2–chain $c = (c_{1:b_2}) \in C_2(X)$ as

$$\text{area}(c) = \sum_i |c_i| \text{Area}(\sigma_i)$$

where $\text{Area}(\sigma)$ denote the area of the 2–dimensional simplex σ. Finally, we define the *minimum area bounding chain*[4] of a 1–cycle p as

$$c^* = \underset{c \in C_2(X)}{\arg\min}\ \text{area}(c) \qquad \text{subject to: } \partial c = p \tag{B.2}$$

[4] In [7] we call it a "minimum area homology".

2 Methodology

We start this section by presenting and describing the algorithms that make up the main contributions of this paper as well as provide the complexity of Algorithm B3. In the remainder of the section, we introduce the theory that ensures the correctness of our algorithms.

2.1 Algorithms

We now present the algorithm to compute the quotient of a simplicial complex (Algorithm B2), and to compute the homology-area of associated to a set of paths (Algorithm B3).

In Algorithm B2, we assume the simplicial complex S is given as a pair of sets $(points, tris)$ where $points \subseteq \mathbb{R}^2$ is a (finite) set containing the vertices of the simplicial complex, and $tris$ is a triangulation of those points, i.e. $tris = S^{(2)}$ (the complex is assumed to satisfy the fact that the 1–dimensional simplices are the boundary of 2–simplices). Similarly, the quotient Q is represented by a pair of points and triangles $(qPts, qTris)$.

The subset $A \subseteq S$ so that $Q = S/A$ is passed to Algorithm B2 as a characteristic function $q : \mathbb{R}^2 \to \{0, 1\}$ such that $q(x) = 1$ if and only if $x \in A$.

The algorithm proceeds by first letting $qPts$ have $Null$ as its first point, as this will be the image of every point in A and therefore has no (unique) correspondence with a point in S, and then adding every point that is not in A. Secondly in the lines 6–16 it translates the triangles in $tris$ to triangles with the corresponding vertices in $qPts$. This translation is done in lines 9–14, where the indices in a triangle tri are replaced in $qTri$ by their counterparts from $qPts$ unless the point is in A; subsequently if tri contains a point in A the index 0 is added to $qTri$ in line 16, so that it is added at most once. The resulting simplex $qTri$ is then added to $qTris$, and $Q = (qPts, qTris)$ is returned.

Now we proceed to show how we can use the quotient complex and bounding chains to efficiently calculate a distance array using homology area in Algorithm B3. As we note in Section 3 even though the process of calculating a bounding chain is computationally expensive, calculating the distance array can be done efficiently due to Corollary 3.

Algorithm B3 comprises two phases, phase one in lines 1–16 performs the heavy calculations (i.e. it computes the necessary bounding chains). Phase two, in lines 17–27 uses these bounding chains to compute the array of distances between paths.

This algorithm uses the subroutine $BoundingChain(p, q)$ which, given two 1–chains p, q, calculates the bounding chain of $p - q$ in case they are homologous, and reports a failure otherwise. In our implementation, we use a least squares algorithm to solve equation (B.2) (which we justify in Section 2.2), and intuitively, given the innate sparsity of the boundary matrix, an alternative would be to use a simple row-reduction algorithm, but the associated change of base matrix will be dense, making it more computationally expensive.

Algorithm B2: Computing a quotient where the support of a characteristic function is collapsed to a single point

```
   input  : S /* simplicial complex                                   */
   input  : q /* characteristic function                             */
   /* q satisfies q(x) = 1 for x ∈ A,                                 */
   /* and q(x) = 0 otherwise.                                         */
   /* S = (points, tris) where:                                       */
   /* pts:  list of 2D points.                                        */
   /* tris:  list of triangles, each                                  */
   /* is a triple of indices in pts.                                  */
   output: Q /* simplicial complex                                   */
   /* Q is a representative of S/A                                    */
 1 qPts = {Null};
   /* 1st is the image of A                                           */
 2 for p ∈ pts do
 3     if q(p) = 0 then
 4         append p to qPts;
 5 qTris = {};
 6 for tri ∈ tris do
 7     qTri = {};
 8     has0 = False;
 9     for v ∈ tri do
10         if q(pts[v]) = 1 then
11             has0 = True;
12         else
13             qv = index of v in qPts;
14             append qv to qTri;
15     if has0 then
           /* include 0 at most once                                  */
16         prepend 0 to qTri;
17 return Q = (qPts, qTris)
```

In phase one, the algorithm builds up three lists $PathCluster$, a list that at position i it has the homology class of path $Paths[i]$; $HomologyClusters$, a list of lists where the elements of $HomologyClusters[i]$ are the indices of paths in the i–th homology class. The third list, $BoundingChains$, stores, at position i the results of $BoundingChain(p, q)$ where p is the first path in the i–th homology cluster, and q ranges over the paths homologous to p, in order of appearance in $Paths$.

In phase two, the algorithm uses lists built up in phase one to produce the array of distances $Distarray$. To achieve this it checks what the homology cluster of path

Algorithm B3: Distance array using homology area.

```
   input : Paths /* a list of paths                                    */
   output: An array of pairwise distances
 1 PathCluster = {};
 2 BoundingChains = {};
 3 HomologyClusters = {};
 4 for i ∈ {0, ..., #Paths} do
 5     for Cluster ∈ HomologyClusters do
           /* check if the current path is in any existing cluster
              */
 6         p = Paths[i];
 7         q = Paths[Cluster[0]];
 8         j = Index(Cluster, HomologyClusters);
 9         Chain = BoundingChain(q, p);
10         if successful then
11             append j to PathCluster;
12             append i to Cluster;
13             append Chain to BoundingChains[j];
14     if no cluster was found then
           /* not homologous to any already seen path, new cluster
              */
15         append {i} to HomologyClusters;
16         append {} to BoundingChains;
   /* use the bounding chains to compute pairwise distances     */
17 Distarray = {};
18 for i ∈ {0, ..., #Paths} do
19     c = PathCluster[i];
20     for j ∈ {i + 1, ..., #Paths} do
21         if PathCluster[j] = c then
22             k = index of i in HomologyCluster[c];
23             l = index of j in HomologyCluster[c];
24             Chain = BoundingChains[c, k − 1] − BoundingChains[c, l − 1];
25             append area(Chain) to Distarray;
26         else
27             append ∞ to Distarray;
28 return Distarray
```

i is, (line 19) and computes the distance to path j for every $j > i$ by checking if they have the same homology class; if they are, it uses Corollary 3 (which is summarized

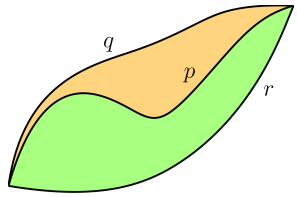


Figure B.4: Example of how given a bounding chain for $p - q$ (orange region) and one for $p - r$ (green region) we obtain a bounding chain for $q - r$ (union of orange and green regions). The fact that the bounding chains have signed area, ensures that when two regions intersect, the area cancels out.

in Figure B.4) to calculate the distance (lines 21–25), otherwise, it stores the value ∞ (line 27).

Now, note that finding the minimum area bounding chain amounts to solving a $k \times l$ linear system, where k, l are respectively the number of edges and faces of the simplicial complex. If we let $\mu = \max(k, l)$, such a system can be solved by a linear solver in $O(\mu^2)$. Now, when a path is being analyzed in Algorithm B3, we need to compute at most m bounding chains, where m is the number of elements of $HomologyClusters$ (line 5). Therefore, the complexity of phase 1 is $O(\mu^2 Nm)$ here N is the number of paths. Finally, in each iteration of phase 2, we perform one vector addition, which has complexity $O(\mu)$ and compute the area of the resulting chain. Recall that calculating the area of a bounding chain, amounts to an inner product, and therefore also has complexity $O(\mu)$. Since this is performed for every possible pair of indices, the complexity of phase 2 is $O(N^2\mu)$, which brings the total complexity of Algorithm B3 to $O(N^2\mu + \mu^2 Nm)$.

In the next section we describe the results that guarantee the correctness of our method.

2.2 Theoretical framework

Let $p = p_0, p_1, \ldots, p_n$ be an edge path in X, that is, a list of vertices of the X such that $[p_i, p_{i+1}] \in X^{(1)}$. Now we observe, as pointed out in [18], that any such path can be represented by a 1–chain in $C_1(X) = \mathbb{R}^{b_1}$, i.e. a vector of the form $\tilde{p} = (\tilde{p}_0, \ldots, \tilde{p}_{b_1})$ where $\tilde{p}_i$ is the number of times the i–th edge is traversed in the path counted with orientation[5]. Further, note that given two paths $p = p_0, \ldots, p_n, \quad q = q_0, \ldots, q_m$, if $p_0 = q_0$ and $p_n = q_m$ then $\partial(\tilde{p} - \tilde{q}) = 0$.

So, given two edge-paths p, q on a simplicial complex X we say that they are *homologous* if there exists a 2–chain c such that $\partial c = p - q$, which brings us to the following definition:

[5]That is, if the i–th edge is of the form $[e, e']$ then $\tilde{p}_i = p_+ - p_-$ where p_+ is the number of times the edge is traversed from e to e', and p_- is the number of times the edge is traversed in the from e' to e.

Definition 6. *Define the homology area path distance on X as*

$$d_{H(X)}(p,q) = \min_{\partial c = p-q} \text{area}(c) \tag{B.3}$$

where $\min_{x\in\emptyset} x = +\infty$.

Proposition 3. *Let X be a simplicial complex where each 2–simplex has non-negative area, then $d_{H(X)}$ is symmetric and satisfies the triangle inequality, i.e.:*

- $d_{H(X)}(p,q) = d_{H(X)}(q,p)$
- $d_{H(X)}(p,q) + d_{H(X)}(q,r) \geq d_{H(X)}(p,r)$

Proof. Let c be a bounding chain of $p-q$, then due to linearity of ∂, $\partial(-c) = q-p$ and $\text{area}(c) = \text{area}(-c)$ meaning that $d_{H(X)}(p,q) = d_{H(X)}(q,p)$.

Assume p,q and r,q are homologous pairs of paths, then there exist 2–chains $c_{p,q}$ and $c_{r,q}$ such that $\partial c_{p,q} = p-q$ and $\partial c_{q,r} = q-r$. This implies that $\partial(c_{p,q}+c_{q,r}) = p-q+q-r = p-r$ and therefore $c = (c_{p,q}+c_{q,r})$ is a bounding chain for $p-r$. Furthermore we see that $c_i \neq 0$ only if $(c_{p,q})_i \neq 0$ or $(c_{q,r})_i \neq 0$ and therefore $\text{area}(c) \leq \text{area}(c_{p,q}) + \text{area}(c_{q,r})$, which implies $d_{H(X)}(p,r) \leq d_{H(X)}(p,q) + d_{H(X)}(q,r)$. □

This defines a distance between edge paths that is only finite for paths that are homologous. A shortcoming of this, is that only homologous paths that have the same first and last points may have finite area. However, this can be mitigated using relative homology.

More specifically, we recall from [18], that by defining *goal regions* A in the space X (associated to a subcomplex) and computing the first relative homology group $H_1(X,A)$ we define instead the homology class of 1–chains with boundary contained in A, which correspond to paths beginning and ending in A.

Remark 1. *By Lemma 3, we can compute the homology group $H_1(X/A)$ instead of $H_1(X,A)$ as these are canonically isomorphic. This has a practical advantage, since X/A has less simplices then X, which reduces the dimension of the boundary matrices $C_1(X/A), C_2(X/A)$ relative to those for $C_1(X,A)$ and $C_2(X,A)$, making the computation of solutions to* (B.3) *more efficient.*

Algorithm B2 shows how one can create a quotient complex from a simplicial complex and the characteristic function of a closed region A.

Now, given a 2–dimensional simplicial complex X and a subcomplex A, we note that any 2–simplex in $\sigma \in (X/A)^2$ is a quotient of some 2–simplex $\sigma' \in X^2$ and hence we can define $\text{Area}(\sigma) = \text{Area}(\sigma')$, which then allows us to implement minimum area homology on X/A.

[7] presented a framework to compute a minimum area bounding chain on a general simplicial complex by computing the solution to equation (B.2) and propose several methods by which this may be achieved. Particularly, they propose

computing least squares solutions to equation (B.2), as this can be done efficiently on sparse systems[6]. However, in a general setting, fast methods such as least squares may yield chains that do not have minimal area. In what follows, we prove that if the chain complex X can be embedded in $\mathbb{R}^2$, there is at most one bounding chain for any 1–cycle, thereby proving that in such cases approximate methods yield exact solutions. In Proposition 4 we generalize to quotients over sets A with no holes.

Lemma 4. *Any simplicial complex X which can be embedded into $\mathbb{R}^2$ satisfies $H_2(X) = 0$.*

Proof. Let X be a simplicial complex that can be embedded into $\mathbb{R}^2$ and X^c denote (a triangulation of the closure of) its complement, then using the Mayer-Vietoris sequence of $\mathbb{R}^2 = X \cup X^c$, we have:

$$\begin{aligned}\cdots \to H_3(\mathbb{R}^2) \to H_2(X \cap X^c) \to \\ \to H_2(X) \oplus H_2(X^c) \to H_2(\mathbb{R}^2) \to \cdots\end{aligned}$$

Now, recall that $H_i(\mathbb{R}^k) = 0 \quad$ for $i > 0$ ([9]). Note also that $X \cap X^c = \partial X$, therefore any cell in $X \cap X^c$ lies in the boundary of some cell in X or X^c making it at most, 1–dimensional which implies $X \cap X^c$ is a 1–dimensional subcomplex of X and X^c, and so $H_2(X \cap X^c) = 0$ (as it has no 2–cells), hence we conclude that the sequence:

$$0 \to H_2(X) \oplus H_2(X^c) \to 0$$

is exact, and $H_2(X)$ is a summand of 0 and hence $H_2(X) = 0$. □

Proposition 4. *Let X be a simplicial complex which can be embedded in $\mathbb{R}^2$, and A a subcomplex of X such that $H_1(A) = 0$. Then, given any 1–cycle c, there exists, at most one 2–chain s, such that $\partial s = c$.*

Proof. To begin, consider the long exact sequence in homology of the pair (A, X):

$$\begin{aligned}\cdots \to H_{i+1}(A) \to H_{i+1}(X) \to \\ \to H_{i+1}(X, A) \to H_i(A) \to \cdots\end{aligned}$$

Particularly for $i = 1$, $H_i(A) = 0$ and so we have:

$$\cdots \to H_2(A) \to H_2(X) \to H_2(X, A) \to 0 \to \cdots$$

So, using Lemma 4 we know that $H_2(A) = H_2(X) = 0$ and so $H_2(X, A) = 0$. Finally, given any 1–cycle c let s, s' be two 2–chains satisfying $\partial s = \partial s' = c$, then $t = s - s'$ is a 2–cycle, and since $H_2(X, A) = 0$, we conclude that t is the boundary of some 3–chain. However, since X can be embedded in $\mathbb{R}^2$, it has no 3–simplices, from which we can conclude that $t = 0$ and $s = s'$. □

[6] using for example the `LSQR` algorithm [16].

We now show how we can use this result to quickly calculate bounding chains between paths.

Corollary 3. *Let X be a two-dimensional simplicial complex and A a subcomplex with $H_1(A) = 0$, then for three one chains p, q, r such that $c_{p,q}$ and $c_{p,r}$ are the minimum area bounding chains of $p - q$ and $p - r$, respectively, then $(c_{p,r} - c_{p,q})$ is the minimum area bounding chain of $q - r$.*

Proof. Recall from the proof or Proposition 3 that $(c_{p,r} - c_{p,q})$ is a bounding chain of $q - r$, (this comes from linearity of the boundary operator). Furthermore, from Proposition 4 if there is a bounding chain of $q - r$ then it is unique, therefore, $(c_{p,r} - c_{p,q})$ must be the minimum area bounding chain of $q - r$. □

Finally, since our main goal is to cluster paths on a simplicial complex, we note that many algorithms used for path clustering rely on computing distances between all pairs of paths. Specifically, if we want to cluster the paths $p_0, p_1, \ldots, p_N$ with a distance function d, then we need to compute:

$$\begin{aligned} Distarray = [\, & d(p_0, p_1), \ldots, d(p_0, p_N), d(p_1, p_2), \ldots \\ & \ldots, d(p_1, p_N), \ldots, d(p_{N-1}, p_N)\,] \end{aligned}$$

which corresponds to the entries of the matrix $[d(p_i, p_j)]_{i,j=0}^{N}$ that lie above the main diagonal. In the case that the distance function is hard to compute, as it often is in the case of path distances ([12]), producing such a distance array can take a long time. Corollary 3 provides a way to mitigate this cost when using the homology area distance function, and we exploit this fact in Algorithm B3 to avoid calculating unnecessary bounding chains.

3 Experiments

To test our framework, we used two distinct environments. First, we created a synthetic dataset described in Section 3.1 and tested our framework on it. Secondly, in Section 3.2 we used our framework on real data, namely on the set of tracks of pedestrians on the computer science forum at Edinburgh University [8], a small subset of which can be seen in Figure B.6. We compare the output of our approach to that of Dynamic Time Warping, over a single homology cluster for clarity.

3.1 Toy example

The space created for the synthetic dataset comprises a bounded region of the plane with two holes. The simplicial complex was obtained by random sampling on this area. The relative region we chose comprises two rectangles along the side so that we can observe different clusters of paths going left to right.

In order to be able to observe the characteristics of clustering by relative homology previously identified in [18], as well as the distinction between different sub-clusters (using minimum area homology), we synthesized paths satisfying:

- Each path begins in one of two small areas in the relative region to the left (we call them O_1 and O_2).
- Each path ends in one of two small areas in the relative region to the right (we call them T_1 and T_2).
- Each path passes through a preselected (randomly chosen) point in a neighborhood of the middle vertical axis of the space (comprising three connected components which we call M_1, M_2, and M_3).

Paths were then computed by randomly drawing a triple from $O_i \times M_j \times T_k$, and interpolating these points with a Gaussian process using a Gaussian kernel function. By choosing the paths in this way we have access to a ground truth model which is simply given by the triple (i, j, k). This environment is depicted together with the ground truth in Figure B.5.

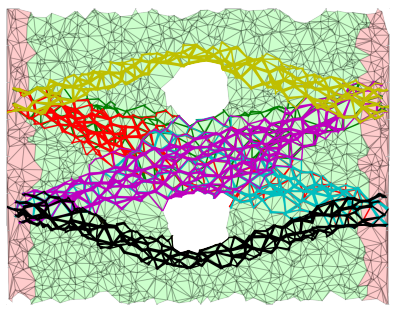


Figure B.5: Randomly generated simplicial complex with two holes (in white), and a set of randomly generated paths connecting the relative regions of the complex.

As expected, we observed that relative homology alone was enough to detect the different clusters depending on j. Furthermore, by calculating the distance array using homology area, we were able to fully recover the ground truth, via complete linkage clustering, to distinguish the four clusters with $j = 2$ which are indistinguishable using relative homology class alone. For comparison, we computed the distance array using DTW [19] and also performed complete linkage clustering, which was also able to recover the ground truth.

3.2 Edinburgh dataset

In this experiment, we randomly selected 2000 paths from the dataset of trajectories collected in the computer science forum of Edinburgh University (Figure B.6 [8]).

Figure B.6: Picture of the Edinburgh computer science forum overlaid with a set of paths from the dataset. The areas shaded in red were used to generate the relative region for the homology area clustering.

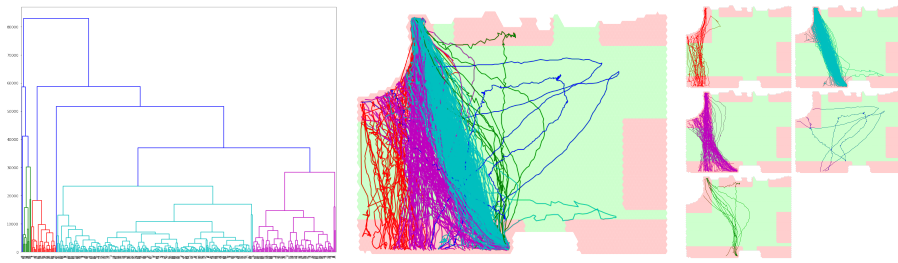


Figure B.7: One of the relative homology classes of paths from [8] clustered according to homology area using complete linkage clustering. From left to right we have the associated dendrogram, the paths colored according to the cluster in the dendrogram, and the individual clusters. The dark blue "cluster" consists of 3 outliers that have been singled out at this resolution level.

From these paths, we constructed a simplicial complex that fully covered the area around them as shown in Figure B.7, and the relative regions were chosen so that they contain the endpoints of most paths (those that were not encompassed were discarded). To create the complex, we made a regular triangulation of the area around the paths using equilateral triangles with side length of 7 pixels, this resulted in a complex with a total of 6,502 points, 19,141 edges and 12,640 faces, which resulted in a quotient complex comprising 4,478 points, 13,344 edges, and 8,861 faces meaning that computing a bounding chain amounts to solving a $13,344 \times 8,861$ linear system. Now, although the area depicted in Figure B.7 has no holes, we recall that the quotient complex has holes associated to the pairs of connected components of the relative region (See Figure B.3), namely it has non-zero homology classes associated to paths going from one relative region to another.

In Figures B.7 and B.8, we show the result of clustering the paths of a single

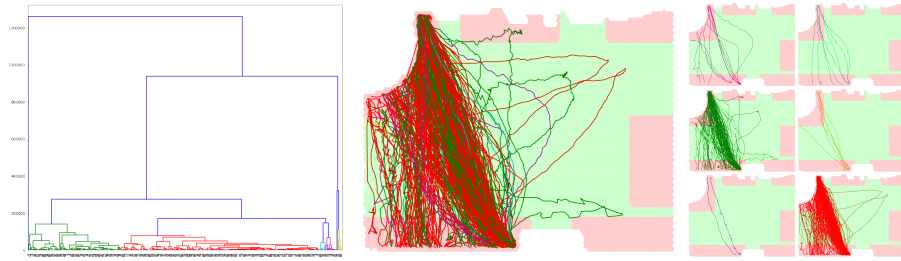


Figure B.8: The same homology cluster as the one used in Figure B.7, now clustered with complete linkage clustering according to the DTW distance. Again, from left to right we have the associated dendrogram, the paths colored according to the cluster in the dendrogram, and the individual clusters. Just as in Figure B.7 the blue "cluster" consists of 2 outliers that have been singled out at this resolution level.

relative homology class by homology area and dynamic time warping[7] distance respectively. To cluster the paths, we computed the pairwise distances using the respective distance function, and use the produced distance array to cluster the paths using complete linkage clustering. The threshold used to separate classes from the for the complete linkage clustering was chosen so that both dendrograms would split into 7 clusters.

In order to test the performance of our algorithm we computed the array of distances using DTW, FastDTW [19] and our implementation of Homology area, and noted the times in Table B.1. These tests were performed using a single core of an Intel Core i7–4790K @ 4.00GHz with 32Gb RAM. Our algorithms were fully implemented in Python 3.5, and for comparison we used the implementations of DTW and FastDTW from [20]. From these results we observe that Homology area outperforms (in Total time) DTW and FastDTW. The reason for this is that, even though it takes more time to calculate each individual pairwise distances with Homology Area, by using Algorithm B3, most of the bounding chains needed to compute the distance array are obtained as linear combinations of a minority of bounding chains. Another point to take into account, is that our algorithm uses the fact that the relation "two paths are homologous" is n equivalence relation to avoid calculating bounding chains that we know do not exist.

4 Conclusions

In this paper, we extend topological methods for path clustering to provide a compatible path distance. Furthermore, we were able to observe that the proposed method is able to cope with large numbers of paths by leveraging the fact that

[7] We use the implementation of DTW found in [20]

Algorithm	Average time (ms)	Total time (h:m)
Homology Area	90.809 ms	0 h 17 m
FastDTW	5.301 ms	2 h 42 m
DTW	18.152 ms	10 h 21 m

Table B.1: Time elapsed in computing *Distarray* with DTW, fastDTW, and Homology area. "Average time" denotes the average amount of time to compute the distance between two paths (Averaged over 10000 computations of distances between paths randomly drawn from the Edinburgh dataset [8]), whereas "Total time" denotes the total amount of time spent in computing the distance array for a set of 2000 paths.

not every distance needs to be computed from scratch, thereby reducing the time needed to produce a full distance array (essential for distance-based clustering) by an order of magnitude.

Acknowledgements The work in this paper was supported by the European Union through the grant H2020-FETPROACT-2014, 641321 (socSMCs), the Knut and Alice Wallenberg Foundation and the Swedish Research Council. The authors would like to thank Erin Wolf Chambers for her contribution to the research and structure of this paper.

References

[1] Saeed Aghabozorgi, Ali Seyed Shirkhorshidi, and Teh Ying Wah. Time-series clustering – A decade review. *Information Systems*, 53, oct 2015.

[2] S. Atev, O. Masoud, and N. Papanikolopoulos. Learning Traffic Patterns at Intersections by Spectral Clustering of Motion Trajectories. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, October 2006.

[3] S. Atev, G. Miller, and N.P. Papanikolopoulos. Clustering of Vehicle Trajectories. *IEEE Transactions on Intelligent Transportation Systems*, 11(3), September 2010.

[4] Aude Billard, Sylvain Calinon, Rüdiger Dillmann, and Stefan Schaal. Robot Programming by Demonstration. In *Springer Handbook of Robotics*. Springer Berlin Heidelberg, 2008.

[5] Christopher M. Bishop. *Pattern recognition and machine learning*. Springer, New York, 2006.

[6] L. Brun, A. Saggese, and M. Vento. Dynamic Scene Understanding for Behavior Analysis Based on String Kernels. *IEEE Transactions on Circuits and Systems for Video Technology*, 24(10), October 2014.

[7] Erin Wolf Chambers and Mikael Vejdemo-Johansson. Computing minimum area homologies. In *Computer Graphics Forum*, volume 34. Wiley Online Library, 2015.

[8] Robert Fischer. *Edinburgh Informatics Forum Pedestrian Database.* [Online] `http://homepages.inf.ed.ac.uk/rbf/FORUMTRACKING/`.

[9] Allen Hatcher. *Algebraic topology.* Cambridge University Press, 2002.

[10] Eamonn J. Keogh and Michael J. Pazzani. Scaling Up Dynamic Time Warping for Datamining Applications. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, New York, NY, USA, 2000. ACM.

[11] D. Makris and T. Ellis. Learning semantic scene models from observing activity in visual surveillance. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 35(3), June 2005.

[12] B. Morris and M. Trivedi. Learning trajectory patterns by clustering: Experimental studies and comparative evaluation. In *IEEE Conference on Computer Vision and Pattern Recognition, 2009. CVPR 2009*, June 2009.

[13] B.T. Morris and M.M. Trivedi. A Survey of Vision-Based Trajectory Learning and Analysis for Surveillance. *IEEE Transactions on Circuits and Systems for Video Technology*, 18(8), August 2008.

[14] Müllner, D. (2011) Modern hierarchical, agglomerative clustering algorithms. [Online] arXiv:1109.2378.

[15] *OpenStreetMap.* [Online]`https://www.openstreetmap.org`

[16] Christopher C. Paige and Michael A. Saunders. LSQR: An Algorithm for Sparse Linear Equations and Sparse Least Squares. *ACM Trans. Math. Softw.*, 8(1), March 1982.

[17] M. Pfeiffer, M. Schaeuble, J. Nieto, R. Siegwart, and C. Cadena. From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017.

[18] F. T. Pokorny, K. Goldberg, and D. Kragic. Topological trajectory clustering with relative persistent homology. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, May 2016.

[19] Stan Salvador and Philip Chan. Toward Accurate Dynamic Time Warping in Linear Time and Space. *Intell. Data Anal.*, 11, October 2007.

[20] Kazuaki Tanida. Fastdtw. [Online]`https://pypi.python.org/pypi/fastdtw`

Paper C

Long-term Prediction of Motion Trajectories Using Path Homology Clusters

J. Frederico Carvalho, Mikael Vejdemo-Johansson,
Florian T. Pokorny and Danica Kragic

Abstract

In order for robots to share their workspace with people, they need to reason about human motion efficiently. In this work we leverage large datasets of paths in order to infer local models that are able to perform long-term predictions of human motion. Further, since our method is based on simple dynamics, it is conceptually simple to understand and allows one to interpret the predictions produced, as well as to extract a cost function that can be used for planning. The main difference between our method and similar systems, is that we employ a map of the space and translate the motion of groups of paths into vector fields on that map. We test our method on synthetic data and show its performance on the Edinburgh forum pedestrian long-term tracking dataset [1] where we were able to outperform a Gaussian Mixture Model tasked with extracting dynamics from the paths.

1 Introduction

In order for robots to share their environment with people, it is necessary for them to reason about human motion, and use this capability to not inconvenience those they share their workspace with. We propose to address this problem through a motion prediction algorithm that leverages previously observed paths to predict the future motion of partial paths in an unsupervised manner.

Intuitively, we can formulate the problem as a model learning problem, where we hypothesize that human paths follow a certain model $M(\theta)$ and we want to find θ so that the error between the path predicted by the model, and the original path is minimized.

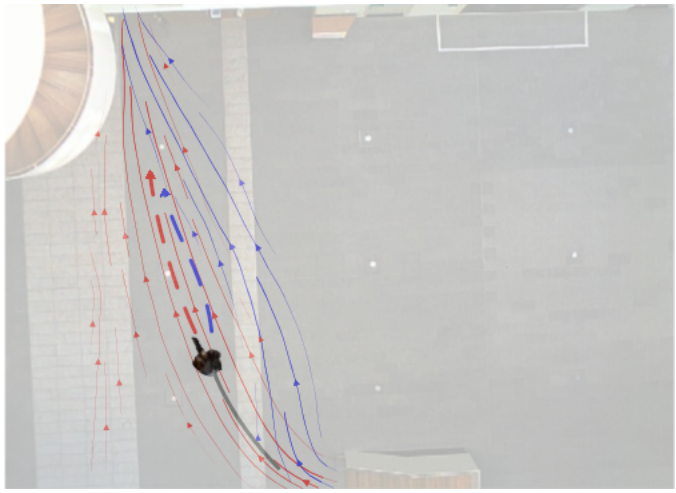


Figure C.1: Illustration of our method, a pedestrian path in an environment, together with two vector fields generated by pedestrian paths (red and blue), and the resulting different predicted paths (dashed).

Current methods for this type of motion prediction fall into two categories, i) those which rely on the dynamics of motion, assuming that a subject moves in efficient paths, i.e. straight lines, and any deviations from such motions are due to obstacle avoidance which may include other agents [2–5]. ii) those that rely on training parametric models such as neural networks or Hidden Markov Models (HMM) that produce a prediction given past observations [6–9].

In the first instance, by assuming that human motion only deviates from a straight line to avoid obstacles, the methods ignore the possibility that subjects may have preferred pathways, and place the onus of predicting deviations from efficiency on the exactness of the tracker, or the environment model. Whereas in the second case, the obtained predictions may be represented in the form of a probability density as in [9] or a static model of occupancy as in [6] which necessitate one to be aware of the threshold of occupancy probabilities when using such methods for planning.

In this paper we expand our previous work in path clustering [10,11] and compute vector fields which represent the average motion of paths in each cluster. We exploit the nature of our distance metric, which ensures a cluster should cover a small area of the environment to ensure that different vector fields encode motions in different parts of the workspace. We employ the resulting fields to predict motion by integrating the field that most closely resembles the path observed so far.

To the best of our knowledge, our method is the first to employ insights obtained from the topology of the space of paths to extract higher order dynamics and use them for motion prediction.

2 Related work

Motion prediction is a fairly well-studied task and is closely related to object tracking, therefore rather similar methods can be used to address it. In [5] the authors present a Kalman filter based method for predicting the motion of pedestrians in a 2D scene. In [8] the authors use HMMs to predict the motion of construction workers in a cluttered environment in order to improve building site safety. Similarly, in [9] the authors employ an HMM to the motion of people in an office environment, using both the immediate and far past observations to obtain a distribution over future observations. Similarly, recurrent neural network models, have been used in [7] to infer driver intention in a roundabout. In [12] the authors employ Support Vector Machines (SVM) and K-means clustering with respect to the edit distance on a dataset of paths as a training step, subsequently future motions are predicted from a partial path by matching it to one of the clusters.

One important application of motion prediction is the task of compliant motion planning, since by predicting the motion of a pedestrian in a scene, a robot is able to plan its path so as to not interfere with the pedestrian. In [13] the authors cluster paths to extract motion patterns using an expectation-maximization (EM) algorithm, these models are then used for prediction using a HMM which are used to synthesize compliant motion. An alternative approach was presented in [6] where the authors use a CNN to infer which parts of an environment are most traversed by humans, and use this in order to generate a robot motion that is non-obstructive. In [14] the authors use trajectory clustering in order to predict the motion of an agent, so that they are able to plan an intersecting path.

An area related to motion prediction is learning by demonstration. Here instead of using a model to infer where an object is moving toward, the model is used to instruct a robot how to move in order to perform some task. Popular approaches to this problem employ Gaussian Processes (GP) such as in [15] where the authors use local GP regression to perform a given task from a handful of demonstrations. Another common choice of model is a Gaussian Mixture Model (GMM) such as in [16] where the authors train a GMM from few demonstrations in order to obtain a compact model that generalizes a given task.

One way to extract motion primitives is through path clustering, where one intends to group a dataset of paths into subsets of paths that are pairwise similar. Generally, this is achieved through distance-based clustering methods, and a recent survey of such distance based methods can be found in [17]. Another possibility is to concentrate on path clusters that are fundamentally different with respect to their relationship to obstacles in the environments as in [10], and such methods can even be combined with a compatible distance function [18] to obtain more efficient clustering, as we showed in [11].

In this paper we present an algorithm for extracting motion primitives from datasets of paths in an efficient way. We do this by employing our path clustering pipeline from [11] to produce groups of paths that are deemed to be similar, and aggregating the observed velocities into a set of vector fields. To predict motion of

a path up to a point, we choose the vector field which is most similar to the path observed so far, and extrapolate the motion by integrating that vector field.

The remainder of the paper is structured as follows: In Section 3 we give a brief explanation of simplicial complexes and their homology, followed by a short explanation of our notation conventions on vector fields. These will be used to extend our path clustering pipeline from [11], and to establish the main contributions of the paper, respectively. In Section 4, we provide the aforementioned extensions to our path clustering pipeline, followed by an algorithm to calculate vector fields based on those path clusters as well as describe how we employ these vector fields for motion prediction. In Section 5 we describe our construction of synthetic datasets, and present the results from running our pipeline on such datasets, followed by the results of running the aforementioned pipeline on the dataset from [1].

3 Background

In this section we give a short overview of the background and terminology that will be used throughout the rest of the paper. Namely, we provide a short introduction to simplicial complexes and their homology, which will be necessary in order to prove that our alteration to the path distance metric from [11] produces the same clustering results. We also provide a short overview of our notation for vector fields which form the basis of our method for motion prediction.

3.1 Simplicial Complexes

Here we give a very short introduction to the notion of simplicial complexes. More detailed summaries can be found in [11, 19] as well as in more classical sources such as chapter 2 of [20].

Intuitively, a finite *simplicial complex* is a generalization of a triangulation. Formally it can be specified by a set X of subsets of $\{1, \dots, N\}$ such that for any $\sigma \in X$, and any non-empty $\tau \subset \sigma$, $\tau \in X$. Each element $\tau \in X$ is called a *face* of X.

If we consider a set of N points in $\mathbb{R}^2$, X can be seen as a triangulation where $\{x_i \mid i \in \sigma\}$ is the set of *vertices* of a *triangle* or *edge* when $|\sigma| = 3, 2$. However, we have to make a further restriction that convex hulls of $[\sigma] := \mathrm{Conv}(\{x_i \mid i \in \sigma\})$ satisfy $[\sigma] \cap [\tau] = \mathrm{Conv}(\{x_i \mid i \in (\sigma \cap \tau)\})$ i.e. only intersect at shared faces. The *dimension* of a simplex $\sigma \in X$ is defined given by $\dim(\sigma) = |\sigma| - 1$ and the dimension of the complex X is given by $\dim(X) = \max_{\sigma \in X} \dim(\sigma)$. We denote $X^{(i)}$ as the i-th *level* of X, containing all the i-dimensional simplices of X and $X^i = \bigcup_{j=0}^{i} X^{(j)}$, which is a simplicial complex called the i-th skeleton.

A simplicial complex X gives rise to a chain complex $\{C_i(X)\}_{i=1}^{\dim X}$. This corresponds to a sequence of vector spaces, where $C_i(X)$ has $X^{(i)}$ as a base together with a family of boundary maps $\partial_i : C_i(X) \to C_{i-1}(X)$ which send a basis element to a formal sum of its boundary elements with signs shifted according to the induced

orientation. A more extensive description can be found in chapter 2 of [20]. An element $c \in C_i(X)$ is called an *i-chain* of the complex, furthermore if $\partial_i c = 0$ then c is called a *i-cycle* and if there exists some $b \in C_{i+1}(X)$ such that $\partial_{i+1} b = c$, then c is called a boundary.

It can be shown that in this setup every boundary is a cycle, and that cycles and boundaries form subspaces of the $C_i(X)$. The i-th *homology* group of X consists of equivalence classes of cycles in $C_i(X)$ modulo the boundaries in $C_i(X)$.

When dealing with paths, we view these as an ordered list of (oriented) edges on the simplicial complex which are placed end-to-end. These we regard as 1-chains of X, with integer weights. For example, a weight of 2 in a particular edge indicates that the edge is traversed twice, and a weight of -1 indicates that the edge is traversed once but in the negative direction. We call a 1-chain obtained from such a path a *path chain*. Note that our definition of simplicial complex implies that any two vertices share at most one edge, therefore a path chain p is uniquely determined by an ordered list of vertices $p_0, p_1, \ldots, p_n$.

We say that two path chains p, q are homologous if $p - q$ is a boundary, i.e. if there exists a 2-chain $c_{p,q}$ so that $\partial c_{p,q} = p - q$. In this case $c_{p,q}$ is said to be the *bounding chain* of $p - q$. However, the requirement that $p - q$ forms a cycle is in general too restrictive, therefore it will be useful to consider *quotients* of X. The quotient X/A is defined from the simplicial complex X by identifying every simplex in the subcomplex $A \subset X$ with a single point in the quotient. Implying that the paths in X whose endpoints lie in A are in fact cycles in X/A.

3.2 Vector Fields

Given a subset of $X \subseteq \mathbb{R}^2$ we define a vector field V on X as a continuous function $V : X \to \mathbb{R}^2$. Such a vector field can model the motion of a particle by setting $\dot{x} = V(x)$. We define the (positive) flow of a vector field starting at x_0 as a function $\Phi_{V,x_0}(t) : J \to X$ where J is an interval in $\mathbb{R}$, containing 0 and $\Phi_{V,x_0}(t)$ satisfies $\frac{d}{dt}\Phi_{V,x_0}(t) = V(\Phi_{V,x_0}(t))$ and $\Phi_{V,x_0}(0) = x_0$. The existence of J and Φ_{V,x_0} in some region around x_0 is guaranteed by the Picard-Lindelöf theorem [21] provided that the vector field V is Lipschitz continuous in x in some neighborhood of x.

We employ a discrete approximation of vector fields in the plane that consists of an assignment $P \mapsto \mathbb{R}^2$ where P is a finite subset of $\mathbb{R}^2$. This can then be extrapolated to a vector field in each simplex of a triangulation of P by linear interpolation. Due to this approximation we guarantee that the obtained vector field is Lipschitz continuous.

4 Methods

We begin by describing a small alteration to our path clustering pipeline. In [11] we used a model of the quotient of the base space X by a subspace $A \subset X$ which encompasses the endpoints of the paths in the dataset. As noted in Section 3.1, this

allows us to treat path chains as cycles and use that to find if they are homologous. We then defined the distance between the paths to be either the area of the bounding chain if they are homologous, and infinity otherwise.

In the following we show that under mild assumptions a pair of paths in X/A can only be homologous if they begin in the same component of A and end in the same component of A. Therefore, under this condition the results from [11] are valid when instead of considering X/A we use the decomposition of A into its connected components $A_1 \cup, \ldots, \cup A_k$ and consider instead the *iterated quotient* $(\ldots(X/A_1)/\ldots)/A_k$ which we denote by $X/(A_1, \ldots, A_k)$. All proofs except for Lemma 5 are contained in appendix.

The following lemma establishes that the results we obtain are independent of the ordering of $A_1, \ldots, A_k$.

Lemma 5. *Let $A_1, \ldots, A_k$ be disjoint subsets of X and let $\mathbb{S}_k$ be the set of permutations of k elements, then for any $\sigma \in \mathbb{S}_k$ we have:*

$$X/(A_1, \ldots, A_k) \cong X/(A_{\sigma_1}, \ldots, A_{\sigma_k}).$$

For convenience, we will further restrict the subcomplexes $A_1, \ldots, A_k$ to satisfy the following:

Definition 7. *We say that a sequence of subcomplexes $A_1, \ldots, A_k \subseteq X$ are* well disconnected *if for every simplex $\sigma \in X$, $\sigma \cap A_i \neq \emptyset$ implies $\sigma \cap A_j = \emptyset$ for all $j \neq i$.*

The following proposition is a reformulation of Proposition 2 from [11] which establishes the uniqueness of bounding chains in our setting and guarantees that the distance function is well defined.

Proposition 5. *Let X be a simplicial complex which can be embedded in $\mathbb{R}^2$ and $A_1, \ldots, A_k \subseteq X$ disconnected subcomplexes such that $H_1(A_i) = 0$, then given any 1-cycle c of $X/(A_1, \ldots, A_k)$, there exists at most one 2-chain s such that $\partial s = c$.*

The following lemmas characterize the notion of cycle and boundary arising from path chains in an iterated quotient $X/(A_1, \ldots, A_k)$ when compared to the quotient X/A.

Lemma 6. *Given two path chains $p = p_0, \ldots, p_n$ and $q = q_0, \ldots, q_m$ in a simplicial complex, then $p - q$ is a cycle if and only if both $p_0 = q_0$ and $p_n = q_m$ or p and q are both cycles.*

Lemma 7. *Let X be a simplicial complex embedded in $\mathbb{R}^2$ and $A_1, \ldots, A_k$ be well disconnected subcomplexes so that $H_1(A_i) = 0$. Let c be a path chain in X, then the associated chain $[c]$ is a boundary in $C_1(X/(A_1 \cup \ldots \cup A_k))$ if and only if it is a boundary in $C_1(X/(A_1, \ldots, A_k))$.*

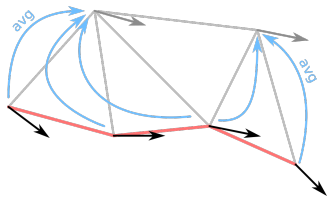


Figure C.2: Illustration of the process by which the vector field is expanded to neighboring vertices. The vertices along the line in red are in the path, and the black arrows on them indicate the value of the vector field along that path. The lines in light gray are the edges of the mesh, and the arrows in blue indicate which elements are averaged together before into the arrow in dark gray that represents the vector field at the opposite (w.r.t. the red path) end of the edge.

These results summarize the situation for path chains in the iterated quotient $X/(A_1, \ldots, A_k)$ for $X \subset \mathbb{R}^2$ and $A_1, \ldots, A_k$ well disconnected. Namely, two path chains p, q are homologous if and only if they were homologous in X/A, and if they are homologous, there is at most one bounding chain. This entails that our notion of area homology distance from [11] carries over to this setting without change.

4.1 Calculating Vector Fields

We define a vector field starting from the first order differences of the path and extrapolating to the rest of the space by using an underlying simplicial complex. In Algorithm C2 we provide the pseudocode for generating such a vector field from a path. The remainder of this subsection is dedicated to explaining how this algorithm works, and we defer to the next subsection to explain how we use the vector fields for motion prediction.

Definition 8. *Let $\gamma = (t_0, p_0), (t_1, p_1), \ldots, (t_n, p_n)$ be a path, then we define the* differential *of γ at a given point p_i to be given by:*

$$d\gamma(t_i) = \frac{p_{i+1} - p_i}{t_{i+1} - t_i}.$$

The differential defines a vector field associated to γ but only at the points p_i. To extrapolate this field to other points in $\mathbb{R}^2$ we first extrapolate it to neighboring points by averaging the contributions from neighbors (as illustrated in Fig. C.2).

The main idea of Algorithm C2 is to establish the vector field along the path, and then use the edges of the mesh S to expand it to the rest of the space. This allows one to define the vector field as a simple $N \times 2$ matrix where N is the number of vertices S. For points which are not vertices of S the value of the vector field can be established through linear interpolation.

Algorithm C2 begins by initializing the vector field vP in line 3 to be zero at every point, and the variable dP in line 2 which stores the differential of the path

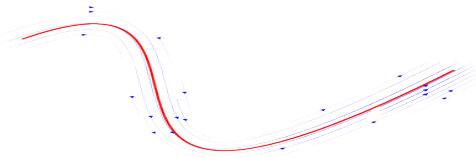


Figure C.3: Example of a path and the associated vector fields. Depicted are a simple curve γ in red with its associated vector field V_γ in blue computed by Algorithm C2. As we can see, the field forms a "corridor" around the path.

P. Two further auxiliary variables are set: $Npts_i$, which is the number of points that are averaged when producing the estimation at the i-th vertex of the mesh, and Lv_i which indicates at which "level" the i-th point has been calculated. Here i is assumed to range over the index of all points in V.

In lines 5 to 10, the vector field is calculated at the nearest neighbors of the points in the path, using to this effect the values of dP. The expansion of the vector field to the mesh begins in earnest with the loop in lines 12 to 17. In it, the points which were previously assigned a vector field value through the nearest neighbor query, are assigned level zero, whereas their neighbors are inserted into the variable $NextLv$ which contains the next "level" of points to be analyzed.

Finally, the main loop in lines 19 to 35 proceeds by checking the neighbors of each element of $NextLv$ which has not been previously analyzed, and averaging the values of the vector field vP at those neighbors that are at a level lower than the current level $CurrLv$. The neighbors that have not been assigned a level yet are then added to the new level set $NewLv$. At the end of this loop the new level set replaces the level set and the current level is incremented.

Finally, in order to obtain the vector field associated to a cluster of paths $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$ we will have to average the different vector fields. To this end, we need to make the different fields comparable, which we achieve by renormalizing them so that the highest velocity is 1:

$$W_\gamma(x) = \frac{V_\gamma(x)}{\sup_{y \in X} V_\gamma(y)}$$

Similarly, we obtain the vector field V_Γ associated to the cluster Γ by combining the vector fields W_γ and renormalizing them the same way:

$$W_\Gamma(x) = \sum_{\gamma \in \Gamma} W_\gamma(x) \qquad V_\Gamma(x) = \frac{W_\Gamma(x)}{\sup_{y \in X} W_\Gamma(y)}$$

4.2 Cluster assignment and motion prediction

With the vector fields calculated we can finally assign a hypothesis cluster to a path that has only been partially observed. Namely, for each vector field we measure

"how parallel" it is to the path that is being examined, and the corresponding cluster is then the cluster that is assigned i.e. for a given path $\eta = (t_1, \eta_1), \ldots, (t_k, \eta_l)$ we assign the cluster:

$$\Xi(\eta) = \arg\max_{i=1,\ldots,k} \sum_{j=1}^{l} \langle d\eta(t_j), V_{\Gamma_i}(\eta(t_j)) \rangle$$

and we predict the path to be the integral path of the vector field starting at the last point in the path while maintaining the current velocity by (numerically) solving the integral equation:

$$\tilde{\eta}(t) = \eta(t_l) + \|d\eta(t_l)\| \int_{t_l}^{t} \frac{V_{\Xi(\eta)}(\tilde{\eta}(s))}{\|V_{\Xi(\eta)}(\tilde{\eta}(s))\|} ds$$

5 Experiments

In this section, we present our experimental results. We start with two synthetic examples, where the data is drawn from a known distribution. The first set exemplifies an ideal scenario where the data is well separated, and our score function is ideally suited to distinguish between the different sets of data. The second dataset contains more complex dynamics, and showcases situations in which our score function can underperform and be led to produce misclassification errors. Finally, in the real world data example, we show how our method performs on the dataset from [1].

We compare our method with simple dynamics based motion prediction methods, the first one simply integrates the current speed of the path into the future, i.e. given the path γ up to time t we predict that at time $t+s$ the path is at $\gamma(t)+s\dot{\gamma}(t)$. The second can only be used when there is a known goal region, which is the case for the synthetic datasets, and works by setting an attractor on a target S in the goal region but maintains the norm of the velocity, so in essence we solve the initial value problem:

$$\frac{d}{dt}\begin{pmatrix}\gamma(t+s)\\ \dot{\gamma}(t+s)\end{pmatrix} = \frac{\|\dot{\gamma}(t)\|}{\|\dot{\gamma}(t+s)\|}\begin{pmatrix}\dot{\gamma}(t+s)\\ \beta(S-\gamma(t+s))\end{pmatrix}$$

where the parameter β is chosen to minimize the prediction error. In the real world dataset we use an alternative vector field model, that consists of modeling the vector field associated to each path cluster using a GMM as in [22]. All integrations are calculated using an explicit fourth order Runge-Kutta method with a step size of 0.01 on the synthetic datasets, and $(1/90)s^{-1}$ on the real-world dataset, this corresponds to ten steps per frame, as the paths were drawn from a tracking process running at 9 frames per second.

5.1 Simple Synthetic Data

In this test we employed a simple model where we draw paths in a 100×100 square in $\mathbb{R}^2$ (the distance units can be taken to be meters) and demark four regions:

- left near point $(0, 50)$ where all paths begin.
- right near point $(100, 50)$ where all paths end.
- top near point $(50, 100)$ which contains the midpoints of all paths in cluster 1.
- bottom near point $(50, 0)$ which contains the midpoints of all paths in cluster 2.

Each path is interpolated from three points (beginning, middle and end with timestamps 0, 0.5 and 1 respectively) using a GP, the resulting paths can be seen in Figure C.4. To draw a path from the distribution we first randomly choose either the top or bottom region, then we draw the prescribed three way points that define the path, and interpolate them as specified.

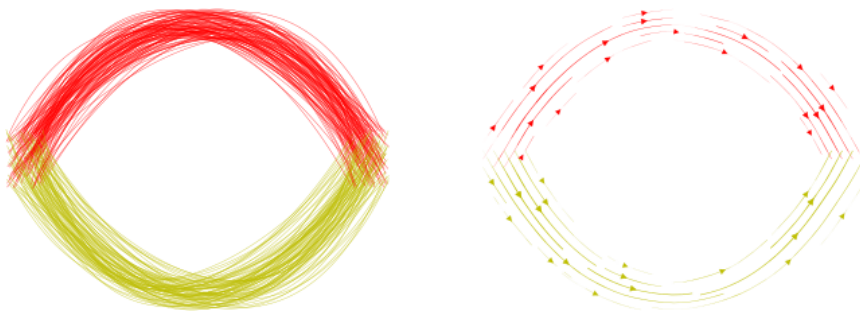


Figure C.4: Simple synthetic data set: The first cluster of paths is depicted in red, and the second one is depicted in yellow. To the right we show the stream plot with the vector fields associated to each cluster in the respective color.

To test in this scenario we sampled randomly 1000 paths from the distribution and clustered them into two clusters, and calculated the associated vector fields. We tested the performance of the resulting model on 100 more paths also drawn from the distribution.

In Figure C.5 we show an example of predictions from our method, together with the error incurred by both our method, and the constant-velocity and the attractor-towards goal models. The error is represented in time (of the path/predicted path which starts at the point where the prediction starts) versus distance between the ground-truth and the predicted path. As we can see our method is able to track the overall shape of a simple path, which on these instances allows it to outperform both other predictors.

In Figure C.6 we present the statistics of the errors for the three different models, namely we plot the prediction horizon (that is, amount of time *after* the beginning of the prediction) versus error (again, distance between the predicted path and the ground truth at the given prediction horizon). For each model we plot the average (solid line) and standard deviation (shaded region) of the error at any given

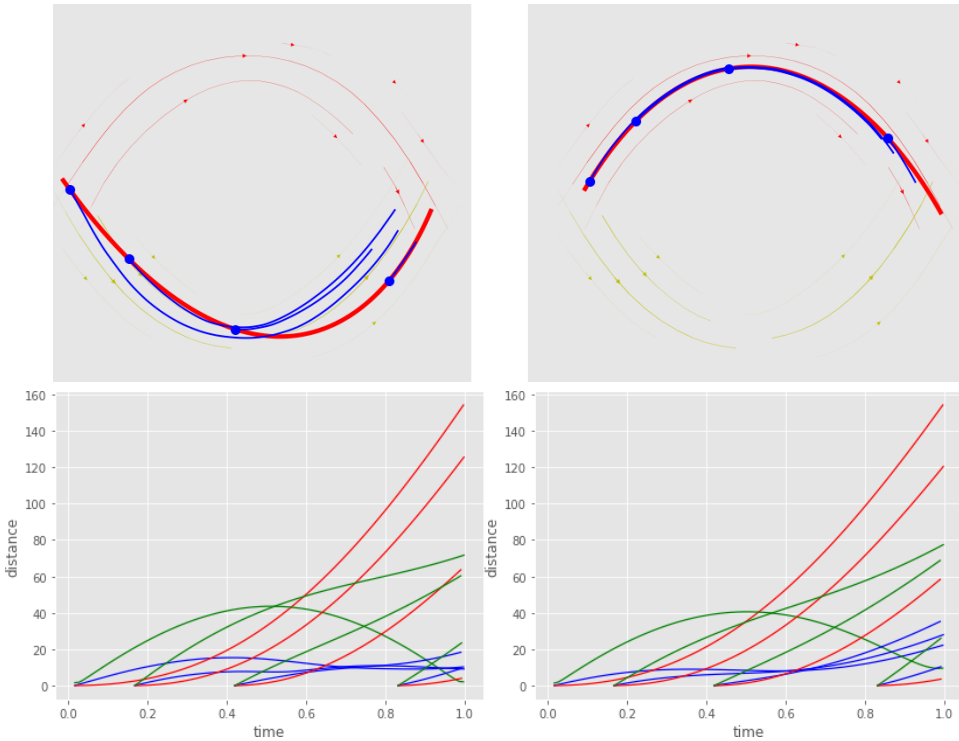


Figure C.5: Top row: Both plots contain a path in red drawn from the same distribution as the dataset used to calculate the vector fields. Each of the blue paths is predicted with our method starting from the large blue point at its extremity. Bottom row: Error for the predicted paths in the top row. The horizontal axis represents time, and the vertical axis represents the distance from the ground truth. Each of the blue,red and green lines represents the measured error when motion is predicted using our method (blue), constant velocity prediction (red) and an attractor at the goal region (green).

time. As we can see our observations from Figure C.5 remain valid when we look at statistics of the error.

Note that in this case we used partial information from how the dataset was generated to calculate the vector fields, namely we used two path clusters, which generated two vector fields, each of which closely tracks the associated cluster, therefore as long as the correct vector field is identified, the prediction error should remain low. Furthermore, since the vector fields almost do not overlap in space, and when they do, they are mostly perpendicular to each other, identifying the correct vector field becomes a simple task. Next we show another synthetic example where this separation between fields is not well demarcated, and therefore incurs in higher

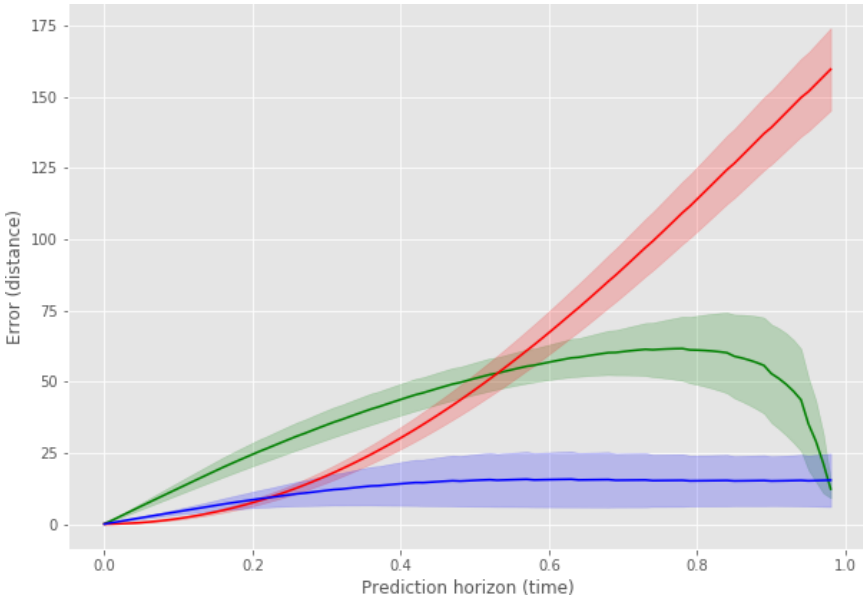


Figure C.6: Error statistics for the prediction models for the dataset in Figure C.4. Here we plot the statistics for the three different models, our model is in blue, the constant velocity prediction is in red, and the attractor model is in green. The statistics shown are the mean error (solid line) and the standard deviation region (shaded region).

error predictions for our model.

Algorithm C2: Calculate the vector field associated to a path

input : $S = (V, E)$
input : $P = (p_1, t_1), ..., (p_n, t_n)$
$Lv_i \leftarrow \infty$ where $i \leftarrow 1, \ldots, \text{size}(V)$;
$Npts_i \leftarrow 0$ where $i \leftarrow 1, \ldots, \text{size}(V)$;
$dP_i \leftarrow (p_i, \frac{p_{i+1}-p_i}{t_{i+1}-t_i})$ where $i \leftarrow 1, \ldots, n$;
$vP_i \leftarrow (0, 0)$ where $i \leftarrow 1, \ldots, \text{size}(V)$;
for $(p, dp) \leftarrow dP_0, dP_1, \ldots$ **do**
  $q \leftarrow \text{nearest}(V, p)$;
  $i \leftarrow \text{index}(q, V)$;
  $d \leftarrow dist(q, p)$;
  $vP_i \leftarrow vP_i + \text{DECAY}(d)dp$;
  $Npts_i \leftarrow Npts_i + 1$;
$NextLv \leftarrow \{\}$;
for $i \leftarrow 0, \ldots, \text{size}(V)$ **do**
  $n \leftarrow Npts_i$;
  if $n > 0$ **then**
    $vP_i \leftarrow vP_i/n$;
    $Lv_i \leftarrow 0$;
    append$(NextLv, neighborIdx(S, V_i))$;
$CurrLv \leftarrow 1$;
while $notEmpty(NextLv)$ **do**
  $NewLv \leftarrow \{\}$;
  for $i \leftarrow NextLv_0, NextLv_1, \ldots$ **do**
    if $Lv_i < CurrLv$ **then**
      skip
    $Nbh \leftarrow neighborIdx(S, V_i)$;
    $N \leftarrow 0$;
    for $j \leftarrow Nbh_0, Nbh_1, \ldots$ **do**
      if $Lv_j < CurrLv$ **then**
        $d \leftarrow dist(V_i, V_j)$;
        $vP_i \leftarrow vP_i + \text{DECAY}(d)vP_j$;
        $N \leftarrow N + 1$;
      else
        append$(NewLv, j)$;
    $vP_i \leftarrow \frac{vP_i}{N}$;
  $NextLv \leftarrow eliminateRepeated(NewLv)$;
  $CurrLv \leftarrow CurrLv + 1$;
return vP;

5.2 More Complex Synthetic Data

With this example we aim to show that our model is able to encode more complex dynamics than a simple single curved motion towards a target, and to show where prediction errors may arise in our model. To generate the data we use an almost identical procedure to the one before, except we have an extra "foreword" region near the point $(25, 0)$ where all paths go through. Once again, to draw a path from one cluster we draw four points (one from the beginning region, one from the foreword region, one from the top or bottom region, and one from the end region). The points are then given timestamps 0, 0.2 , 0.7 and 1 respectively, and are interpolated with a Gaussian process.

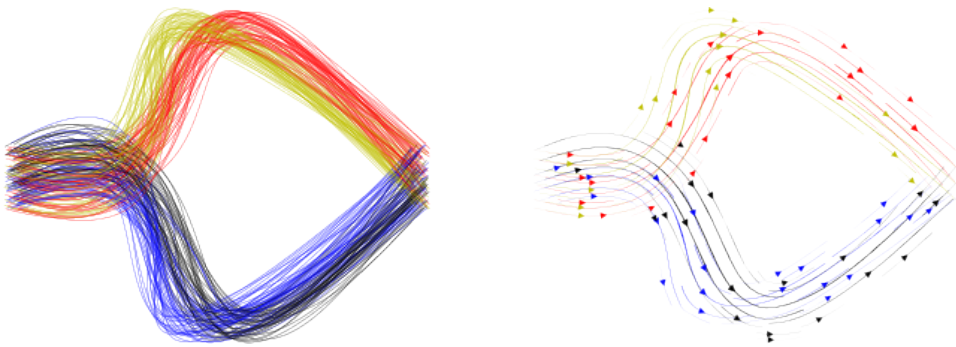


Figure C.7: A more complicated dataset, comprising four path clusters (red, blue, black and yellow). To the right we see the associated vector fields.

From the description of the generated dataset it would be expected that there would be only two clusters as in the previous example, however, our clustering method proposed a model with four clusters as seen in Figure C.7 when maximizing the ratio of inter-cluster distance to in-cluster distance. In Figure C.8 both demonstrate our method in the top row, and compare it with the two simple dynamics methods used in the previous example in the bottom row. From looking at the predictions we notice that one of them follows the wrong vector field leading to a high prediction error. This happens because at the point of prediction all the fields are similar, this implies that the score functions will also have a similar value and leads to the possibility of misclassification which may result in higher error than the simple models.

However, when we once again analyze the error statistics for the predictions in this dataset, we see that our model once again outperforms the simple dynamics models. However, the mean and standard deviations of the error at all horizons are now larger than they were in the simple dataset which comes from the fact that the possibility of misclassification leads to larger errors at all prediction horizons.

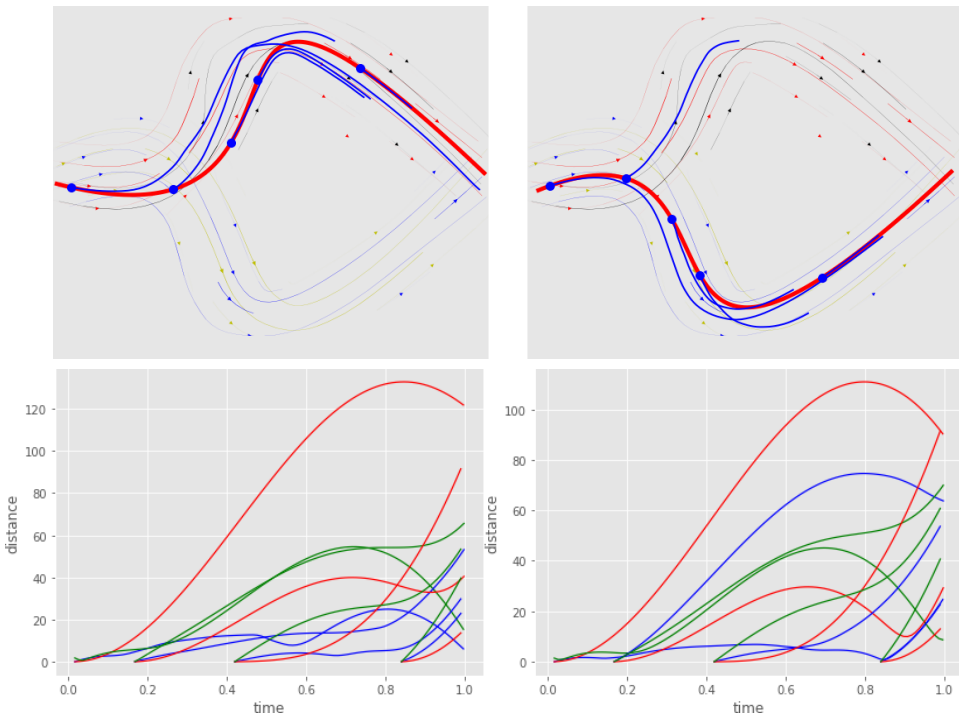


Figure C.8: Top row: Two paths from the dataset Figure C.7 together with predictions performed using our method, as in the top row of Figure C.5. Bottom row: Error for the predicted paths in the top row, the legend is the same as in the bottom row of Figure C.5.

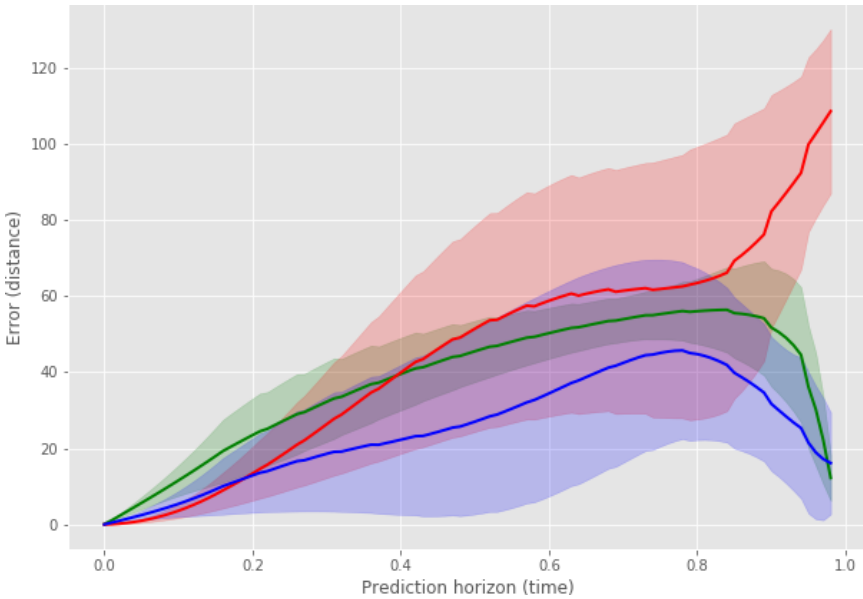


Figure C.9: Error statistics for the prediction models for the dataset in Figure C.7. Here we plot the statistics for the three different models, our model is in blue, the constant velocity prediction is in red, and the attractor model is in green. The statistics shown are the mean error (solid line) and the standard deviation region (shaded region).

5.3 Real-world Data

We now repeat the analysis performed for the synthetic datasets, using instead the Edinburgh dataset [1]. This dataset comprises tracks of pedestrians in Edinburgh University, and part of it is pictured in Figure C.10 (left).

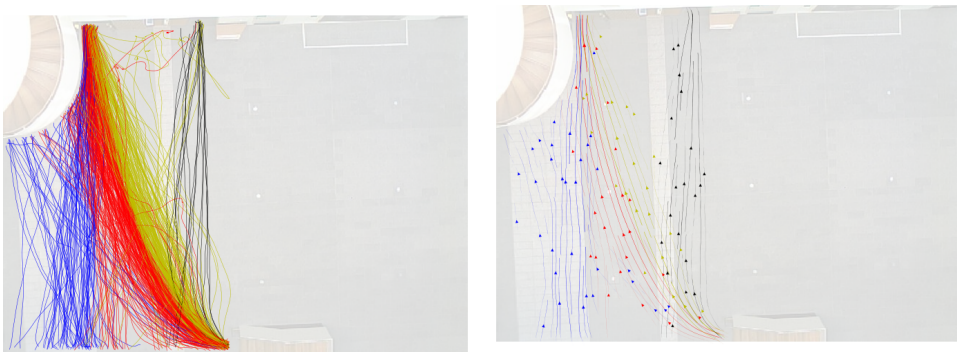


Figure C.10: Subset of the path clusters (left) and corresponding vector fields (right) associated to paths from the Edinburgh pedestrian tracking dataset [1] which share origin and destination regions.

However, the tracking data contains high-frequency noise from measurement errors which is detrimental to calculating velocities. For example, the positions output by the tracker are measured in pixels, so even when an object is stationary the position is expected to change due to naturally shifting lighting conditions. We thus need to pre-process the data so that velocities vary smoothly as expected. To this end we assume the noise is white, so that it can be filtered by using a simple 5-point moving average ($\hat{x}_t = (x_{t-2} + \cdots + x_{t+2})/5$).

In Figure C.10 (right) we show a picture of the vector fields generated from the paths in the left, as we can see the vector fields have a large overlap, which results in misclassification errors, as we previously pointed out. As we also pointed out before, we compare our method to using the simple constant velocity prediction and a GMM vector field approach [22]. In this model we calculate all pairs $(\gamma(t), \dot{\gamma}(t))$ from paths γ in some cluster Γ, which are then assumed to be samples drawn from a random variable (X_Γ, V_Γ) with domain $\mathbb{R}^2 \times \mathbb{R}^2$ with a GMM distribution. To choose the vector field to integrate in order to predict a new path γ we select the cluster Γ that maximizes the likelihood $\sum_{t=0}^{T} \log(p_{(X_\Gamma, V_\Gamma)}(\gamma(t), \dot{\gamma}(t)))$ and we calculate the value of the vector field at point x as being the expected value $E(V_\Gamma | X_\Gamma = x)$.

However, when analyzing the predictions in Figure C.11, we see that once again, in the long term our method is able to attain a smaller error than simply following the current velocity or using the GMM vector field model. Upon a closer inspection and comparing with Figure C.10, we see that the paths on the top row are not always attributed to the same vector field. Namely the first two predictions of the

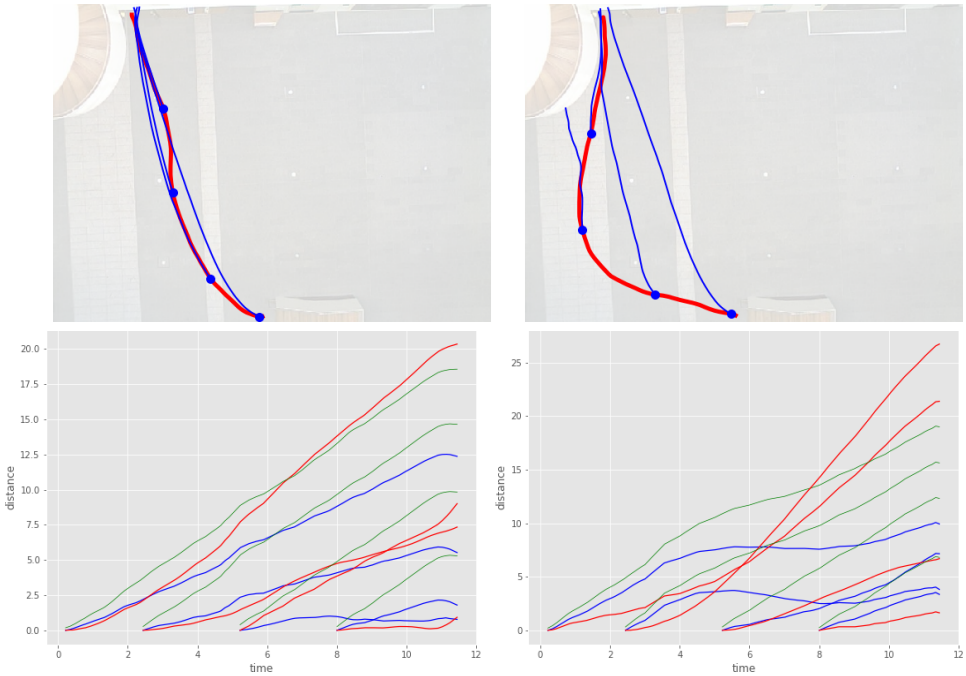


Figure C.11: Top row: two paths from the Edinburgh pedestrian tracking dataset and associated predictions using our method, starting at different points along the path. Bottom row: comparison of the prediction error using our method (blue) and constant velocity prediction (red), and GMM vector field integration (green).

path on the right follow the red cluster, the third prediction follows the blue cluster as the path curves upward more abruptly in a region where the red cluster does not.

Finally, the statistics in Figure C.12, second our observations from the analysis of Figure C.11 and we we can see that our method outperforms on average constant-velocity extrapolation and integrating the GMM vector field model, while also attaining lower variance.

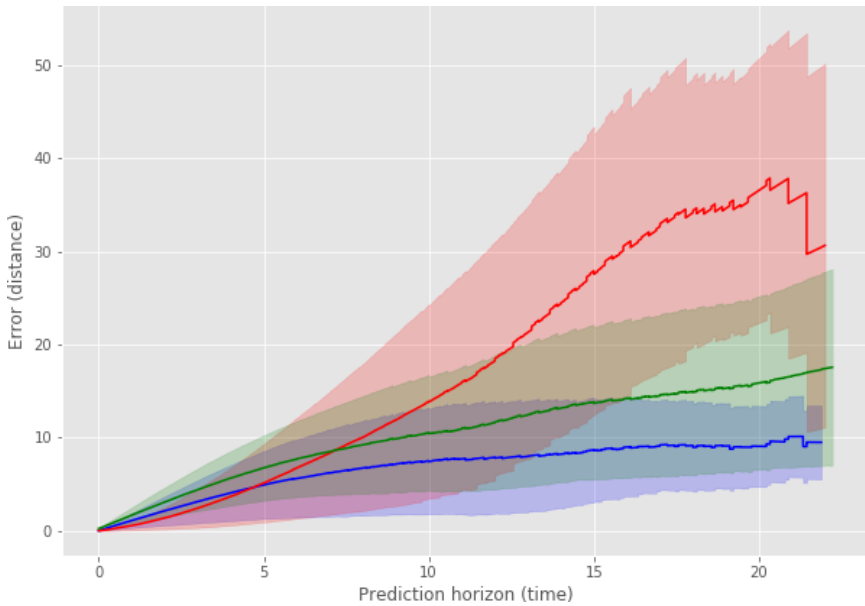


Figure C.12: Error statistics for the prediction models for the Edinburgh pedestrian tracking dataset. Depicted are the error statistics for our model (blue), constant velocity prediction (red) and the GMM vector field (green). The statistics shown are the mean error (solid line) and the standard deviation region (shaded region).

6 Conclusions and Future Work

In this paper we presented a method to extract motion primitives in a scalable way from large datasets of paths using piece-wise linear vector fields. We compared our method to other dynamics based methods, and observed that our approach is able to effectively capture more complex dynamics than conceptually simpler primitives, and in the tested datasets lead to more faithful predictions than a GMM vector field approach at all time horizons.

In future work we hope to improve our framework in several ways, namely by using sparse vector field models, instead of the dense model currently used, and using insights from sheaf theory [23] in order to provide more resilient prediction dynamics. We also intend to integrate our method into a tracking framework, for example a Bayesian filter to test the system performance with our system dynamics.

Appendix

A Proof of Proposition 5:

We follow an argument similar to the proof of Proposition 2 in [11]. Namely when $A \subseteq X$ is a subcomplex, we can build the exact sequence of the pair [20].

$$\cdots \to H_n(A) \to H_n(X) \to H_n(X/A) \to H_{n-1}(A) \to \cdots$$

Since this sequence is in reduced homology, and each A_i is of dimension 2 and embedded in $\mathbb{R}^2$, therefore has $H_2(A_i) = 0$, furthermore it satisfies $H_1(A_i) = 0$ and since it is connected, it also satisfies $H_0(A_i) = 0$ (since it is reduced homology). Therefore $H_n(A_i) = 0$ for every n and there is an isomorphism in homology between $H_n(X)$ and $H_n(X/A_i)$ furthermore since the A_i are disjoint, the subcomplex $A_i/A_j \subseteq X/A_j$ is isomorphic to A_i and so still satisfies $H_n(A_i) = 0$.

This implies that in the conditions of the proposition $H_n(X/(A_1, \ldots, A_k)) \cong H_n(X)$ for every n, and therefore $H_2(X/(A_1, \ldots, A_k)) = 0$ which implies that for any 1-chain c there exists at most one 2-chain s so that $\partial s = c$. □

B Proof of Lemma 6:

Since p is a path chain, $\partial p = (p_1 - p_0) + (p_2 - p_1) + \ldots + (p_n - p_{n-1}) = p_n - p_0$. In a similar fashion we have $\partial q = q_m - q_0$ and so $\partial(p - q) = 0$ if and only if $p_n - p_0 = q_m - q_0$, i.e. if and only if $p_n = q_m$ and $p_0 = q_0$ or $p_n = p_0$ and $q_m = q_0$. □

C Proof of Lemma 7:

Assume, without loss of generality that $k = 2$, and therefore the relative region has only two connected components A_1, and A_2. This decomposition induces a factorization of the quotient $X \to X/(A_1 \cup A_2)$ through the iterated quotient $X/(A_1, A_2)$ (which corresponds simply to the identification of the points representing each of the connected components). Call these maps ϕ and ψ:

$$X \xrightarrow{\phi} \frac{X}{A_1, A_2} \xrightarrow{\psi} \frac{X}{A_1 \cup A_2}$$

Since ϕ and ψ are simplicial maps, they induce chain homomorphisms $\phi_*([c]) = [\phi(c)]$, and $\psi_*([d]) = [\psi(d)]$. By definition of chain homomorphism, they commute with the differential, meaning $\partial \circ \psi_*([d]) = \psi_*(\partial[d])$. Therefore if $\phi_*([c])$ is a boundary, then it can be written as $\partial[b]$ for some 2-chain $[b]$ which means.

$$\psi_*(\phi_*([c])) = \psi_*(\partial[b]) = \partial(\psi_*([b]))$$

i.e. $\psi_* \circ \phi_*([c]) = [\psi \circ \phi(c)]$ is also a boundary. To prove the reciprocal assume now that c is a path chain, and $[\psi \circ \phi(c)]$ is a boundary. Note once again, that all that ψ does is identify the points representing A_1 and A_2, leaving every other simplex in

$X/(A_1, A_2)$ intact. Furthermore if we let the points in A_1, A_2 be the first two points in the ordering of $(X/(A_1, A_2))^{(0)}$, we note that since A_1, A_2 are well disjoint, so they do not share an edge, and hence ψ preserves simplex orientations. With that in mind, since any $[b]$ such that $\partial[b] = \psi_* \circ \phi_*([c])$ is a 2-chain, we can calculate the preimage of $[b]$ by ψ_*, which is itself a well defined 2-chain $[b^*]$ satisfying:

$$\psi_*(\phi_*([c])) = \partial\psi_*([b^*]) = \psi_*(\partial[b^*])$$

And again since ψ_* identifies the 0-simplices corresponding to A_1 and A_2 it is an isomorphism on 1-chains, and therefore $\phi_*([c]) = \partial[b^*]$ meaning, $\phi_*([c])$ is a boundary. □

Acknowledgements

The authors would like to thank Patric Jensfelt for his help in the writing of this paper. This work was supported by the Knut and Alice Wallenberg Foundation and the Swedish Research Council.

References

[1] B. Majecka, "Statistical models of pedestrian behaviour in the forum," Master's thesis, School of Informatics, University of Edinburgh, 2009. MSc Dissertation.

[2] S. Kim, A. Bera, A. Best, R. Chabra, and D. Manocha, "Interactive and adaptive data-driven crowd simulation," in *Virtual Reality*, pp. 29–38, IEEE, Mar. 2016.

[3] D. Helbing and P. Molnár, "Social force model for pedestrian dynamics," *Physical Review E*, pp. 4282–4286, 1995.

[4] S. Kim, S. J. Guy, W. Liu, D. Wilkie, R. W. Lau, M. C. Lin, and D. Manocha, "Brvo: Predicting pedestrian trajectories using velocity-space reasoning," *The International Journal of Robotics Research*, vol. 34, no. 2, pp. 201–217, 2015.

[5] A. Bera, S. Kim, T. Randhavane, S. Pratapa, and D. Manocha, "Glmp - realtime pedestrian path prediction using global and local movement patterns," in *International Conference on Robotics and Automation*, IEEE, May 2016.

[6] J. Doellinger, M. Spies, and W. Burgard, "Predicting occupancy distributions of walking humans with convolutional neural networks," *Robotics and Automation Letters*, vol. 3, pp. 1522–1528, July 2018.

[7] A. Zyner, S. Worrall, and E. Nebot, "A recurrent neural network solution for predicting driver intentions at unsignalized intersections," *Robotics and Automation Letters*, vol. 3, July 2018.

[8] K. M. Rashid and A. H. Behzadan, "Enhancing motion trajectory prediction for site safety by incorporating attitude toward risk," in *Computing in Civil Engineering*, 2017.

[9] Z. Wang, P. Jensfelt, and J. Folkesson, "Multi-scale conditional transition map: Modeling spatial-temporal dynamics of human movements with local and long-term correlations," in *International Conference on Intelegent Robots and Systems*, pp. 6244–6251, IEEE/RSJ, Sept. 2015.

[10] F. T. Pokorny, K. Goldberg, and D. Kragic, "Topological trajectory clustering with relative persistent homology," in *International Conference on Robotics and Automation*, pp. 16–23, IEEE, May 2016.

[11] J. F. Carvalho, D. Kragic, M. Vejdemo-Johansson, and F. T. Pokorny, "Path clustering with homology area," in *International Conference on Robotics and Automation*, pp. 00–08, IEEE, May 2018.

[12] S. Xiao, Z. Wang, and J. Folkesson, "Unsupervised robot learning to predict person motion," in *International Conference on Robotics and Automation*, pp. 691–696, IEEE, May 2015.

[13] M. Bennewitz, W. Burgard, G. Cielniak, and S. Thrun, "Learning motion patterns of people for compliant robot motion," *The International Journal of Robotics Research*, vol. 24, no. 1, pp. 31–48, 2005.

[14] C. Sung, D. Feldman, and D. Rus, "Trajectory clustering for motion prediction," in *International Conference on Intelligent Robots and Systems*, pp. 1547–1552, Oct. 2012.

[15] M. Schneider and W. Ertel, "Robot learning by demonstration with local gaussian process regression," in *International Conference on Intelligent Robots and Systems*, pp. 255–260, IEEE/RSJ, Oct. 2010.

[16] S. Calinon, T. Alizadeh, and D. G. Caldwell, "On improving the extrapolation capability of task-parameterized movement models," in *International Conference on Intelligent Robots and Systems*, pp. 610–616, IEEE/RSJ, Nov. 2013.

[17] S. Aghabozorgi, A. Seyed Shirkhorshidi, and T. Ying Wah, "Time-series clustering — a decade review," *Information Systems*, vol. 53, Oct. 2015.

[18] E. W. Chambers and M. Vejdemo-Johansson, "Computing minimum area homologies," *Comput. Graph. Forum*, vol. 34, pp. 13–21, Sept. 2015.

[19] J. F. Carvalho, M. Vejdemo-Johansson, D. Kragic, and F. T. Pokorny, "An algorithm for calculating top-dimensional bounding chains," *PeerJ Computer Science*, vol. 4, p. e153, May 2018.

[20] A. Hatcher, *Algebraic Topology*. Cambridge University Press, 2002.

[21] S. Lang, *Fundamentals of Differential Geometry*. New York, NY: Springer US, 1999.

[22] S. Calinon, F. Guenter, and A. Billard, "On learning, representing and generalizing a task in a humanoid robot," *Transactions on Systems, Man and Cybernetics*, vol. 37, no. 2, pp. 286–298, 2007.

[23] M. Robinson, "Sheaves are the canonical data structure for sensor integration," *Information Fusion*, vol. 36, pp. 208–224, 2017.

Paper D

Free Space of Rigid Objects: Caging, Path Non-Existence, and Narrow Passage Detection

Anastasiia Varava*, J. Frederico Carvalho*,
Danica Kragic, Florian T. Pokorny

Abstract

In this work we propose algorithms to explicitly construct a conservative estimate of the configuration spaces of rigid objects in 2D and 3D. Our approach is able to detect compact path components and narrow passages in configuration space which are important for applications in robotic manipulation and path planning. Moreover, as we demonstrate, they are also applicable to identification of molecular cages in chemistry. Our algorithms are based on a decomposition of the resulting 3 and 6 dimensional configuration spaces into slices corresponding to a finite sample of fixed orientations in configuration space. We utilize dual diagrams of unions of balls and uniform grids of orientations to approximate the configuration space. We carry out experiments to evaluate the computational efficiency on a set of objects with different geometric features thus demonstrating that our approach is applicable to different object shapes. We investigate the performance of our algorithm by computing increasingly fine-grained approximations of the object's configuration space. A multithreaded implementation of our approach is shown to result in significant speed improvements.

1 Introduction

A basic question one may ask about a rigid object is to what extent it can be moved relative to other rigidly fixed objects in the environment. In robotic manipulation, this has lead to the notion of a *cage*. An object is considered caged by rigid

fixtures, or a robotic manipulator, if it cannot be moved arbitrarily far from its initial configuration. In terms of the configuration space of the rigid object, this is equivalent to being able to answer whether an object is located in a configuration contained in a bounded path component of its configuration space. Similarly, in fields such as chemistry and biology, this notion of a cage is a useful basic concept for predicting how molecules can restrict each other's mobility which has important applications to drug delivery and related problems [Mitra et al. (2013), Rother et al. (2016)].

The main challenge in caging verification is that the configuration space of a rigid object in 3D is in general a 6 dimensional subset of SE(3). For this reason, explicit reconstruction of the configuration space in terms of higher-dimensional analogues of discretization techniques such as voxel-grids and triangulations has been considered a computationally infeasible approach to this problem [Makita and Wan (2017)]. Past work has instead focused on analyzing caging configurations either for 2D objects only [McCarthy et al. (2012)], or for specific 3D objects with special geometric properties, such as handles and narrow parts [Pokorny et al. (2013), Varava et al. (2016)], which allow one to avoid modelling the configuration space directly.

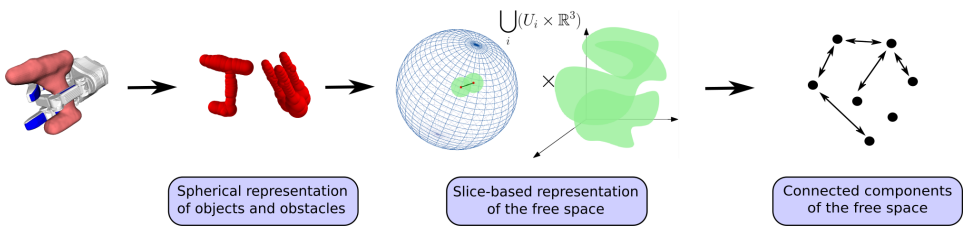


Figure D.1: Diagram of our method. We approximate the collision space of an object by choosing a finite set of fixed object's orientations and considering the corresponding slices of the collision space to $\mathbb{R}^n$ ($n \in \{2, 3\}$). From the collision space slices we compute approximations of the free space slices. Finally, we analyze the connectivity between neighboring slices to get an approximation of the connected components of the entire free space.

We study caging as a special case of proving path non-existence between a pair of configurations. To show that two configurations are disconnected, we construct an approximation of the object's collision space. Intuitively, we construct a set of slices of the object's collision space to subspaces corresponding to fixed orientations of the object, see Fig. D.1. We then compute the free space approximation as the complement to the collision space of each slice. By construction, our collision space approximation is a proper subset of the real collision space, which implies that when our algorithm reports that the two configurations are not path-connected, then there is no path between them in the real free space. However, for the same reason, our algorithm is not guaranteed to find all possible caging configurations, since we do not reconstruct the entire collision space.

The key contribution of the work we present here is to show that *it is possible to compute explicit approximations of configuration spaces of generic rigid bodies in 3D relatively efficiently,* while maintaining provability guarantees with respect to reasoning about caging and, more generally, path existence. Our technique for constructing such an approximation is based on the following key insights:

- **Representation:** We utilize a union-of-balls based object and obstacles representation that allows one to utilize dual diagrams to approximate the free space in a provably correct manner.
- **Slicing:** We use an approximate configuration space decomposition based on locally fixed orientations.
- **Object shrinking:** We use a subset of the object (its $\varepsilon-$core), which makes it possible to construct a sliced-based approximation of its configuration space.
- **Parallelization:** Our approach allows for parallel slice computation, leveraging modern CPU architectures.

The presented work constitutes an extended version of our initial work at WAFR'18 [Varava et al. (2018)]. It features an revised introduction and algorithms description, as well as extended experimental results, a parallelized version of the presented algorithm and its evaluation on 3D models of real objects.

2 Related Work

In manipulation, caging can be considered as an alternative to a force-closure grasp [Makita and Maeda (2008), Makita et al. (2013), Pokorny et al. (2013), Varava et al. (2016)], as well as an intermediate step on the way towards a form-closure grasp [Rodriguez et al. (2012)]. Unlike classical grasping, caging can be formulated as a purely geometric problem, and therefore one can derive sufficient conditions for an object to be caged. To prove that a rigid object is caged, it is enough to prove this for any subset (part) of the object. This allows one to consider caging a subset of the object instead of the whole object, and makes caging robust to noise and uncertainties arising from shape reconstruction and position detection.

The notion of a planar cage was initially introduced by [Kuperberg (1990)] as a set of n points lying in the complement of a polygon and preventing it from escaping arbitrarily far from its initial position. In robotics, it was subsequently studied in the context of point-based caging in 2D by [Rimon and Blake (1999), Pipattanasomporn and Sudsang (2006), Vahedi and van der Stappen (2008)], and others. A similar approach has also been adopted for caging 3D objects. For instance, [Pipattanasomporn and Sudsang (2011)] proposed an algorithm for computing all two-finger cages for non-convex polytopes. [Pereira et al. (2004)] and [Wang and Kumar (2002)] present a set of 2D caging-based algorithms enabling a group of mobile robots to cooperatively drag a trapped object to the desired goal.

In the above mentioned works fingertips are represented as points or spheres. Later, more complex shapes of caging tools were taken into account by [Pokorny et al. (2013), Stork et al. (2013), Varava et al. (2016), Makita and Maeda (2008), Makita et al. (2013)]. In these works, sufficient conditions for caging were derived for objects with particular shape features. [Makapunyo et al. (2013)] proposed a heuristic metric for partial caging based on the length and curvature of escape paths generated by a motion planner. The authors suggested that configurations that allow only rare escape motions may be used as cages in practice.

We address caging as a special case of the path non-existence problem: an object is caged if there is no path leading it to an unbounded path-connected component. The problem of proving path non-existence has been addressed by [Basch et al. (2001)] in the context of motion planning, motivated by the fact that most modern sampling-based planning algorithms do not guarantee that two configurations are disconnected, and rely on stopping heuristics in such situations [Latombe (1991)]. [Basch et al. (2001)] provide an algorithm to prove that two configurations are disconnected when the object is 'too big' or 'too long' to pass through a 'gate' between them. There are also some related results on approximating configuration spaces of 2D objects. [Zhang et al. (2008)] use approximate cell decomposition and prove path non-existence for 2D rigid objects. They decompose a configuration space into a set of cells and for each cell decide if it lies in the collision space. [McCarthy et al. (2012)] propose a related approach. There, they randomly sample the configuration space of a planar rigid object and reconstruct its approximation as an alpha complex. They later use it to check the connectivity between pairs of configurations. This approach has been later extended to planar energy-bounded caging by [Mahler et al. (2016)].

The problem of explicit construction (either exact or approximate) of configuration spaces has been studied for several decades in the context of motion planning, and a summary of early results can be found in the survey by [Wise and Bowyer (2000)]. [Lozano-Perez (1983)] introduced the idea of slicing along the rotational axis. To connect two consecutive slices, the authors proposed to use the area swept by the robot rotating between two consecutive orientation values. [Zhu and Latombe (1991)] extended this idea and used both outer and inner swept areas to construct a subset and a superset of the collision space of polygonal robots. The outer and inner swept areas are represented as generalized polygons defined as the union and intersection of all polygons representing robot's shape rotating in a certain interval of orientation values, respectively. Several recent works propose methods for exact computation of configuration spaces of planar objects [Behar and Lien (2013), Milenkovic et al. (2013)]. [Behar and Lien (2013)] proposed a method towards exact computation of the boundary of the collision space. [Milenkovic et al. (2013)] explicitly compute the free space for complete motion planning.

Thus, several approaches to representing configuration spaces of 2D objects, both exact and approximate, have been proposed and successfully implemented in the past. The problem is however more difficult if we consider a 3D object, as its configuration space is 6-dimensional. In the recent survey on caging by [Makita and Wan (2017)], the authors hypothesise that recovering a 6D configuration space and

understanding caged subspaces is computationally infeasible. To the best of our knowledge, our paper presents the first practical and provably-correct method to approximate a 6D configuration space.

Our approximation is computed by decomposing the configuration space into a finite set of lower dimensional slices. Although the idea of slicing is not novel and was introduced by [Lozano-Perez (1983)], recent advances in computational geometry and topology, as well as a significant increase in processing power, have made it possible to approximate a 6D configuration space on a common laptop. We identify two main challenges to slicing a 6D configuration space of a rigid object: how to quickly compute 3D slices of the free space, and how to efficiently discretize the orientation space. For slice approximation, our method relies on the fact that the collision space associated to a rigid object with a fixed orientation and an obstacle represented as a union of balls is itself a union of balls. Then we use the dual diagram to a union of balls presented by [Edelsbrunner (1999)] as an approximation of the free space of the slice. This way, we do not need to use generalized polygons, which makes previous approaches more difficult in 2D and very hard to generalize to 3D workspaces. To discretize $SO(3)$, we use the method by [Yershova et al. (2009)], which provides a uniform grid representation of the space. The confluence of these factors results in overcoming the dimensionality problem without losing necessary information about the topology of the configuration space, and achieving both practical feasibility and theoretical guarantees at the same time.

Finally, our method does not require precise information about the shape of objects and obstacles, and the only requirement is that balls must be located strictly inside them, which makes our approach more robust to noisy and incomplete sensor data.

We implemented our algorithm to approximate 3D and 6D configuration spaces and verify that it has in practice a reasonable runtime on a single core of Intel Core i7 processor. We provide a parallel implementation which makes use of modern parallel CPU architectures and investigate the effect of parallelization on the runtime of the algorithm.

3 Definitions and Notation

3.1 Objects and obstacles

For the sake of generality, in this paper we use the terms 'object' for objects and autonomous rigid robots (e.g., disc robots) moving in n-dimensional workspaces, where $n \in \{2, 3\}$. Similarly, we use the term 'obstacle' for everything that restricts mobility of the object – e.g., manipulators, walls, other rigidly fixed objects, etc.

When formally defining an object and a set of obstacles, we make a few mild assumptions to define a large class of shapes and include most of the regular objects in the real world. Since we want to represent both the object and the obstacles as a set of $n-$dimensional balls, we do not allow them to have 'thin parts'. Formally, we assume that they can be represented as *regular sets* [Rodriguez and Mason (2012)]:

Definition 9. *A set U is* regular *if it is equal to the closure of its interior:* $U = \mathrm{cl}(\mathrm{int}(U))$.

In the above definition, the interior of U is the largest open set contained in U, and the closure of $\mathrm{int}(U)$ is the smallest closed set containing $\mathrm{int}(U)$. In this paper, we assume that both the object and the obstacles are regular sets. Now, we define an object and a set of obstacles as follows:

Definition 10. *A* rigid object *is a regular compact connected non-empty subset of* $\mathbb{R}^n$. *A* set of obstacles *is a regular compact non-empty subset of* $\mathbb{R}^n$.

We approximate both the obstacles, $\mathcal{S}$ and the object, $\mathcal{O}$ as unions of balls which lie in their interior, that is, $\mathcal{S} = \{B_{R_1}(X_1), \ldots, B_{R_n}(X_n)\}$ with radii $R_1, \ldots, R_n$, and $\mathcal{O} = \{B_{r_1}(Y_1), \ldots, B_{r_m}(Y_m)\}$ with radii $r_1, \ldots, r_m$.

Let $\mathcal{C}(\mathcal{O}) = SE(n)$ denote the configuration space of the object. We define its *collision space*[1] $\mathcal{C}^{col}(\mathcal{O})$ as the set of the objects configurations in which the object penetrates the obstacles:

Definition 11. $\mathcal{C}^{col}(\mathcal{O}) = \{c \in \mathcal{C} \mid [\mathrm{int}\, c(\mathcal{O})] \cap [\mathrm{int}\, \mathcal{S}] \neq \emptyset\}$, *where $c(\mathcal{O})$ denotes the object in a configuration c. The* free space $\mathcal{C}^{free}(\mathcal{O})$ *is the complement of the collision space:* $\mathcal{C}^{free}(\mathcal{O}) = \mathcal{C}(\mathcal{O}) - \mathcal{C}^{col}(\mathcal{O})$.

Note that this definition allows the object to be in contact with the obstacles.

Definition 12. *Two configurations c_1 and c_2 are called* path-connected *if there exists a continuous collision-free path $\gamma : [0, 1] \to \mathcal{C}^{free}(\mathcal{O})$ between them: $\gamma(0) = c_1$, $\gamma(1) = c_2$. Two configurations are path-connected if and only if they belong to the same* path-connected component.

To compute path-connected components of the free space, we decompose the free space into a set of n-dimensional slices.

3.2 Slice-based representation of the C-space

In our previous work [Varava et al. (2017)], we suggested that configuration space decomposition may be a more computationally efficient alternative to its direct construction. We represent the configuration space of an object as a product $\mathcal{C}(\mathcal{O}) = \mathbb{R}^n \times SO(n)$, and consider a finite covering of $SO(n)$ by open sets (this is always possible, since $SO(n)$ is compact): $SO(n) = \bigcup_{i \in \{1,\ldots,s\}} U_i$. We recall the notion of a slice [Varava et al. (2017)]:

Definition 13. *A* slice *of the configuration space $\mathcal{C}(\mathcal{O})$ of a rigid object, is a subset of $\mathcal{C}(\mathcal{O})$ defined as follows: $Sl_U(\mathcal{O}) = \mathbb{R}^n \times U$, where U is an subset of $SO(n)$.*

[1] A theoretical analysis of different ways to define the free space can be found in [Rodriguez and Mason (2012)].

We denote a slice of the collision (free) space by $Sl_U^{col}(\mathcal{O})$ ($Sl_U^{free}(\mathcal{O})$, respectively). For each slice, we construct an approximation $aSl_U^{col}(\mathcal{O})$ of its collision space in such a way that our approximation lies inside the real collision space of the slice, $aSl_U^{col}(\mathcal{O}) \subset Sl_U^{col}(\mathcal{O})$.

This way, we approximate the entire collision space by a subset $a\mathcal{C}^{col}(\mathcal{O})$:

$$a\mathcal{C}^{col}(\mathcal{O}) = \left(\bigcup_{i \in \{1,\dots,s\}} aSl_{U_i}^{col}(\mathcal{O}) \right) \subset \mathcal{C}^{col}(\mathcal{O})$$

Now, we discuss how to construct slice approximations.

3.3 An $\varepsilon-$core of the object

First of all observe that by Def. 11, if a subset $\mathcal{O}'$ of an object $\mathcal{O}$ placed in configuration $c \in \mathcal{C}(\mathcal{O})$ is in collision, then the entire object $\mathcal{O}$ is in collision. Therefore, the collision space of $\mathcal{O}$ is completely contained within the collision space of $\mathcal{O}'$. This allows us to make the following observation:

Observation 1. *Consider an object $\mathcal{O}$ and a set of obstacles $\mathcal{S}$. Let $c_1, c_2 \in \mathcal{C}^{free}(\mathcal{O})$ be two collision-free configurations of the object. If there is no collision-free path between these configurations for its subset $\mathcal{O}' \subset \mathcal{O}$, then there is no collision-free path connecting these configurations for $\mathcal{O}$.*

Therefore, if some subset $\mathcal{O}'$ of $\mathcal{O}$ in configuration c is caged, then the entire object $\mathcal{O}$ in the same configuration c is caged. This means that if we construct aSl_U^{col} in such a way that for any configuration $c \in aSl_U^{col}$ there exists a subset $\mathcal{O}'$ of $c(\mathcal{O})$ such that $\mathcal{O}'$ is in collision, then $c(\mathcal{O})$ is also in collision. In our previous work ([Varava et al. (2017)]) we defined an $\varepsilon-$core of an object as follows:

Definition 14. *The ε-core of an object $\mathcal{O}$ is the set $\mathcal{O}_\varepsilon$ comprising the points of $\mathcal{O}$ which lie at a distance[2] of at least ε from the boundary of $\mathcal{O}$: $\mathcal{O}_\varepsilon = \{p \in \mathcal{O} \mid d(p, \partial\mathcal{O}) \geq \varepsilon\}$.*

Now, for an object $\mathcal{O}$ and its ε-core $\mathcal{O}_\varepsilon$, we write $\mathcal{O}^\phi$ and $\mathcal{O}_\varepsilon^\phi$ respectively to mean that their orientation is fixed at $\phi \in SO(n)$. So, let $\mathcal{C}^{col}(\mathcal{O}_\varepsilon^\phi)$ denote the collision space of $\mathcal{O}_\varepsilon$ with a fixed orientation ϕ. Note that since the orientation is fixed, we can identify $\mathcal{C}^{col}(\mathcal{O}_\varepsilon^\phi)$ with a subset of $\mathbb{R}^n$.

In [Varava et al. (2017)], we showed that for an object $\mathcal{O}$, $\varepsilon > 0$ and a fixed orientation $\phi \in SO(n)$ there exists a non-empty neighbourhood $U(\phi, \varepsilon)$ of ϕ such that for any $\theta \in U(\phi, \varepsilon)$, $\mathcal{O}_\varepsilon^\phi$ is contained in $\mathcal{O}^\theta$, see Fig. D.2.

In Sec. 5, we address the problem of representing and discretizing the space of orientations $SO(n)$, and show how it is related to the notion of $\varepsilon-$core.

[2] By distance here we mean Euclidean distance in $\mathbb{R}^n$

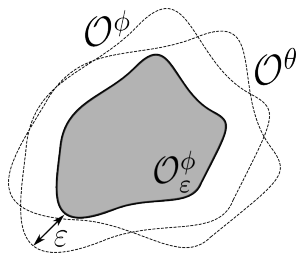


Figure D.2: An $\varepsilon-$core remains inside the object when we slightly rotate it

4 Existence of δ-clearance paths

For safety reasons, in path planning applications a path is often required to keep some minimum amount of clearance to obstacles. The notion of clearance of an escaping path can also be applied to caging: one can say that an object is partially caged if there exist escaping paths, but their clearance is small and therefore the object is unlikely to escape.

Definition 15. *We say that two configurations are δ-connected if and only if there exists a collision-free path of clearance at least ε connecting these configurations.*

Consider a superset of our object $\mathcal{O}$, defined as a set of points lying at most at distance δ from $\mathcal{O}$, and let us call it a *δ-offset* of the object: $\mathcal{O}_{+\delta} = \{p \in \mathbb{R}^n \mid d(p, \mathcal{O}) \leq \delta\}$. Equivalently to Def. 15, we can say that two configurations c_1 and c_2 are δ-connected in $\mathcal{C}^{free}(\mathcal{O})$ if and only if they are path-connected in $\mathcal{C}^{free}(\mathcal{O}_{+\delta})$.

Consider now the following modification of our algorithm. If we enlarge the $\varepsilon-$core by a $\delta-$offset, then our rotated object will not be guaranteed to contain it anymore, but the distance between any point of the enlarged core that is located outside of the rotated object and the object itself will not exceed δ. This means that if for this enlarged core our algorithm reports that two configurations are disconnected, then they either are disconnected in reality, or can be connected by a path of clearance at most δ. Let us denote the resulting space approximation by $a\mathcal{C}^{free}_{\varepsilon,\delta}(\mathcal{O})$.

One application of this observation is narrow passage detection. One can use our free space approximation to identify narrow passages as follows. If two configurations are disconnected in $a\mathcal{C}^{free}_{\varepsilon,\delta}(\mathcal{O})$, but connected in $a\mathcal{C}^{free}_{\varepsilon}(\mathcal{O})$, then they are connected by a narrow passage with clearance at most δ. Our approximation then can be used to localize this passage, so that probabilistic path planning algorithm can sample in this location.

Furthermore, we can view δ as the level of accuracy of the approximation: assume we want to use a coarse discretization of the orientation space, and therefore the distance between adjacent orientations is large. This will require a larger ε, in which

case some important information about the object's shape might not be captured by the $\varepsilon-$core. This might lead to a very conservative approximation of the free space. If now we consider $a\mathcal{C}^{free}_{\varepsilon,\delta}(\mathcal{O})$, we might get more caging configurations. These results will be considered with respect to the used parameter δ: if two configurations are disconnected in $a\mathcal{C}^{free}_{\varepsilon,\delta}(\mathcal{O})$, then the maximum possible clearance of a path connecting them in reality is at most δ.

This leads us to a tradeoff between the desired accuracy of our approximation expressed in terms of clearance δ, and the number of slices that we are willing to compute. The fewer slices we use, the larger δ we will need to consider.

5 Discretization of $SO(n)$

Elements of $SO(n)$ can be seen as parametrizing rotations in $\mathbb{R}^n$, and for any $q \in SO(n)$ we define R_q as the associated rotation. The notion of *displacement* of a point after applying a rotation helps us to understand how the size of $\varepsilon-$core is related to the discretization of $SO(n)$:

Definition 16. *Let* $D(\mathrm{R}_q)$ *denote the maximal* displacement *of any point* $p \in \mathcal{O}$ *after applying* R_q*, i.e.* $D(\mathrm{R}_q) = \max_{p\in\mathcal{O}}(d(p, \mathrm{R}_q(p)))$*, then* $\mathcal{O}_\varepsilon \subset \mathrm{R}_q(\mathcal{O})$ *if* $D(\mathrm{R}_q) < \varepsilon$*.*

Our goal now is to derive upper bounds for maximum displacement of any point in the object to make sure that the $\varepsilon-$core always remains inside the object when it is being rotated between different orientations belonging to the same slice, and how to efficiently discretize the orientation space.

5.1 Displacement in 2D

In our previous work ([Varava et al. (2017)]), we derived the following upper bound for the displacement of a two-dimensional object:

$$D(\mathrm{R}_q) \leq 2|\sin(q/2)| \cdot \mathrm{rad}(\mathcal{O}),$$

assuming that we rotate the object around its geometric center, $\mathrm{rad}(\mathcal{O})$ denotes the maximum distance from it to any point of the object, and q is the rotation angle.

In the two-dimensional case, discretization of the rotation space is simple: given a desired number of slices, we obtain the displacement $D(R_q)$ induced by rotation between two neighboring orientations, and compute a set of orientation samples $\{\phi_1 = 0, \phi_2 = 2q, \ldots, \phi_s = 2(s-1)q\}$, where $s = \lceil \pi/q \rceil$. Then, we choose the $\varepsilon > D(\mathrm{R}_q)$. This gives us a covering $\{U(\phi_1, \varepsilon), \ldots, U(\phi_s, \varepsilon)\}$ of $SO(2)$, where for each $i \in \{1, \ldots, s\}$ we define $U(\phi_i, \varepsilon) = [\phi_i - q, \phi_i + q]$.[3].

[3]This is a cover by closed sets, but given $q' > q$ satisfying $D(R_q) < \varepsilon$ we can use instead $U(\phi_i, \varepsilon) = (\phi_i - q', \phi_i + q')$ which results in the same graph.

5.2 Displacement in 3D

We now discuss the three-dimensional case. Similarly to the previous case, our goal is to cover $SO(3)$ with balls of fixed radius. To parametrize $SO(3)$ we use unit quaternions. For simplicity, we extend the real and imaginary part notation from complex numbers to quaternions, where $\Re q$ and $\Im q$ denote the real and "imaginary" parts of the quaternion q. Further, we identify $\Im q = q_i i + q_j j + q_k k$ with the vector $(q_i, q_j, q_k) \in \mathbb{R}^3$; and we write $\bar{q}$, and $|q|$ to mean the conjugate $\Re q - \Im q$ and the norm $\sqrt{q\bar{q}}$, respectively.

A unit quaternion q defines a rotation R_q, having an angle of $\theta_q = 2\cos^{-1}(\Re q)$ around axis $w_q = \frac{\Im q}{|\Im q|}$. This allows one to calculate the displacement of the rotation $D(q) = D(R_q)$ as:

$$\frac{D(q)}{\mathrm{rad}(\mathcal{O})} = 2\sin(\frac{\theta_q}{2}) = 2\sin(\cos^{-1}(\Re q)) = 2|\Im q|$$

We use the *angular distance* ([Yershova et al. (2009)]) to define the distance between two orientations:

$$\rho(x, y) = \arccos(|\langle x, y\rangle|),$$

where x and y are two elements of $SO(3)$ represented as unit quaternions which are regarded as vectors in $\mathbb{R}^4$, and $\langle x, y\rangle$ is their inner product. We define the angle distance from a point to a set $S \subseteq SO(3)$ in the usual way as

$$\rho(S, x) = \min_{y \in S} \rho(y, x)$$

[Yershova et al. (2009)] provide a deterministic sampling scheme to minimize the *dispersion*

$$\Delta(S) = \max_{x \in SO(3)} \rho(S, x).$$

Intuitively, the dispersion of a set $\Delta(S)$ determines how far a point can be from S, and in this way it determines how well the set S covers $SO(3)$. Now, assume we are given a set of samples $S \subseteq SO(3)$ such that $\Delta(S) < \Delta$. Then for any point $p \in SO(3)$ denote $D(p\bar{S}) = \max_{q \in S} D(p\bar{q})$, we want to show that in these conditions, there exists some small ε such that $\max_{p \in SO(3)} D(p\bar{S}) < \varepsilon$. This would imply that there exists some $\Delta' > \Delta$ such that if we take $U = \{q \in SO(3) \mid \rho(1, q) < \Delta'\}$, then denoting $U_p = \{qp \mid q \in U\}$, the family $\{U_p\}_{p \in S}$ fully covers $SO(3)$ and for any $q \in U_p$ satisfies $D(q\bar{p}) < \varepsilon$.

Now, Proposition 6 allows us to establish the relation between the distance between two quaternions and displacement associated to the rotation between them.

Proposition 6. *Given two unit quaternions p, q, the following equation holds:*

$$D(p\bar{q}) = 2\sin(\rho(p, q))\,\mathrm{rad}(\mathcal{O}). \tag{D.1}$$

The proof of Proposition 6 can be found in appendix.

This means that if we want to cover $SO(3)$ with patches U_i centered at a point p_i such that $D(p_i\bar{q})$ for any $q \in U_i$ is smaller than some ε, we can use any deterministic sampling scheme on $SO(3)$ (e.g. [Yershova et al. (2009)]) to obtain a set S with dispersion $\Delta(S) < \arcsin(\frac{\varepsilon}{2\,\mathrm{rad}(\mathcal{O})})$. Finally, by considering patches of the form $U(s,\varepsilon) = \{p \in SO(3) | \rho(s,p) < \Delta\}$, we obtain the corollary:

Corollary 4. *If $S \subseteq SO(3)$ has a dispersion $\Delta(S) < \arcsin(\frac{\varepsilon}{2\,\mathrm{rad}(\mathcal{O})})$ then the family of patches $\{U(s,\varepsilon) \mid s \in S\}$ forms a cover of $SO(3)$.*

Given such a cover of $SO(3)$, recall that we want to approximately reconstruct the full free space of the object $\mathcal{C}^{free}$ as the union of slices $aSl^{free}_{U(s,\varepsilon)}$. This requires us to test whether the orientation components of two slices overlap. To make this efficient, we create a *slice adjacency graph*, which is simply a graph with vertices S and edges (s,s') if $U(s,\varepsilon) \cap U(s',\varepsilon) \neq \emptyset$.

To compute the graph adjacency, we note that if two slices $U(s,\varepsilon), U(s',\varepsilon)$ overlap, there must exist some $p \in SO(3)$ such that $\rho(p,s), \rho(p,s') < \Delta$, which implies $\rho(s,s') < \rho(s,p) + \rho(s',p) < 2\Delta$. This is leveraged in Alg. D2.

Algorithm D2: ComputeAdjacentOrientations

input : S — a set of points in $SO(3)$.
Δ — dispersion of S.
output: G — a patch adjacency graph.

```
1 V ← S
2 T ← KDTree(V)
3 E ← {}
4 for p ∈ V do
5 |   P ← T.query(p, dist ≤ 2 sin(Δ)) \ {p}
6 |   P ← P ∪ T.query(−p, dist ≤ 2 sin(Δ))
7 |   for q ∈ L do
8 |   |   add (p, q) to E
9 return G = (V, E)
```

The algorithm starts by setting $V = S$, as this is the set of vertices in the graph, and we put these vertices in a $KDTree$ in order to quickly perform nearest neighbor queries. Now, to compute the set of edges E, we locate for each $p \in S$ the points at an angle distance smaller than 2Δ [4] in line 5. Finally, in line 6 we also add edges to the points at an angle distance smaller than 2Δ of $-p$, as both p and $-p$ represent the same orientation.

[4] Since the $KDTree$ T uses the Euclidean distance in $\mathbb{R}^4$ we employ the formula $\|p-q\| = 2\sin(\frac{\rho(p,q)}{2})$.

5.3 Different resolutions and dispersion estimation

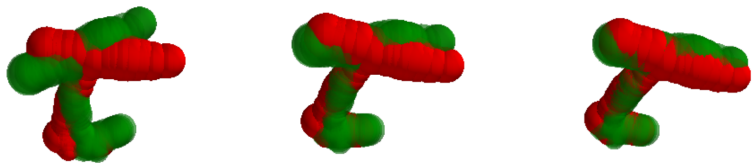


Figure D.3: In this figure we show an approximation of a drill as a collection of balls around its medial axis. This representation is visualized at two distinct orientations which are adjacent in the graph over $SO(3)$. The labels underneath each subfigure show how many vertices the corresponding graph has. As expected the larger the grid, the denser the sampling of $SO(3)$ and the closer the two objects are to each other. On one extreme we observe that the overlap between the object in the first grid is very limited and therefore the obtained connectivity graph should not be taken to be very informative. On the other extreme, in the third grid, we see that the overlap between adjacent orientations is almost complete.

Algorithm D3: Approximating the dispersion of a set of points $S \subset SO(3)$

input : S — Points in $SO(3)$.
output: Δ — Approximate dispersion.

1 $\Delta \leftarrow 0$
2 $T \leftarrow KDTree(V)$
3 **for** $i \in \{1, \ldots, N\}$ **do**
4 $\quad v \leftarrow \arg\max_{w \in SO(3)} \rho(S, w)$
5 $\quad w \leftarrow T.query(v)$
6 $\quad \Delta \leftarrow \max(\Delta, 2\sin^{-1}(\frac{\|w-v\|}{2}))$
7 **return** Δ

One of the prerequisites of Alg. D2 is the availability of an estimate of the dispersion of $\Delta(S)$. In our implementation we used the algorithm designed by [Yershova et al. (2009)] to compute S where the authors provide a provable upper bound for dispersion.

However, since this is an upper bound, and we want a tighter estimate of the dispersion to reduce the number of slices, in our implementation we employed a random sampling method to estimate the dispersion.

Alg. D3 summarizes the procedure by which we approximate the dispersion. On each iteration of the main loop (lines 3-6) it retrieves an element of $SO(3)$ which

is locally the furthest from S this is done by drawing a random sample followed by gradient ascent in line 4. In line 5 the nearest element to w of S is retrieved (since the distance to this element will be the distance from w to S), and finally in 6 we use the fact that $\|w - v\| = 2\sin(\frac{\rho(w,v)}{2})$ to update the current value of the dispersion.

5.4 The choice of ε and $SO(3)$ discretization in practice

Depending on the chosen discretization of the orientation space and its dispersion, the ε value can be chosen such that it is greater than the estimated displacement:

$$\varepsilon > 2\sin\left(\frac{\Delta(S)}{2}\right)\mathrm{rad}(\mathcal{O})$$

To discretize $SO(3)$, we compute 3 different grids corresponding to different resolution levels using the software provided by [Yershova et al. (2009)]. The first grid contains 576 vertices, and the respective dispersion estimate is 0.23. The second grid consists of 4608 vertices and its estimated dispersion is 0.10, and the third grid has 36864 vertices and a disperion estimate of 0.05. Fig. D.3 illustrates how much an object rotates between two neighboring orientations in each of these grids. In our experiments, we analyze how the performance of the algorithm is affected by the choice of a grid.

6 Free Space Approximation

Let us now discuss how we connect the slices to construct an approximation of the entire free space, see Alg. D4.

Let $\mathcal{G}(a\mathcal{C}_\varepsilon^{col}(\mathcal{O})) = (V, E)$ be a graph approximating the free space. The vertices of $\mathcal{G}$ correspond to the connected components $\{aC_1^i, \ldots, aC_{n_i}^i\}$ of each slice, $i \in \{1, \ldots, s\}$, and are denoted by $v = (aC, U)$, where aC and U are the corresponding component and orientation interval. Two vertices representing components $C_p \subset aSl_{U_i}^{free}$ and $C_q \subset aSl_{U_j}^{free}$, $i \neq j$, are connected by an edge if the object can directly move between them. For that, both the sets U_i, U_j, and aC_q, aC_p must overlap: $U_i \cap U_j, aC_q \cap aC_p \neq \emptyset$. $\mathcal{G}(a\mathcal{C}_\varepsilon^{col}(\mathcal{O}))$ approximates the free space of the object: if there is no path in $\mathcal{G}(a\mathcal{C}_\varepsilon^{col}(\mathcal{O}))$ between the vertices associated to the path components of two configurations c_1, c_2, then they are disconnected in $\mathcal{C}^{free}(\mathcal{O})$.

We start by choosing one orientation and run a breadth-first search over the orientation grid $\mathcal{Q}$. When reaching a particular orientation, we construct the slice corresponding to it. In line 14, we compute the free space of a slice as explained in Alg. D5. In line 17, we check which connected components of adjacent slices overlap, and add edges between them, see Alg. D6.

In order to query connectivity, the slices approximations should be preserved. In our current implementation, each slice is deleted as soon as the connectivity check between it and the slices adjacent to it is performed, in order to optimize memory

Algorithm D4: ComputeConnectivityGraph

input : O_{obst}, O_{obj} — spherical representations of the obstacles and the object.
δ — clearance parameter
$\mathcal{Q}$ — a grid over $SO(3)$.
output: $\mathcal{G}$ — a connectivity graph of the free space.

$O_\varepsilon \leftarrow ComputeEpsilonCore(O_{obj}, \delta, \mathcal{Q})$
$MarkedOrientations \leftarrow \emptyset$
$Queue \leftarrow \emptyset$
$q \leftarrow \mathcal{Q}[0]$
$MarkedOrientations.add(q)$
$Slices[q] \leftarrow ComputeSlice(O_{obst}, O_\varepsilon, q)$
$Queue.add(q)$
while $Queue \neq \emptyset$ **do**
  $q_{cur} \leftarrow Queue.dequeue()$
  $CurrentSlice \leftarrow Slices[q_{cur}]$
  for $q_{adj} \in \mathcal{Q}.adjacentOrientations(q_{cur})$ **do**
    if $q_{cur} \notin MarkedOrientations$ **then**
      if $Slices[q_{adj}] = \emptyset$ **then**
        $Slices[q_{adj}] \leftarrow ComputeSlice(O_{obst}, O_{obj}, q)$
        $Queue.enqueue(q_{cur})$
        $MarkedOrientations.add(q_{cur})$
      $AddEdges(Slices[q_{cur}], Slices[q_{adj}])$
  $Slices[q_{cur}].reset()$
return $\mathcal{G}$

usage. In this case, one can save the slice approximation together with the resulting graph to disk, for later use by a querying algorithm.

6.1 Construction of Slices

Now, let us discuss how we approximate path-connected components of the free space of each slice, see Alg. D5. Given a set of obstacles, an object, and a particular orientation of its $\varepsilon-$core, we start by computing the collision space of the slice.

In [Varava et al. (2017)], we derive the following representation for the collision space of $\mathcal{O}_\varepsilon^\phi$:

$$\mathcal{C}^{col}(\mathcal{O}_\varepsilon^\phi) = \bigcup_{i,j} (B_{R_j + r_i - \varepsilon}(X_j - \overline{GY_i})),$$

where G is the origin chosen as the geometric center of the object, and $\overline{GY_i}$ denotes the vector from G to Y_i.

Indeed, the object represented as a union of balls collides with the obstacles if at least one of the balls is in collision, so the collision space of the object is a union of the collision spaces of the balls shifted with respect to the position of the balls centers:

$$\mathcal{C}^{col}(\mathcal{O}_\varepsilon^\phi) = \bigcup_{i\in\{1\ldots m\}} \mathcal{C}^{col}(B_{r_i-\varepsilon}(Y_i)) - \overline{GY_i}$$

Now, each ball collides with the obstacles when the distance from the obstacles to its center is not greater than the radius of the ball, so the collision space of a single ball of radius $r_i - \varepsilon$ can be written as:

$$\{x \in \mathbb{R}^d | \, \mathrm{d}(x, S) \le r_i - \varepsilon\} = \bigcup_{j\in\{1..n\}} B_{R_j+r_i-\varepsilon}(X_j)$$

Algorithm D5: ComputeSlice

input : O_{obst}, O_ε — spherical representations of the obstacles and the $\varepsilon-$core.
q — orientation.
output: aSl^{free} — A set of connected components of the slice corresponding to q

```
1  C^col(O_ε) ← Collision-Space(O_obst, O_obj, q)
2  V(C^col(O_ε)) ← Weighted-Voronoi-Diagram(C^col(O_ε))
3  aC^free ← Dual-Diagram(V(C^col(O_ε)))
4  aC_0 ← Compute-Infinite-Component()
5  i ← 0
6  foreach ball ∈ aC^free do
7      if Connected-Component(ball) = ∅ then
8          i ← i + 1
9          aC_i ← Compute-Component(ball)
10 return aSl^free = {aC_0, ..., aC_n}
```

At the next step, we approximate the free space of the slice. For this, we approximate the complement of the collision space by constructing the *dual diagram* ([Edelsbrunner (1999)]) to the set of balls representing the collision space.

A dual diagram $Dual(\bigcup B_i)$ of a union of balls $\bigcup B_i$ is a finite collection of balls such that the complement of its interior is contained in and is homotopy equivalent to $\bigcup B_i$. It is convenient to approximate $\mathcal{C}^{free}(\mathcal{O}_\varepsilon^\phi)$ as a dual diagram of collision space balls for several reasons. First, balls are simple geometric primitives which make intersection checks trivial. Second, the complement of the dual diagram is guaranteed to lie strictly inside the collision space, which provides us with a conservative approximation of the free space. Finally, homotopy equivalence between its complement and the approximate collision space implies that our approximation

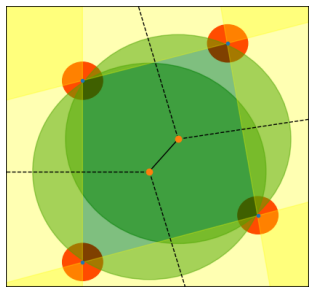


Figure D.4: Green circles and yellow halfspaces ("infinite circles") together represent a dual diagram of the red circles. Green circles are centered at the vertices of the Voronoi diagram; yellow halfspaces correspond to infinite edges of the diagram (dashed lines).

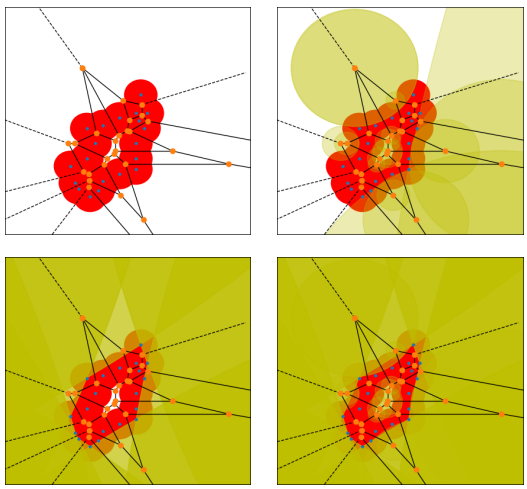


Figure D.5: From top-left to bottom-right, the elements of the dual diagram approximating the free space. We start with building a Voronoi diagram (visualized in black); the second figure shows a set of finite orthogonal balls, corresponding to the regular edges of the diagram; the third figure visualizes the "infinite" orthogonal balls corresponding to the infinite edges of the diagram; finally, the last figure depicts full diagram. Red circles with blue centers represent the collision space, the yellow circles approximate the dual diagram.

preserves the connectivity of the free space that we want to approximate. Another advantage of a dual diagram is that it is easy to construct.

A weighted Voronoi diagram is a special case of a Voronoi diagram, where instead of Euclidean distance between points, a special distance function is used.

In our case, the weighted distance of a point $x \in \mathbb{R}^n$ from $B_{r_i}(z_i)$ is equal to $d_w(x, B_{r_i}(z_i)) = ||x - z_i|| - r_i^2$.

To construct a dual diagram of a union of balls $\bigcup_i B_{r_i}(z_i)$, we first construct their weighted Voronoi diagram, see Fig. D.4. For each vertex y_j of the weighted Voronoi diagram, let $B_{q_j}(y_j)$ whose radius q_j is equal to the square root of the weighted distance of y_j to any of the four (three in the two-dimensional case) balls from $\bigcup_i B_{r_i}(z_i)$ generating y_j. Then, take each infinite edge of the Voronoi diagram, and add an degenerate "infinitely large" ball (a half-space) with center at the infinite end of this edge. The entire process of dual diagram construction can be seen on Fig. D.5.

After constructing the dual diagram, we find pairs of overlapping balls in it and run depth-first search to identify connected components. It is important to note that each dual diagram always has exactly one unbounded connected component, representing the outside world.

Finally, let us discuss how to understand whether free space approximations of neighboring slices overlap. In Alg. D6, to check whether two connected components in adjacent slices intersect (line 7), we recall that they are just finite unions of balls. Instead of computing all pairwise intersections of balls, we approximate each ball by its bounding box and then use the CGAL implementation of Zomorodian's algorithm ([Zomorodian and Edelsbrunner (2000)]), which efficiently computes the intersection of two sequences of three-dimensional boxes. Every time it finds an overlapping pair of boxes, we check whether the respective balls also intersect.

Algorithm D6: AddEdges

input : $aSl_{U_i}^{free}, aSl_{U_j}^{free}$ — free space approximations of two adjacent slices.

output: E_{ij} — a set of edges between the connected components of $aSl_{U_i}^{free}$ and $aSl_{U_j}^{free}$.

```
1 E_{i,j} ← ∅
2 for aC_i ∈ ConnectedComponents(aSl_{U_i}^{free}) do
3     for aC_j ∈ ConnectedComponents(aSl_{U_j}^{free}) do
4         if aC_i.isInfinite() AND aC_j.isInfinite() then
5             E_{i,j}.addEdge(v(aC_i, U_i), v(aC_j, U_i))
6         else
7             if DoIntersect(aC_i, aC_j) then
8                 E_{i,j}.addEdge(v(aC_i, U_i), v(aC_j, U_i))
9 return E_{i,j}
```

Remark 1. *Recall that in each slice the $a\mathcal{C}^{free}(\mathcal{O}_\varepsilon^{\phi_i})$ are constructed as the dual of the collision space $\mathcal{C}^{col}(\mathcal{O}_\varepsilon^{\phi_i})$, which entails that $a\mathcal{C}^{free}(\mathcal{O}_\varepsilon^{\phi_i})$ has the same con-*

nectivity as $\mathcal{C}^{free}(\mathcal{O}_\varepsilon^{\phi_i})$. However, it also entails that any connected component of $a\mathcal{C}^{free}(\mathcal{O}_\varepsilon^{\phi_i})$ partially overlaps with the collision space $\mathcal{C}^{col}(\mathcal{O}_\varepsilon^{\phi_i})$. This means that for two connected components $C_j^i, C_{j'}^{i'}$ of adjacent slices which do not overlap, it may occur that the corresponding approximations $aC_j^i, aC_{j'}^{i'}$ do overlap. In this case the resulting graph $\mathcal{G}(a\mathcal{C}_\varepsilon^{col}(\mathcal{O}))$ would contain an edge between the corresponding vetices. This effect can be mitigated by verifying whether the overlap between the approximations occurs within the collision space of both slices. This can be done for example by covering the intersection $aC_j^i \cap aC_{j'}^{i'}$ with a union of balls and checking if it is contained inside the collision space $\mathcal{C}^{col}(\mathcal{O}_\varepsilon^{\phi_i}) \cup \mathcal{C}^{col}(\mathcal{O}_\varepsilon^{\phi_{i'}})$.

7 Parallel implementation

To increase the performance of our algorithm, we designed and implemented a parallelized version that uses multiple threads to compute different slices simultaniously. Alg. D7 describes the process.

Recall that in the single-thread version of the algorithm (see Alg. D4), we performed breadth-first search (BFS) over the orientation grid $\mathcal{Q}$. For each orientation, we computed the corresponding slice of the free space and its overlap with neighboring slices. The slice was deleted when all its neighbours were constructed and the connectivity check between them was performed.

A naive approach to parallelization would have a main thread call child threads on to process each new orientation in the queue, however since the orientations are enqueued in sequence it would increase the chances that different threads try to compete for computing the same slice data, necessitating threads to wait for each other to finish. Instead we opted for a simple scheme where each thread performs its own BFS from a different point in the tree taking care to lock access to resources every time shared data is accessed.

Alg. D7 describes the procedure followed by each thread. It assumes that all threads share an access to the array of slices and the connectivity graph, as well as the book-keeping data, i.e. an array of mutexes, and an array containing the status of each slice. The status array ($\mathcal{S}$) is used to decide what (if anything) should be computed about a given slice at a given moment, whereas the mutex array ($\mathcal{M}$) is used so two threads don't try to alter the same slice at the same time.

The main loop (lines 4-26) proceeds as follows: it pops the first element of the queue q_{cur} and locks its corresponding mutex. If it cannot obtain ownership of the mutex, it waits until this is freed by the other thread. It then verifies the status of the current orientation (lines 7 and 10), if the slice associated to the orientation has yet to be computed it does so (line 8) and if its edges have not yet been processed it goes on to process the edges to its neighboring slices (lines 12-25). To compute the edges between the slices corresponding to the current orientation q_{cur} and an adjacent orientation q_{adj} the thread needs to also lock the mutex associated to q_{adj} (line 13) if it does so successfully then it proceeds to check the connection between q_{cur} and q_{adj}, otherwise it checks the next adjacent orientation. When it has tried

Algorithm D7: ConnectivityGraphThread

input : O_{obst}, O_{obj} — spherical representations of the obstacles and the object.
δ — clearance parameter.
$\mathcal{Q}$ — a graph over $SO(3)$.
i — an initial orientation.

shared: $\mathcal{M}$ — an array containing a mutex per slice.
$Slices$ — an array of slices (one per orientation).
$\mathcal{S}$ — an integer array indicating slice status (0:unseen, 1:seen, 2:fully processed).
$\mathcal{G}$ — a connectivity graph of the free space

```
1  O_ε ← ComputeEpsilonCore(O_obj, δ, Q)
2  Queue ← ∅
3  Queue.enqueue(i)
4  while Queue ≠ ∅ do
5      q_cur ← Queue.dequeue()
6      Lock(M[q_cur])
7      if S[q_cur] = 0 then
8          Slices[q_cur] ← ComputeSlice(O_obst, O_ε, q_cur)
9          S[q_cur] ← 1
10     if S[q_cur] = 1 then
11         n ← Size(Q.adjacentOrientations(q_cur))
12         for q_adj ∈ Q.adjacentOrientations(q_cur) do
13             lock ← TryLock(M[q_adj])
14             if lock.successful() then
15                 if S[q_adj] = 0 then
16                     Slices[q_adj] ← ComputeSlice(O_obst, O_ε, q_adj)
17                     S[q_adj] ← 1
18                     Queue.enqueue(q_adj)
19                 if S[q_adj] = 1 then
20                     AddEdges(Slices[q_cur], Slices[q_adj])
21                 n ← n − 1
22                 Unlock(M[q_adj])
23         if n = 0 then
24             S[q_cur] ← 2
25             Slices[q_cur].reset()
26     Unlock(M[q_cur])
27 NotifyParent()
```

to calculate the edges to all the orientations adjacent to q_{cur} it verifies if all of them were successfully calculated (line 23), if so, the current orientation is marked as fully processed and the slice is deleted. Note also that the edges between $Slice[q_{cur}]$ and $Slice[q_{adj}]$ only get computed in line 20 if $Slice[q_{adj}]$ has not been fully processed itself, as that would mean that the existing edges would have been processed in a previous step.

This procedure guarantees the absence of data corruption since in order to change the data associated to slices or the edges between them, the corresponding mutexes need to be locked first. Furthermore the algorithm is guaranteed to be lock-free given that:

- If two threads try to lock the same q_{cur}, the second one has to wait for the first one to finish before it can proceed.
- If there is an edge (u, v) of the orientation graph and one thread has $q_{cur} = v$ and a second thread has $q_{cur} = u$, then given that both instances only *try* to lock the mutex of the corresponding to the other thread's orientation, they will not cause a deadlock.

Note also that each thread is guaranteed to terminate since orientations only get enqueued if their status is unseen ($\mathcal{S}[q_{adj}] = 0$ in line 15) and this status is revoked immediately afterwards (line 17) before it is enqueued. Furthermore since this occurs while the mutex associated to the corresponding slice is locked, no other thread is able to enqueue the same orientation. This guarantees that each orientation is only enqueued once.

Alg. D7 is called by a threadpool as shown in Alg. D8. The algorithm ComputeConnectivityGraphParallel works by setting up the data that must be shared between the several instances of ConnectivityGraphThread, and calling it starting from different orientations in $\mathcal{Q}$. Once all slices have been constructed the algorithm proceeds by checking that they have all been completely processed, and otherwise checks the intersections with their remaining neighbors. This step is required in case the situation arises where two threads are processing neighboring orientations simultaneously. In this case they may fail to lock the other's mutex and therefore ignore the edges that may exist between their corresponding vertices. However since this failure implies that neither orientation is marked as fully processed, the corresponding slices are not freed in line 25 of Alg. D7, and are therefore available to be further processed in line 16 of Alg. D8.

The function *SelectQuaternion* is used to select the next quaternion which has not been seen before. In our implementation we use a simple scheme by drawing a random number r and choosing the first $i > r$ so that $\mathcal{S}[i] = 0$.

8 Theoretical Properties of Our Approach

In this section, we discuss correctness, completeness and computational complexity of our approach.

Algorithm D8: ComputeConnectivityGraphParallel

input : O_{obst}, O_{obj} — spherical representations of the obstacles and the object
δ — clearance parameter.
$\mathcal{Q}$ — a graph over $SO(3)$
output: $\mathcal{G}$ — a connectivity graph of the free space

shared $\mathcal{G} \leftarrow (\emptyset, \emptyset)$
shared $\mathcal{M} \leftarrow mutexArray[|\mathcal{Q}|]$
shared $Slices \leftarrow sliceArray[|\mathcal{Q}|]$
shared $\mathcal{S} \leftarrow intArray[|\mathcal{Q}|]$
while $\exists i : \mathcal{S}[i] = 0$ **do**
    $q \leftarrow SelectQuaternion(\mathcal{S})$
    $LaunchThread\ ConnectivityGraphThread(O_{obst}, O_{obj}, \mathcal{Q}, \delta, q)$
    $NThreads \leftarrow NThreads - 1$
    if $NThreads = 0$ **then**
        $WaitForAnyChild()$
$WaitForAllChildren()$
for $q_{cur} \leftarrow \mathcal{Q}$ **do**
    if $\mathcal{S}[q_{cur}] = 1$ **then**
        for $q_{adj} \in \mathcal{Q}.adjacentOrientations(q_{cur})$ **do**
            if $\mathcal{S}[q_{adj}] = 1$ **then**
                $AddEdges(Slices[q_{cur}], Slices[q_{adj}])$
        $Slices[q_{cur}].reset()$
return $\mathcal{G}$

8.1 Correctness

First let us show that our algorithm is correct: i.e., if there is no collision-free path between two configurations in our approximation of the free space, then these configurations are also disconnected in the actual free space.

Proposition 7 (correctness)**.** *Consider an object $\mathcal{O}$ and a set of obstacles $\mathcal{S}$. Consider two collision-free configurations of the object. If they are not path-connected in $\mathcal{G}(a\mathcal{C}_\varepsilon^{free}(\mathcal{O}))$, then they are not path-connected in $\mathcal{C}^{free}(\mathcal{O})$.*

Proof. Recall that the approximation of the free space is constructed as follows:

$$a\mathcal{C}_\varepsilon^{free}(\mathcal{O}) = \bigcup_{i=1}^{s} aSl_{U(\phi_i,\varepsilon)}^{free},$$

where

$$aSl_{U(\phi_i,\varepsilon)}^{free} = Dual(\mathcal{C}^{col}(\mathcal{O}_\varepsilon^{\phi_i})) \times U(\phi_i, \varepsilon) \tag{D.2}$$

Now, recall that by definition $(Dual(\mathcal{C}^{col}(\mathcal{O}^{\phi_i}_{\varepsilon})))^c \subset \mathcal{C}^{col}(\mathcal{O}^{\phi_i}_{\varepsilon})$ ([Edelsbrunner (1999)]), and that we choose ε and $U(\phi_i, \varepsilon)$ so that for any $\phi \in U(\phi_i, \varepsilon)$, $\mathcal{C}^{col}(\mathcal{O}^{\phi_i}_{\varepsilon}) \subset \mathcal{C}^{col}(\mathcal{O}^{\phi})$. This implies that $(Dual(\mathcal{C}^{col}(\mathcal{O}^{\phi_i}_{\varepsilon})))^c \subseteq \mathcal{C}^{col}(\mathcal{O}^{\phi})$ for any $\phi \in U(\phi_i, \varepsilon)$, and conversely that $\mathcal{C}^{free}(\mathcal{O}^{\phi}) \subset Dual(\mathcal{C}^{col}(\mathcal{O}^{\phi_i}_{\varepsilon}))$. Finally, since $Sl^{free}_{U(\phi_i,\varepsilon)} = \bigcup_{\phi} \mathcal{C}^{free}(\mathcal{O}^{\phi}) \times \{\phi\}$, we have:

$$Sl^{free}_{U(\phi_i,\varepsilon)} \subseteq aSl^{free}_{U(\phi_i,\varepsilon)}, \tag{D.3}$$

We now want to show that if there is no path between two vertices $v = (aC, U)$ and $v' = (aC', U')$ in $\mathcal{G}(a\mathcal{C}^{free}_{\varepsilon})$, then there is no path between connected components of $a\mathcal{C}^{free}_{\varepsilon}(\mathcal{O})$ corresponding to them. It is enough to show that if two vertices corresponding to adjacent slices are not connected by an edge, then they represent two components which are disconnected in $Sl^{free}_{U} \cup Sl^{free}_{U'}$.

Consider two adjacent slices $Sl_{U(\phi_i,\varepsilon)}$ and $Sl_{U(\phi_j,\varepsilon)}$, and two path-connected components $C_1 \subset aSl^{free}_{U(\phi_i,\varepsilon)}$ and $C_2 \subset aSl^{free}_{U(\phi_j,\varepsilon)}$. Let aC_1 and aC_2 be their respective representations as unions of balls.

Let v_1 and v_2 be the vertices of $\mathcal{G}(a\mathcal{C}^{free}_{\varepsilon}(\mathcal{O}))$ corresponding to these components: $v_1 = (aC_1, U(\phi_i, \varepsilon))$ and $v_2 = (aC_2, U(\phi_j, \varepsilon))$. By construction, these are adjacent slices, therefore $U(\phi_i, \varepsilon) \cap U(\phi_j, \varepsilon) \neq \emptyset$ and since there is no edge between v_1 and v_2, we get $aC_1 \cap aC_2 = \emptyset$. But, by construction $C_1 \subseteq aC_1 \times U(\phi_i, \varepsilon)$ and $C_2 \subseteq aC_2 \times U(\phi_j, \varepsilon)$, therefore, we get:

$$C_1 \cap C_2 \subseteq (aC_1 \times U(\phi_i, \varepsilon)) \cap (aC_2 \times U(\phi_j, \varepsilon)) = \emptyset$$

And so C_1 and C_2 are disjoint in the union of the corresponding slices, which concludes the proof. □

8.2 $\delta-$Completeness

Now, we show that if two configurations are not path-connected in $\mathcal{C}^{free}(\mathcal{O})$, we can construct an approximation of $\mathcal{C}^{free}(\mathcal{O})$ in which these configurations are either disconnected or connected by a narrow passage.

Proposition 8 (δ-completeness)**.** *Let c_1, c_2 be two configurations in $\mathcal{C}^{free}(\mathcal{O})$. If they are not path-connected in $\mathcal{C}^{free}(\mathcal{O})$, then for any $\delta > 0$ there exists $\varepsilon > 0$ such that the corresponding configurations are not path-connected in $\mathcal{G}(a\mathcal{C}^{free}_{\varepsilon}(\mathcal{O}_{+\delta}))$, where the graph is produced according to the procedure outlined in Rem. 1.*

In proving this proposition we make used of the notion of the signed distance between two sets:

$$\bar{\mathrm{d}}_{\mathrm{s}}(\mathcal{O}, \mathcal{S}) = \begin{cases} \min_{p \in O} d(p, \mathcal{S}) & \text{if } \mathcal{O} \cap \mathcal{S} \neq \emptyset \\ -\max_{p \in \mathcal{O} \cap \mathcal{S}} d(p, \partial\mathcal{S}) & \text{otherwise} \end{cases}$$

Where $\partial\mathcal{S}$ denotes the boundary of $\mathcal{S}$. Note that $\bar{\mathrm{d}}_{\mathrm{s}}(A, B)$ is not necessarily the same as $\bar{\mathrm{d}}_{\mathrm{s}}(B, A)$ so we instead consider $\mathrm{d}_{\mathrm{s}}(A, B) = \min(\bar{\mathrm{d}}_{\mathrm{s}}(A, B), \bar{\mathrm{d}}_{\mathrm{s}}(B, A))$.

Proof. Recall Rem. 1 that there is an edge between vertices (aC_1, ϕ_1), (aC_2, ϕ_2) only if $U(\phi_1, \varepsilon)$ overlaps with $U(\phi_2, \varepsilon)$ and C_1 overlaps with C_2 where $C_i = aC_i \cap \mathcal{C}^{free}(\mathcal{O}_\varepsilon^{\phi_i})$ for $i = 1, 2$ i.e. the components of the actual free space of $\mathcal{O}_\varepsilon^{\phi_i}$ $(i = 1, 2)$ corresponding to the approximations aC_1 and aC_2.

Recall now that we want to prove that for a pair of configurations c_1, c_2 which are not path-connected in $\mathcal{C}^{free}(\mathcal{O})$, then for any $\delta > 0$ there exists some $\varepsilon > 0$ so that they are not path-connected in $\mathcal{G}(a\mathcal{C}_\varepsilon^{free}(\mathcal{O}_{+\delta}))$ for. Therefore, we start by noting that since c_1 and c_2 are not path-connected there exists a collision configuration c in any path between them, which implies $\mathrm{d_s}(c(\mathcal{O}), \mathcal{S}) < 0$. Thus, for the same configuration c we have $\mathrm{d_s}(c(\mathcal{O}_{+\delta}), \mathcal{S}) < -\delta$.

To see that this will result in path non-existence, we take an arbitrary $\varepsilon > 0$ and consider the collision space $a\mathcal{C}_\varepsilon^{col}(\mathcal{O}_{+\delta})$. Further, we let $c = (p, \phi) \in \mathbb{R}^3 \times SO(3)$, and for any ϕ_i such that $\phi \in U(\phi_i, \varepsilon)$, we define $c_i = (p, \phi_i)$. Finally, we restrict ourselves to the case where $\varepsilon < \delta$ and define $\delta' = \delta - \varepsilon$, then we get:

$$\begin{aligned}\mathrm{d_s}(c_i(\mathcal{O}_{+\delta'}), \mathcal{S}) &\leq d(c(\mathcal{O}_{+\delta'}), c_i(\mathcal{O}_{+\delta'})) + \mathrm{d_s}(c(\mathcal{O}_{+\delta'}), \mathcal{S}) \\ &\leq \varepsilon \left(1 + \frac{\delta'}{\mathrm{rad}(\mathcal{O})}\right) - \delta'\end{aligned}$$

Where the term $\varepsilon \left(1 + \frac{\delta'}{\mathrm{rad}(\mathcal{O})}\right)$ corresponds to the maximum displacement of the object offset $\mathcal{O}_{+\delta}$ associated to a rotation of $\mathcal{O}$ with maximum displacement ε. Which implies that, as long as we choose ε such that

$$\varepsilon \left(1 + \frac{\delta - \varepsilon}{\mathrm{rad}(\mathcal{O})}\right) - (\delta - \varepsilon) < 0$$

then c is in collision and the proof will be concluded. To see that this inequality has a solution with $0 < \varepsilon < \delta$ we simply note that with $\varepsilon = 0$ it is reduced to $-\delta < 0$. And the roots of the equation in terms of δ are given by:

$$\varepsilon = \frac{\mathrm{rad}(\mathcal{O})}{2} \left(2 + \frac{\delta}{\mathrm{rad}(\mathcal{O})} \pm \sqrt{\left(2 + \frac{\delta}{\mathrm{rad}(\mathcal{O})}\right)^2 - \frac{4\delta}{\mathrm{rad}(\mathcal{O})}}\right)$$

which are both positive. Finally call the smallest of the two roots $\tilde{\varepsilon}$ and the inequality holds as long as $\varepsilon \in (0, \min\{\tilde{\varepsilon}, \delta)\}$, concluding the proof. □

8.3 Computational complexity

Let us now discuss the total computational complexity of the algorithm. Let s be the number of slices, and let n and m be the number of balls in the object's and the obstacle's spherical representation, respectively. We have pre-computed the grids over $SO(3)$ corresponding to different dispersion values, and therefore we

are only interested in the complexity of the connectivity graph construction. For each slice, we execute two computationally expensive procedures: we compute a weighted Voronoi diagram of the collision space, which allows us to extract the balls representation of the free space, and then for each slice we compute its intersections with adjacent slices. In practice, each orientation in $\mathcal{Q}$ has around 20 adjacent orientation values, so each slice has around 20 neighbours[5].

In CGAL representation, the regular triangulation contains the corresponding weighted Voronoi diagram. Note that a weighted Voronoi diagram can be constructed by other means using for example the algorithm from ([Aurenhammer et al. (1984)]). The complexity of this step is $O(n^2m^2)$. The computation of the connected components of each slice is linear on the number of balls in the dual diagram, which makes the overall complexity of this step $O(n^2m^2)$.

The complexity of finding the intersections between two connected components belonging to different slices $O(b \log^3(b) + k)$ in the worst-case [Zomorodian and Edelsbrunner (2000)], where b is the number of balls in both connected components, and k the output complexity, i.e., the number of pairwise intersections of the balls.

The complexity of the final stage of the algorithm — computing connected components of the connectivity graph — is linear on the number of vertices, and can be expressed as $O(s\ c)$, where c is the average number of connected components per slice (a small number in practice).

9 Examples and Experiments

In this section, we test our algorithm on 2D and 3D objects. In the first experiment, we investigate how the quality of the spherical approximation of the workspace affects the output of our algorithm. In the next experiment, we test our algorithm on a set of 3D objects representing different classes of geometric shapes. In our final experiment we investigate how the performance of our parallel implementation changes with respect to different numbers of threads.

9.1 Object and obstacle approximation

A key requirement of our method is the approximation of both objects and obstacles as unions of balls. This presents a challenge especially when dealing with 3D objects. For 2D experiments we were able to obtain good results using a simple grid for the centers of the balls, and choosing the radius of the balls at various levels. This resulted in the approximations of the obstacle seen in Fig. D.6 and the results in Tab. D.1 which will be analyzed in the next section. Simply put this example shows that using this approach, the smaller the radius of the balls used in the approximation the higher detail we are able to capture. However this method can be detrimental to approximate the object, because we use an ε-core, where ε depends

[5]This is the case for $SO(3)$, in the case of $SO(2)$ there are exactly 2 neighbours.

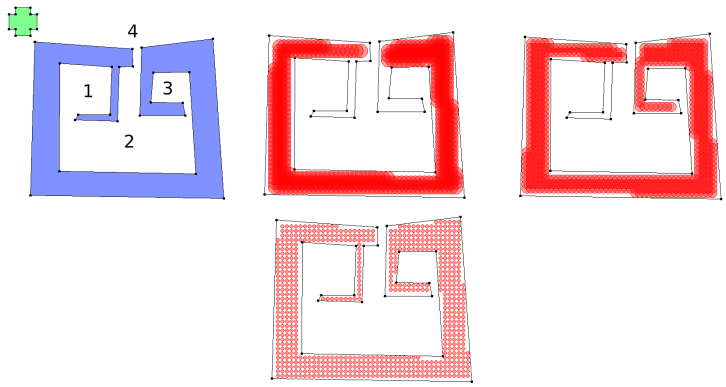


Figure D.6: In the left we present a set of 2D obstacles in blue, and an object in green. The numbers represent the connected components of the free space. In red are three approximations of the obstacles by sets of balls of radius 15, 10, and 4, respectively. Note that the smaller the radius the more features from the original configuration space are preserved. Note that the first approximation significantly simplifies the shape, and has only one narrow passage; the second approximation preserves the shape better and has two narrow passages; the third approximation preserves all the important shape features of the obstacles.

on the dispersion of the grid over $SO(n)$ and the radius of the object and may therefore exceed the necessary ball radius to approximate all the required details.

A more suitable approach to approximate the object is to compute the medial axis of the object using for example the method from [Dey et al. (2006)] which is able to construct it from a pointcloud of the object. Alternatively one can use the method from [Dey et al. (2002)] which is able to retrieve a skelleton of the object (i.e. a 1-dimensional subset of the medial axis). In our experiments we used a water-tight mesh of the drill (see Fig D.8) to extract its skelleton axis and obtained an approximation of the drill by sampling ball centers in the skelleton with the largest radius that did not contain any point in the mesh.

Finally, for the mug and the bugtrap (also Fig. D.8) we traced a slice of the object and manually fit the medial axis of the slice, which we used to create a surface of revolution where the ball centers were placed (bugtrap and body of the mug).

9.2 2D scenario: Different Approximations of the Workspace

In this experiment, we consider how the accuracy of a workspace spherical approximation affects the output and execution time of the algorithm, see Fig. D.6. This experiment was performed on a single thread of Intel Core i7 processor.

For our experiments, we generate a workspace and an object as polygons, and approximate them with unions of balls of equal radii lying strictly inside the polygons.

ε	$R = 15$	$R = 10$	$R = 4$
$0.30 \cdot r$	2 c. 741 ms	3 c. 1287 ms	4 c. 1354 ms
$0.33 \cdot r$	2 c. 688 ms	3 c. 1208 ms	4 c. 1363 ms
$0.37 \cdot r$	2 c. 647 ms	3 c. 1079 ms	4 c. 1287 ms
$0.40 \cdot r$	2 c. 571 ms	3 c. 986 ms	3 c. 1156 ms
$0.43 \cdot r$	2 c. 554 ms	3 c. 950 ms	3 c. 1203 ms

Table D.1: We report the number of path-connected components we found and the computation time for each case. When we were using our first approximation of the workspace, we were able to distinguish only between components 4 and 2 (see Fig. D.6), and therefore prove path non-existence between them. For a more accurate approximation, we were also able to detect component 3. Finally, the third approximation of the workspace allows us to prove path non-existence between every pair of the four components.

Note that the choice of the radius is important: when it is small, we get more balls, which increases the computation time of our algorithm; on the other hand, when the radius is too large, we lose some important information about the shape of the obstacles, because narrow parts cannot be approximated by large balls, see Fig. D.6.

We consider a simple object whose approximation consists of 5 balls. We run our algorithm for all the 3 approximations of the workspace, and take 5 different values of ε, see Tab. D.1. We can observe that as we increase the ε the computation time decreases. This happens because we are using fewer slices. However, we can also observe that when the ε is too large, our approximation of the collision space becomes too small, and we are not able to find one connected component (see the last column of Tab. D.1).

In our opinion, the accuracy of a workspace approximation should depend on the application scenario as well as on the amount of avalibale computational resources.

9.3 3D scenario: Different Types of 3D Caging

As we have mentioned in Sec. 2, a number of approaches towards 3D caging is based on identifying shape features of objects and deriving sufficient caging conditions based on that. By contrast, our method is not restricted to a specific class of shapes, and the aim of this section is to consider examples of objects of different types and run our algorithm on them. The examples are depicted on Fig. D.7, and Tab. D.2 reports execution time for different resolutions of the $SO(3)$ grid.

In all cases (Tab. D.2), the object was shown to be caged when using the third grid. In the case of the narrow part example the algorithm was not able to discern a cage using the first grid. In the rings example it was not able to find the cage with either the second or third grids.

Results that are based on low-resolution grids can in certain cases be inaccurate for two reasons. First, the they have higher dispersion value, which implies that

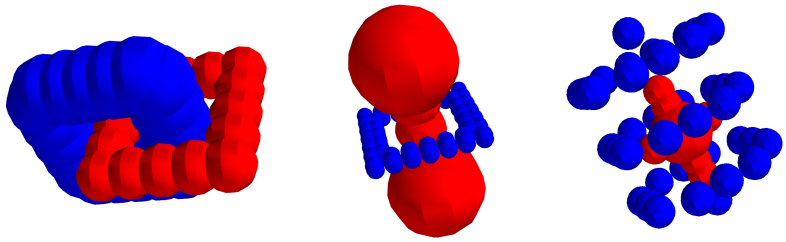


Figure D.7: Different 3D caging scenarios. From left to right: *linking-based caging* ([Pokorny et al. (2013),Stork et al. (2013)]), *narrow part-based caging* ([Varava et al. (2017),Liu et al. (2018)]), *surrounding-based caging.*In linking-based and narrow part-based caging the obstacles form a loop (not necessarily closed) around a handle or narrow part of the object. In surrounding-based the object is surrounded by obstacles so as not to escape.

#slices	rings (320)		narrow part (480)		surrounding (266)	
576	5 s	$\delta = 3.153$	8 s	$\delta = 5.041$	9 s	$\delta = 1.390$
4 608	45 s	$\delta = 0.958$	78 s	$\delta = 1.532$	95 s	$\delta = 0.422$
36 864	485 s	$\delta = 0.000$	674 s	$\delta = 0.000$	875 s	$\delta = 0.000$

Table D.2: Results from running 3D experiments on the objects shown on Fig. D.7 using 3 different resolutions for the $SO(3)$ grid with the non-parallelized algorithm. The number of balls used to approximate the collision space of each model is indicated in parenthesis next to the model name. We report the computational time and the value of clearance δ used to test whether there is a narrow passage of that clearance for each case. The varying clearance corresponds to using the same value for epsilon on each grid.

we need to use a larger value of ε. This results in a smaller $\varepsilon-$core of the object which can escape from a cage even if the entire object cannot. This effect can be mitigated by considering a sufficient clearance δ. Thus, depending on the desired accuracy of the resulting approximation, different numbers of slices can be used. In our current implementation, we precompute grids on $SO(3)$ using the algorithm by [Yershova et al. (2009)], and in our experiments we consider 3 different predefined resolutions. Using a different $SO(3)$-discretization strategy and given a concrete clearance value δ, one can potentially construct a grid on $SO(3)$ with the necessary resolution based on δ.

Another reason for inaccuracies in the low-resolution grid is the fact that the distance between neighboring orientations is so large that adjacent slices might have drastic differences in the topology of their free space. This may lead to adding edges between their connected components that would not be connected if we considered intermediate orientations as well in the case of a more fine grid.

As we see, our algorithm can be applied to different classes of shapes. It certainly cannot replace previous shape-based algorithms, as in some applications global geometric features of objects can be known a priori, in which case it might be faster and easier to make use of the available information. However, in those applications where one deals with a wide class of objects without any knowledge about shape features they might have, our algorithm provides a reasonable alternative to shape-based methods.

9.4 Parallel implementation

Our parallel implementation allows one to run experiments on more complex models of real world objects. In this section, we consider a few objects representing different geometric features, see Fig. D.8. These examples are inspired by manipulation and motion planning applications. The first two models, a mug and a drill, are caged by a Schunk dexterous hand. The mug and the drill illustrate linking-based and narrow part-based caging types, respectively. The third example is a bugtrap and a small cylindrical object. The bugtrap environment is a well-known path planning benchmark created by Parasol MP Group, CS Dept, Texas A & M University. The bugtrap does not cage the object, but contains a narrow passage. This example corresponds to the "surrounding" type of restricting the object's mobility.

First, we observe how the performance of our algorithm changes with respect to the number of slices we consider. Table D.3 reports the results. The respective objects are depicted on Fig. D.8. We report the computational time needed for different resolutions of the orientation grid, as well as the δ value passed to the algorithm. The different δ values correspond to mantaining a constant value of ε across the different grids. In all of the examples, the algorithm was able to detect compact connected components, thus indicating that the objects are caged with clearance δ. In the case of a bugtrap, we were able to detect a narrow passage whose width is at most 0.2. However, when running the same example with clearance parameter $\delta = 0$, the algorithm returned a space approximation with a single connected component, thus indicating that there are no caging configurations. Thus, while the object is not caged by the bugtrap, its free space contains a narrow passage as expected.

#slices	mug (9198 balls)		drill (2646 balls)		bugtrap (2079 balls)	
576	36 s	$\delta = 0.041$	12 s	$\delta = 0.044$	169 s	$\delta = 0.756$
4 608	230 s	$\delta = 0.012$	110 s	$\delta = 0.013$	1 095 s	$\delta = 0.369$
36 864	1 765 s	$\delta = 0.000$	784 s	$\delta = 0.000$	8 263 s	$\delta = 0.200$

Table D.3: Results from running 3D experiments with objects shown on in Fig. D.8 using 3 different resolutions for the $SO(3)$ grid. The number of balls used to approximate the collision space of each model is indicated in parenthesis next to the model name.

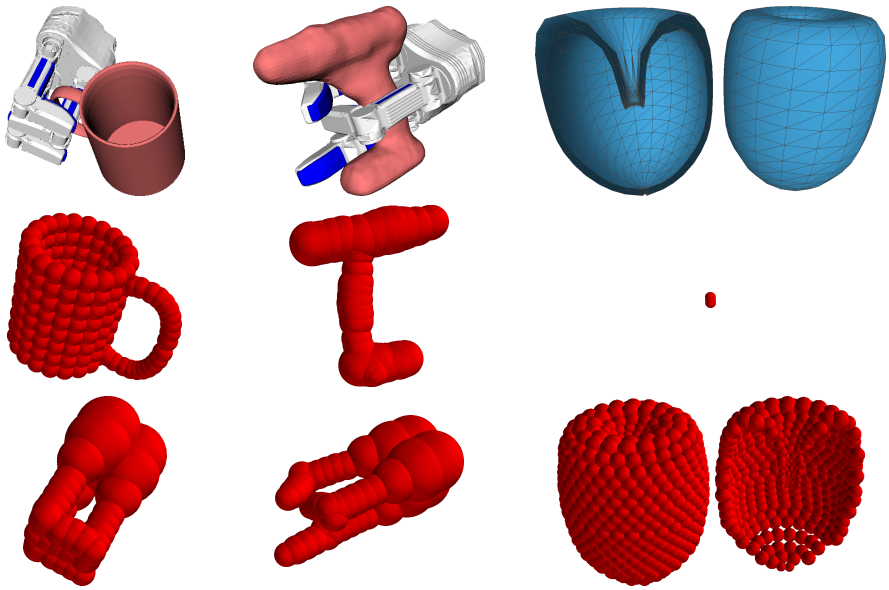


Figure D.8: Caging real-world objects with a 3 finger Schunk hand. First row, from left to right: a mug (linking-based caging), its representations as a union of balls, a drill (narrow part-based caging), a bugtrap (caging by surrounding). Second row: the respective spherical representations of the objects. Third row: a spherical representation of the Schunk hand in the caging configurations used above; an object passing through the bugtrap.

#threads	ring	narrow part	drill
1	95 s	79 s	769 s
2	48 s	40 s	390 s
3	33 s	28 s	270 s
4	26 s	22 s	210 s
5	25 s	20 s	196 s
6	24 s	19 s	184 s
7	22 s	18 s	176 s
8	22 s	18 s	170 s

Table D.4: In this table we show the results from running the experiments on the ring, and narrow part toy examples, as well as on the drill objects, with one to eight threads. These examples stem from using our parallel algorithm (Alg. D8) with the objects and obstacles depicted in Fig. D.7 and Fig. D.8, and a grid over $SO(3)$ comprising 4608 nodes.

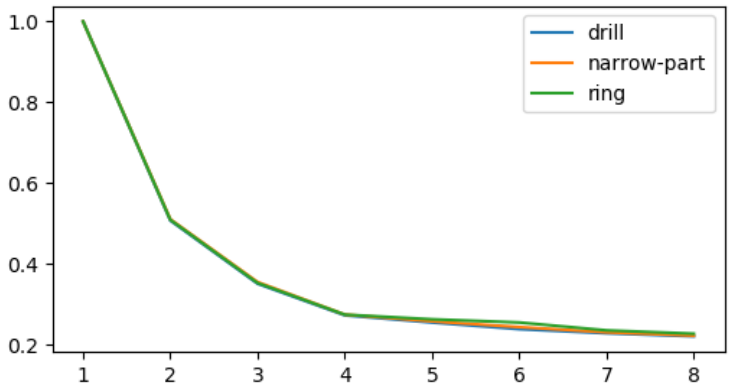


Figure D.9: Relative timing for calculating the configuration space of objects with different numbers of threads. The vertical axis measures time in fraction of time taken by computing with a single thread. The horizontal axis indicates the number of threads. The data corresponds to that presented in Tab. D.4.

Now, we analyze how the performance of the parallelized version of our algorithm changes with respect to the number of threads we use. Table D.4 presents the results of this experiment. We run our algorithm using from one to eight threads on three different object models and report the respective computational time. In all examples we used 4608 slices (second grid) and clearance $\delta = 0$. We observe a drastic increase of performance up to four threads, and a slight increase when using four to eight threads (see Fig. D.9).

10 Discussion and Future Work

In this paper, we provide a computationally feasible and provably-correct algorithm for 3D and 6D configuration space approximation and use it to prove caging and path non-existence. We analyze theoretical properties of the algorithm such as correctness and complexity. We provide a parallel implementation of the algorithm, which allows us to run it on complex object models. In the future, we see several main directions of research that would build upon these results.

10.1 Molecular Cages

In organic chemistry, the notion of caging was introduced independently of the concept of [Mitra et al. (2013)]. Big molecules or their ensembles can form hollow structures with a cavity large enough to envelope a smaller molecule. Simple organic cages are relatively small in size and therefore can be used for selective encapsulation of tiny molecules based on their shape [Mitra et al. (2013)]. In particular, this

property can be used for storage and delivery of small drug molecules in living systems [Rother et al. (2016)]. By designing the caging molecules one is able to control the formation and breakdown of a molecular cage, in such a way as to remain intact until it reaches a specific target where it will disassemble releasing the drug molecule.

An example on Fig. D.10 shows that in principle, our algorithm can be applied to molecular models, assuming they can be treated as rigid bodies. In this example, we consider two molecules to see whether our algorithm can predict their ability to form a cage. Atomic coordinates and van der Waals radii were retrieved from the Cambridge Crystallographic Data Centre (CCDC) database. The depicted molecules are mesitylene (left, CCDC 907758) and CC1 (right, CCDC 707056) ([Mitra et al. (2013)]). Our algorithm reported that this pair forms a cage, as experimentally determined in [Mitra et al. (2013)]. In our future work, we aim to use our algorithm to test a set of potential molecular carriers and ligands to find those pairs which are likely to form cages. This prediction can later be used as a hypothesis for further laboratory experiments.

10.2 Integration with Path Planners

Another possible direction for future work is integration of our approach with sampling-based path planning algorithms. Since the problems of path planning and path non-existence verification are dual to each other, we plan to design a framework where they will be addressed simultaneously: a path planner will attempt to construct a collision-free path as usual, while configuration space approximation will be constructed independently. If there is no path, our configuration space approximation can be used to rigorously demonstrate this instead of relying on heuristical stopping criteria for the planner. Furthermore, we can leverage our approximation to guide sampling, which can be particularly beneficial in the presence of narrow passages. Namely, having an approximation of the narrow regions of the free space, the planner can sample configurations from it.

10.3 Energy-Bounded Caging

[Mahler et al. (2016), Mahler et al. (2018)] defined the concept of energy-bounded caging, when obstacles and external forces (such as gravity, or forces directly applied to the object) complement each other in restricting the object's mobility. The authors proposed an approach towards synthesizing energy-bounded cages in 2D.

To directly extend their method to 3D workspaces one would need to represent the configuration space as a 6D simplicial complex, which is expensive in terms of both required memory and execution time ([Mahler et al. (2016)]). Apart from that, the computation time in their case is dominated by sampling the collision space, and the authors suggest that in the 3D case the necessary number of samples might be even higher.

Analogously to the works by [Mahler et al. (2016), Mahler et al. (2018)], we can model external forces as potential functions and assign potential values to each ball in our approximation of the free space. In each slice, we consider balls with high potential values as "forbidden" parts of the free space, and compute connected components of the rest. In the future, we plan to investigate this direction.

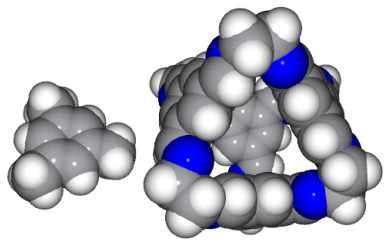


Figure D.10: The small molecule (a guest) can be caged by the big molecule (the host).

11 Acknowledgements

This work has been supported by the Knut and Alice Wallenberg Foundation and Swedish Research Council. The authors are grateful to D. Devaurs, L. Kavraki, and O. Kravchenko for their insights into molecular caging.

References

[Aurenhammer et al. (1984)] Aurenhammer F and Edelsbrunner H (1984) An optimal algorithm for constructing the weighted Voronoi diagram in the plane. In: *Pattern Recognition*, 17(2), pp.251–257.

[Barraquand et al. (1997)] Barraquand J, Kavraki L, Latombe JC, Motwani R, Li TY and Raghavan P (1997) A random sampling scheme for path planning. In: *The International Journal of Robotics Research*, 16(6), pp.759–774.

[Basch et al. (2001)] Basch J, Guibas LJ, Hsu D and Nguyen AT, (2001) Disconnection proofs for motion planning. In: *IEEE International Conference on Robotics and Automation*, vol. 2, pp. 1765–1772.

[Behar and Lien (2013)] Behar, E and Lien, JM (2013) Mapping the configuration space of polygons using reduced convolution In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1242–1248.

[Dey et al. (2002)] Dey TK and Zhao W (2012) Approximate medial axis as a voronoi subcomplex. In: *Proceedings of the seventh ACM symposium on Solid modeling and applications*, pp. 356–366.

[Dey et al. (2006)] Dey TK and Sun J (2006) Normal and feature approximations from noisy point clouds. In: *International Conference on Foundations of Software Technology and Theoretical Computer Science* pp. 21–32.

[Edelsbrunner (1999)] Edelsbrunner H (1999) Deformable smooth surface design. In: *Discrete & Computational Geometry*, 21(1), pp. 87–115.

[Guibas et al. (1985)] Guibas L, and Stolfi J, (1985) Primitives for the manipulation of general subdivisions and the computation of Voronoi. In: *ACM transactions on graphics*, 4(2), pp.74–123.

[Kuperberg (1990)] Kuperberg W (1990) Problems on polytopes and convex sets In: *DIMACS Workshop on polytopes*, pp. 584–589.

[Latombe (1991)] Latombe JC (1991) *Robot motion planning.* vol. 124. Springer International Series in Engineering and Computer Science. Springer US.

[Liu et al. (2018)] Liu J, Xin S, Gaol Z, Xu K, Tu C and Chen B (2018) Caging loops in shape embedding space: theory and computation. In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1–5.

[Lozano-Perez (1983)] Lozano-Perez T (1990) Spatial planning: A configuration space approach. In: *Autonomous robot vehicles*, pp. 259–271. Springer, New York, NY.

[Mahler et al. (2016)] Mahler J, Pokorny FT, McCarthy Z, van der Stappen AF and Goldberg K (2016) Energy-bounded caging: Formal definition and 2-D energy lower bound algorithm based on weighted alpha shapes. In: *IEEE Robotics and Automation Letters*, 1(1), pp.508–515.

[Mahler et al. (2018)] Mahler J, Pokorny FT, Niyaz S and Goldberg K (2018) Synthesis of energy-bounded planar caging grasps using persistent homology. In: *IEEE Transactions on Automation Science and Engineering*, 15(3), pp.908–918.

[Makita and Maeda (2008)] Makita S and Maeda Y (2008) 3D multifingered caging: Basic formulation and planning. In: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2697–2702.

[Makita et al. (2013)] Makita S, Okita K and Maeda Y (2013) 3D two-fingered caging for two types of objects: sufficient conditions and planning. In: *IJMA*, 3(4), pp.263–277.

[Makita and Wan (2017)] Makita S and Wan W (2017) A survey of robotic caging and its applications. In: *Advanced Robotics*, 31(19–20), pp.1071–1085.

[Makapunyo et al. (2013)] Makapunyo T, Phoka T, Pipattanasomporn P, Niparnan N and Sudsang A (2013) Measurement framework of partial cage quality based on probabilistic motion planning. In: *IEEE International Conference on Robotics and Automation*, pp. 1574–1579.

[McCarthy et al. (2012)] McCarthy Z, Bretl T and Hutchinson S (2012) Proving path non-existence using sampling and alpha shapes. In: *IEEE International Conference on Robotics and Automation*, pp. 2563–2569.

[Milenkovic et al. (2013)] Milenkovic V, Sacks E and Trac S (2013) Robust complete path planning in the plane. In: *Algorithmic Foundations of Robotics X*, pp. 37–52. Springer, Berlin, Heidelberg.

[Mitra et al. (2013)] Mitra T, Jelfs KE, Schmidtmann M, Ahmed A, Chong SY, Adams DJ and Cooper AI (2013) Molecular shape sorting using molecular organic cages In: *Nature chemistry*, 5(4), p.276.

[Pereira et al. (2004)] Pereira GA, Campos MF and Kumar V (2004) Decentralized algorithms for multi-robot manipulation via caging. In: *The International Journal of Robotics Research*, 23(7–8), pp.783–795.

[Pipattanasomporn and Sudsang (2006)] Pipattanasomporn P and Sudsang A (2006) Two-finger caging of concave polygon. In: *Proceedings IEEE International Conference on Robotics and Automation* pp. 2137–2142.

[Pipattanasomporn and Sudsang (2011)] Pipattanasomporn P and Sudsang A (2011) Two-finger caging of nonconvex polytopes. In: *IEEE Transactions on Robotics*, 27(2), pp.324–333.

[Pokorny et al. (2013)] Pokorny FT, Stork JA and Kragic D (2013) Grasping objects with holes: A topological approach. In: *2013 IEEE International Conference on Robotics and Automation*, pp. 1100–1107.

[Rimon and Blake (1999)] Rimon E and Blake A (1999) Caging planar bodies by one-parameter two-fingered gripping systems. In: *International Journal of Robotics Research*, 18(3), pp.299–318.

[Rodriguez and Mason (2012)] Rodriguez A and Mason MT (2012) Path connectivity of the free space. In: *IEEE Transactions on Robotics*, 28(5), pp.1177–1180.

[Rodriguez et al. (2012)] Rodriguez A, Mason MT and Ferry S (2012) From caging to grasping. In: *The International Journal of Robotics Research*, 31(7), pp.886–900.

[Rother et al. (2016)] Rother M, Nussbaumer MG, Renggli K and Bruns N (2016) Protein cages and synthetic polymers: a fruitful symbiosis for drug delivery applications, bionanotechnology and materials science. In: *Chemical Society Reviews*, 45(22), pp.6213–6249.

[Stork et al. (2013)] Stork JA, Pokorny FT and Kragic D (2013) Integrated motion and clasp planning with virtual linking. In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3007–3014.

[Vahedi and van der Stappen (2008)] Vahedi M and van der Stappen AF (2008). Caging polygons with two and three fingers. In: *The International Journal of Robotics Research*, 27(11–12), pp.1308–1324.

[Varava et al. (2016)] Varava A, Kragic D and Pokorny FT (2016) Caging grasps of rigid and partially deformable 3-D objects with double fork and neck features. In: *IEEE Transactions on Robotics*, 32(6), pp.1479–1497.

[Varava et al. (2017)] Varava A, Carvalho JF, Pokorny FT and Kragic D (2017) Caging and path non-existence: a deterministic sampling-based verification algorithm. In: *International Symposium on Robotics Research*

[Varava et al. (2018)] Varava A, Carvalho JF, Pokorny FT and Kragic D (2018) Free Space of Rigid Objects: Caging, Path Non-Existence, and Narrow Passage Detection. In: *Workshop on Algorithmic Foundations of Robotics*.

[Wang and Kumar (2002)] Wang Z and Kumar V (2002) Object closure and manipulation by multiple cooperating mobile robots. In: *Proceedings IEEE International Conference on Robotics and Automation*, pp. 394–399.

[Wise and Bowyer (2000)] Wise KD and Bowyer A (2000) A survey of global configuration-space mapping techniques for a single robot in a static environment. In: *The International Journal of Robotics Research*, 19(8), pp.762–779.

[Yershova et al. (2009)] Yershova A, Jain S, Lavalle SM and Mitchell JC (2010) Generating uniform incremental grids on SO(3) using the Hopf fibration. In: *The International journal of robotics research*, 29(7), pp.801–812.

[Zhang et al. (2008)] Zhang L, Kim YJ and Manocha D (2008) Efficient cell labelling and path non-existence computation using C-obstacle query. In: *The International Journal of Robotics Research*, 27(11–12), pp.1246–1257.

[Zhu and Latombe (1991)] Zhu DJ and Latombe JC (1991) New heuristic algorithms for efficient hierarchical path planning. In: *IEEE Transactions on Robotics and Automation*, 7(1), pp.9–20.

[Zomorodian and Edelsbrunner (2000)] Zomorodian A and Edelsbrunner H (2002) Fast software for box intersections. In: *International Journal of Computational Geometry & Applications*, pp.143–172.

A Appendix

Recall the formulation of Proposition 6.

Given two unit quaternions p, q, the following equation holds:

$$D(R_{p\bar{q}}) = 2\sin(\rho(p,q))\,\mathrm{rad}(\mathcal{O})$$

Proof. For notational simplicity we divide both sides of the equation by $\mathrm{rad}(\mathcal{O})$ and define $\bar{D}(R_q) = \frac{D(R_q)}{\mathrm{rad}(\mathcal{O})}$. Which reduces the problem to proving:

$$\bar{D}(R_{p\bar{q}}) = 2\sin(\rho(p,q))$$

We proceed by reducing both sides of the equation to the same formula, starting with the left-hand-side. To do this we once again point out that we can identify quaternions with vectors in $\mathbb{R}^4$, and that a unit quaternion $q = \cos(\frac{\theta_q}{2}) + \sin(\frac{\theta_q}{2})(q_x i + q_y j + q_z k)$ is associated to a 3D rotation of an angle of θ_q around the axis (q_x, q_y, q_z) which we will denote w_q.

$$\begin{aligned}
\bar{D}(R_{p\bar{q}}) &= 2|\Im p\bar{q}| \\
&= 2\|\cos(\frac{\theta_q}{2})\sin(\frac{\theta_p}{2})w_p - \cos(\frac{\theta_p}{2})\sin(\frac{\theta_q}{2})w_q - \sin(\frac{\theta_p}{2})\sin(\frac{\theta_q}{2})w_p \times w_q\| \\
&= 2\sqrt{\|\cos(\frac{\theta_q}{2})\sin(\frac{\theta_p}{2})w_p - \cos(\frac{\theta_p}{2})\sin(\frac{\theta_q}{2})w_q\|^2} \\
&\quad \overline{+\|\sin(\frac{\theta_p}{2})\sin(\frac{\theta_q}{2})w_p \times w_q\|^2}
\end{aligned}$$

Where the last equality is due to the fact that $w_p \times w_q$ is perpendicular to both w_p and w_q, and is therefore a consequence of the Pythagorean theorem. Now recall that $\|w_p \times w_q\| = \sin(\omega_{p,q})$ where $\omega_{p,q}$ is the angle between w_p, w_q and that $\langle w_p, w_q\rangle = \cos(\omega_{p,q})$, since $\|w_p\| = \|w_q\| = 1$. Recall also that $\sin^2(\theta) = 1 - \cos^2(\theta)$, whence we obtain

$$\begin{aligned}
\bar{D}(R_{p\bar{q}}) = 2\sqrt{\|\cos(\frac{\theta_q}{2})\sin(\frac{\theta_p}{2})w_p - \cos(\frac{\theta_p}{2})\sin(\frac{\theta_q}{2})w_q\|^2} \\
\overline{+\sin^2(\frac{\theta_p}{2})\sin^2(\frac{\theta_q}{2})(1 - \langle w_p, w_q\rangle^2)}
\end{aligned}$$

Furthermore let $\tilde{w}_p = \frac{w_q - \langle w_p, w_q\rangle w_p}{\|w_q - \langle w_p, w_q\rangle w_p\|}$ be the component of w_q which is perpendicular to w_p, then we can rewrite $\cos(\frac{\theta_p}{2})\sin(\frac{\theta_q}{2})w_q$ as

$$\cos(\frac{\theta_p}{2})\sin(\frac{\theta_q}{2})w_q = \cos(\frac{\theta_p}{2})\sin(\frac{\theta_q}{2})(\langle w_p, w_q\rangle w_p + \langle \tilde{w}_p, w_q\rangle \tilde{w}_p)$$

Substituting this into the formula, and using the pythagorean theorem to separate he w_p and $\tilde{w}_p$ components, we can proceed with

$$\bar{D}(R_{p\bar{q}}) = \frac{2\sqrt{\|(\cos(\frac{\theta_q}{2})\sin(\frac{\theta_p}{2}) - \cos(\frac{\theta_p}{2})\sin(\frac{\theta_q}{2})\langle w_p, w_q\rangle)w_p\|^2}}{+\|\cos(\frac{\theta_p}{2})\sin(\frac{\theta_q}{2})\langle w_q, \tilde{w}_p\rangle\tilde{w}_p\|^2 + \sin^2(\frac{\theta_p}{2})\sin^2(\frac{\theta_q}{2})(1 - \langle w_p, w_q\rangle^2)}$$

Now we note that all of $\|w_q\|, \|w_p\|, \|\tilde{w}_p\|$ have norm 1, and therefore

$$\|\langle w_p, w_q\rangle w_p + \langle w_q, \tilde{w}_p\rangle\tilde{w}_p\|^2 = 1$$

By the Pythagorean, this theorem gives $\langle w_q, \tilde{w}_p\rangle^2 = 1 - \langle w_p, w_q\rangle^2$, which we can substitute once again.

$$\bar{D}(R_{p\bar{q}}) = \frac{2\sqrt{\left(\cos(\frac{\theta_q}{2})\sin(\frac{\theta_p}{2}) - \cos(\frac{\theta_p}{2})\sin(\frac{\theta_q}{2})\langle w_p, w_q\rangle\right)^2}}{+\cos^2(\frac{\theta_p}{2})\sin^2(\frac{\theta_q}{2})(1 - \langle w_p, w_q\rangle^2) + \sin^2(\frac{\theta_p}{2})\sin^2(\frac{\theta_q}{2})(1 - \langle w_p, w_q\rangle^2)}$$

Now we want to deal only with a combination of tangents, therefore we divide the term inside the square root by $\cos^2(\frac{\theta_p}{2})\cos^2(\frac{\theta_q}{2})$ yielding:

$$\bar{D}(R_{p\bar{q}}) = \frac{2|\cos(\frac{\theta_p}{2})\cos(\frac{\theta_q}{2})|\sqrt{\left(\tan(\frac{\theta_p}{2}) - \tan(\frac{\theta_q}{2})\langle w_p, w_q\rangle\right)^2}}{+\tan^2(\frac{\theta_q}{2})(1 - \langle w_p, w_q\rangle^2) + \tan^2(\frac{\theta_p}{2})\tan^2(\frac{\theta_q}{2})(1 - \langle w_p, w_q\rangle^2)}$$

Now, expanding the sqares and multiplying into all the terms under the squareroot sign, as well as eliminating terms that cancel out, results in:

$$\bar{D}(R_{p\bar{q}}) = \frac{2|\cos(\frac{\theta_p}{2})\cos(\frac{\theta_q}{2})|\sqrt{\tan^2(\frac{\theta_p}{2}) - 2\tan(\frac{\theta_p}{2})\tan(\frac{\theta_q}{2})\langle w_p, w_q\rangle + \tan^2(\frac{\theta_q}{2})}}{+\tan^2(\frac{\theta_p}{2})\tan^2(\frac{\theta_q}{2}) - \tan^2(\frac{\theta_p}{2})\tan^2(\frac{\theta_q}{2})\langle w_p, w_q\rangle^2}$$

By introducing an extra $1 - 1$ into the square root, we can use these terms to complete products in order to simplify the equation.

$$\begin{aligned}\bar{D}(R_{p\bar{q}}) =& 2|\cos(\frac{\theta_p}{2})\cos(\frac{\theta_q}{2})|\sqrt{1+\tan^2(\frac{\theta_p}{2})+\tan^2(\frac{\theta_q}{2})+\tan^2(\frac{\theta_p}{2})\tan^2(\frac{\theta_q}{2})} \\ & \overline{-(1+\tan(\frac{\theta_p}{2})\tan(\frac{\theta_q}{2})\langle w_p, w_q\rangle)^2} \\ =& 2|\cos(\frac{\theta_p}{2})\cos(\frac{\theta_q}{2})|\sqrt{(1+\tan^2(\frac{\theta_p}{2}))(1+\tan^2(\frac{\theta_q}{2}))} \\ & \overline{-(1+\tan(\frac{\theta_p}{2})\tan(\frac{\theta_q}{2})\langle w_p, w_q\rangle)^2}\end{aligned}$$

Finally, recall that $1+\tan^2(\theta) = \frac{1}{\cos^2(\theta)}$, which gives us

$$\bar{D}(R_{p\bar{q}}) = 2\sqrt{1-\cos^2(\frac{\theta_p}{2})\cos^2(\frac{\theta_q}{2})(1+\tan(\frac{\theta_p}{2})\tan(\frac{\theta_q}{2})\langle w_p, w_q\rangle)^2}$$

Now we begin to explore the right-hand side of the equation, by noting that when $\sin(\theta) > 0$ then $\sin(\theta) = |\sin(\theta)| = \sqrt{\sin^2(\theta)} = \sqrt{1-\cos^2(\theta)}$. Furthermore we note that $\cos^{-1}$ maps $[-1,1]$ to $[0,\pi]$ and particularly it maps $[0,1]$ to $[0,\pi/2]$ where the sine function is positive, therefore, we get

$$\begin{aligned}&2\sin(\rho(p,q)) = 2\sin(\cos^{-1}(|\langle p,q\rangle|)) \\ &= 2\sqrt{1-\cos^2(\cos^{-1}(|\langle p,q\rangle|))} \\ &= 2\sqrt{1-\langle p,q\rangle^2} \\ &= 2\sqrt{1-(\cos(\frac{\theta_p}{2})\cos(\frac{\theta_q}{2})+\sin(\frac{\theta_p}{2})\sin(\frac{\theta_q}{2})\langle w_p, w_q\rangle)^2}\end{aligned}$$

And finally, we get the same formula as before:

$$2\sin(\rho(p,q)) = 2\sqrt{1-\cos^2(\frac{\theta_p}{2})\cos^2(\frac{\theta_q}{2})(1+\tan(\frac{\theta_p}{2})\tan(\frac{\theta_q}{2})\langle w_p, w_q\rangle)^2}$$

Hence concluding the proof of Proposition 6. □