Spatial Abstraction of Trace Data

JOHN-PAUL ORE, University of Nebraska, Lincoln, Computer Science and Engineering
MITCH GERRARD, University of Nebraska, Lincoln, Computer Science and Engineering

Ubiquitous position sensors in cars, phones, ships, and satellites continually collect time-sequenced records of spatial coordinates. Analyzing this data to extract useful information is an area of intense study in machine learning, geospatial sciences, and data mining. In this paper, we explore the application of software analysis techniques to this problem, specifically trace analysis for property and invariant detection. We present a novel technique for abstracting away from individual points and explore and discuss properties that can be found within this spatial abstraction for one or more moving things. We implement our technique in a prototype tool and present examples from the application of this tool to real-world data.

Categories and Subject Descriptors: I.2.4 [**Knowledge Representation Formalisms and Methods**]: Representations (procedural and rule-based)

General Terms: Design, Algorithms

Additional Key Words and Phrases: Spatial Abstraction, Knowledge Representation

1. INTRODUCTION

The world is full of moving things, and increasingly, things know where they are. Ubiquitous sensor technologies, like GPS, combined with networked storage enables us to continually detect and archive spatial movement. These myriad data points can answer questions of great value, including geospatial distributions, trends and traffic flows, popular restaurants, migration patterns and much more.

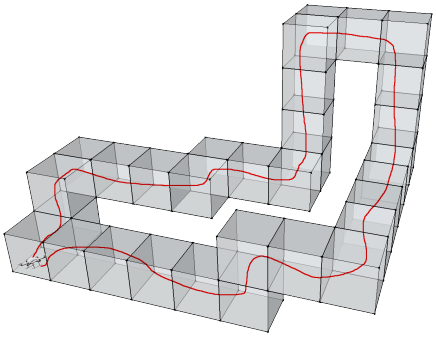

Fig. 1: Unit cube space overapproximates path.

However, we can be overwhelmed because we are "drowning in data" [Morse 1993]. Further, it can be difficult to examine high-level relationships because points and lines are specific; the story is at the level of the forest and we can only see the trees. Often it is only feasible to analyze large data sets by choosing an appropriate level of abstraction. The work presented here explores one such kind of abstraction. We consider a "spatial abstraction" where particular points in space-time are mapped to unit cubes, as shown in Fig. 1. Using techniques inspired by software analysis, specifically Daikon invariants [Kataoka et al. 2001], we scan traces of spatial movement, and like Daikon, search for properties that are true within the abstraction.

Author's addresses: J.P. Ore and M. Gerrard, Computer Science and Engineering, 256 Avery Hall, University of Nebraska, Lincoln, Nebraska, 68588.

DOI:http://dx.doi.org/10.1145/0000000.0000000

In this paper, we set forth our technique for abstracting away from particular points, implement our technique in a prototype tool, *Abstractus*, collect real movement data from a small flying robot and analyze it, and validate our tool's ability to detect spatial properties on one or more actors.

The paper is organized as follows: Sec. 2 gives an overview of both spatial analysis and the Daikon invariant analysis tool; Sec. 3 describes the technique of spatial abstraction, including defining terms and a table of formalized properties; Sec. 4 provides an overview of our prototype tool that implements this method of analysis; Sec. 5 shows an initial application of our tool to a real robotic system; Sec. 6 situates this work relative to other kinds of movement analysis; in Sec. 7 we reflect on some unexpected and curious features of spatial abstraction; and finally, Sec. 8 summarizes this work and describes future lines of inquiry.

2. BACKGROUND

Many people conceive of space in the manner of Descartes, where three perpendicular axes are used to identify every point. Yet other observers follow Minkowski [Minkowski 1952] and imagine curvature in space time. We are interested in a coarsening of these perspectives, imagining the universe cut into regular stacks of identical boxes, akin to an occupancy grid [Moravec and Elfes 1985], or to use more technical terms, a tessellation of $\mathbb{R}^3$by a unit cuboid.

This kind of coarsening is used in robot planning to reason about collision-free trajectories, obstacle avoidance, and planning into the future [Siegwart et al. 2011].

In this paper, we consider a similar over-approximation of the position of some *actor* within space. The key difference is that we consider properties of spatial abstractions that happened *in the past*, instead of the future. To do this, we adopt a very successful methodology from software engineering implemented in the tool Daikon [Kataoka et al. 2001]. Daikon uses a record of what happened during software execution to variable values at particular program points, and this record is called a *program trace*. Daikon infers high-level properties of software behavior based on these traces. We consider a *spatial trace*, a finite time sequence of positions, and try to reason about higher-level properties of movement.

3. TECHNIQUE

In this section, we formalize our technique for reasoning about movement within a coarse 'unit cube' space. Starting from spatial traces, we map sequences of positions into both occupied cubes as well as nodes in a graph, then discuss three classes of properties: individual actor properties, multiple actor properties, and properties of paths.

3.1. Foundations and Definitions

We begin with a set of time-sequenced traces $\mathcal{T}$ of spatial data, with an individual trace $T \in \mathcal{T} = (t, x, y, z)$ where $t \in \mathbb{R}$ is a time relative to a global start time, and $x, y, z \in \mathbb{R}$ are familiar position co-ordinates relative to a world frame. We assume all traces share identical time and world frames, or they have been synchronized by pre-processing. Let trace T be indexed by i so that T_i is the ith time-ordered record in T, and let the set of traces $\mathcal{T}$ be indexed so T^i is the ith trace in $\mathcal{T}$.

Each trace pertains to an *actor*, designated collectively as the set A and individually as $\bigstar$. The relation B links an actor $\bigstar^i$ to trace T^i.

Let there exist a space $Cube^3 \equiv \mathbb{Z}^3$, an integer space. In this space we denote a function f_{cube} that maps from a trace record to a cube, $f_{cube} : T_i \rightarrow cube_i$, thereby coarsening it.

$$f_{cube}(t, x, y, z) = (\lfloor \alpha * x \rfloor, \lfloor \alpha * y \rfloor, \lfloor \alpha * z \rfloor) \tag{1}$$

The symbol α in Eq. 1 denotes a scaling factor to convert between the original unit of measure to the size of the unit cube (See Sec. 7 for a discussion of choosing the size of the unit

cube). For each record in a trace T, we both map its position to a cube and add a node to a multigraph $G = (V, E)$, if it does not already exist. G can be thought of as embedded within $Cube^3$.

$$cube_i \Rightarrow v_i \in V \tag{2}$$

Cubes and nodes in G are in a one-to-one correspondence within a relation R.

If a trace T contains two consecutive records T_i and T_{i+1} that map to different cubes, then we add an edge (v_i, v_{i+1}) to E. Every edge is assigned a label of the time t from T_{i+1} with a label function w, so that the edge label indicates the time when $\bigstar^i$ transitioned from $cube_i$ to $cube_{i+1}$. For every edge $e \in E$ we associate a predecessor and successor, identified in relations F_{prev} and F_{next}, denoted collectively as F. This means each edge in G corresponds to exactly one actor.

$$(f_{cube}(T_i) \neq f_{cube}(T_{i+1})) \Rightarrow (v_i, v_{i+1}) \in E \tag{3}$$

We denote a path between vertices v_i and v_j by $P(v_i, v_j)$ and consider this notation equivalent:

$$P(v_i, v_j) \equiv P(cube_i, cube_j) \tag{4}$$

A spatial abstraction, then, is the tuple:

$$\text{🗗} = (\mathcal{T}, \mathsf{A}, \mathsf{G}, \mathsf{R}, Cube^3, \mathsf{F}, \mathsf{B}, w) \tag{5}$$

A property, ϕ is said to be true for a spatial abstraction 🗗 if the definition of ϕ holds for $(\mathcal{T}, \mathsf{A}, \mathsf{G}, \mathsf{R}, Cube^3, \mathsf{F}, \mathsf{B}, w)$. We denote this by

$$\text{🗗} \models \phi \tag{6}$$

And say a spatial abstraction "models" ϕ. One spatial abstraction can model multiple properties, and properties can pertain to particular actors, groups of actors, paths, or cubes. A list of properties is presented in tabular form below.

3.2. Properties of Single Actors

Note: all properties in this table pertain to a particular actor $\bigstar^i$.

PROPERTY	FORMALISM
Moves	$\phi_{\{moves\}} = \lvert V \rvert > 1$
Returns Home	$\phi_{\{returnhome\}} = \exists P(cube_{home}, cube_{home})$
No repetition	$\phi_{\{noRepetition\}} = \forall (v_a, v_b), (v_c, v_d) in E, v_b \neq v_c, v_a \neq v_d$
Loiters	$\phi_{\{loiters\}} = \exists e \in E$ s.t. $w(e) - w(F_{prev}(e)) > k$; k is time bound
Teleports	$\phi_{\{teleport\}} = \exists cube_i, cube_j$ s.t. $(F_{next}(cube_i, cube_j) \wedge \neg Adjacent(cube_i, cube_j))$
Hot Box	$\phi_{\{hotBoxOfSizeN\}} = \exists cube_i$ s.t. $\lvert e \in E \lvert F_{next}(e) = v_i \rvert > n$

Informally, the above properties correspond to the following descriptions: "Moves" checks whether the actor visits any cube other than its initial one. "Returns home" check whether the actor leaves the initial cube and returns to it at a later point in time. "No repetition" checks that an actor does not re-enter some cube after leaving it. "Loiters" checks that an actor does not stay in some cube for longer than some given time bound. "Teleports" checks that the cube trace is connected within the cube-map, meaning there are no "skipped," or vacant cubes between successive time steps. "Hot box" checks if any cube in the map has been visited by the same $\bigstar^i$ at least n times.

3.3. Properties of Multiple Actors

PROPERTY	DEFINITION
Independence	$\phi_{\{independence\}} = \forall V_{\bigstar^i}, V_{\bigstar^j} \in V, i \neq j \rightarrow V_{\bigstar^i} \cap V_{\bigstar^j} = \emptyset$
k-independence	$\phi_{\{k-independence\}} = Adjacent(V_{\bigstar^i}, k) \cap Adjacent(V_{\bigstar^j}, k) = \emptyset$
Following	$\phi_{\{following\}} = \exists E' \subset E_1 \wedge E'' \subset E_2$ s.t. $P(E') = P(E'')$
Dense Space	$\phi_{\{dense\}} = \exists S \subset Cube^3$ s.t. $> k\bigstar$'s occupy S at time t
Sparse Space	$\phi_{\{sparse\}} = \exists S \subset Cube^3$ s.t. $< k\bigstar$'s occupy S at time t
Slow Traffic	$\phi_{\{slowTraffic\}} = \exists S \subset Cube^3$ s.t. $> \forall \bigstar \in S, \boxplus_S \models \phi_{\{loiters>k\}}$
Fast Traffic	$\phi_{\{fastTraffic\}} = \exists S \subset Cube^3$ s.t. $> \forall \bigstar \in S, \boxplus_s \models \phi_{\{loiters<k\}}$
Mobbing	$\phi_{\{mobbing\}} = \exists [t_i, t_j] \wedge S \subset Cube^3$ s.t. $\phi\{dense\}(t_i) < \phi\{dense\}(t_j)$
Fleeing	$\phi_{\{fleeing\}} = \exists [t_i, t_j] \wedge S \subset Cube^3$ s.t. $\phi\{dense\}(t_i) > \phi\{dense\}(t_j)$
SConnected	$\phi_{\{SConnected\}} = \exists SCC \subset G$
Above	$\phi_{\{above\}} = T_i^1[z] > T_i^2[z] \forall i$ where i is indexing a time interval
Below	$\phi_{\{below\}} = T_i^1[z] < T_i^2[z] \forall i$ where i is indexing a time interval
Front	$\phi_{\{front\}} = T_i^1[x] > T_i^2[x] \forall i$ where i is indexing a time interval
Behind	$\phi_{\{behind\}} = T_i^1[x] < T_i^2[x] \forall i$ where i is indexing a time interval
Right	$\phi_{\{right\}} = T_i^1[y] > T_i^2[y] \forall i$ where i is indexing a time interval
Left	$\phi_{\{left\}} = T_i^1[y] < T_i^2[y] \forall i$ where i is indexing a time interval

Informally, the above properties correspond to the following descriptions. "Independence" checks that any two actors never occupy the same cube at any point in a trace. "Following" checks to see if any two actors share a given cube-path at any point in a trace. "Dense" checks to see if many actors occupy some subspace of $Cube^3$ at a given time. "Sparse" checks to see if relatively few actors occupy some subspace of $Cube^3$ at a given time. "Slow Traffic" identifies a subspace where the time difference of all successive edges is greater than a value k. "Fast Traffic" identifies a subspace where the time difference of all successive edges is less than a value k. "Mobbing" identifies a time window during which a subspace has more actors entering that leaving. "Fleeing" identifies a time window during which a subspace has more actors leaving than entering. "SConnected" identifies a strongly connected component (SCC) in the graph. Collectively, SCCs partition $Cube^3$ into disconnected subspaces. This property can hold for one or more actors. The "above/below" and remaining properties specify partial order properties corresponding to the six directions in $\mathbb{R}^3$for some time-interval within a trace.

3.4. Properties of Paths

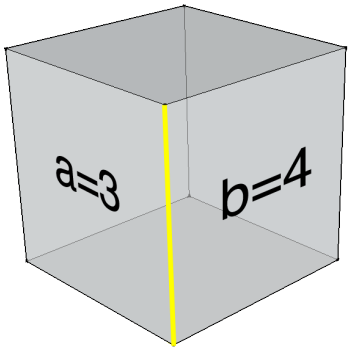

Fig. 2: The ratio cube.

3.4.1. The Ratio Cube. In order to compare multiple paths, we consider for each path P a method of counting how many times an actor moves in each direction. In our model, there are six directions: up, down, left, right, forward, back, and we assign a counter to each. We then compare the ratios of the counters, and use this ratio as a way of comparing multiple paths. A visual representation of these counters and the ratio between directions is shown in Fig. 2, and shows an example where a $P(cube_i, cube_j)$ moves in direction a three units and direction b four units. The cube ratio is undefined when the denominator is zero.

As shown in the figure, the number of moves in each direction is associated with a face of the ratio cube, and the ratios between the directions is associated with an edge of the cube, depicted by a yellow line.

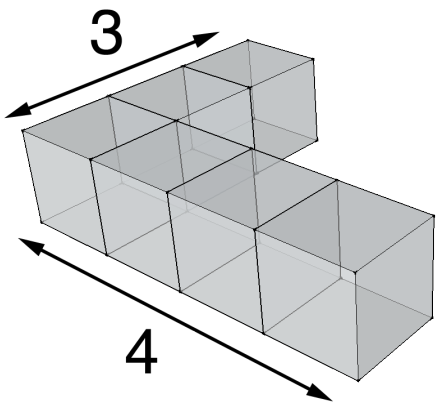

Fig. 3: A path with ratio $a : b = 3 : 4$

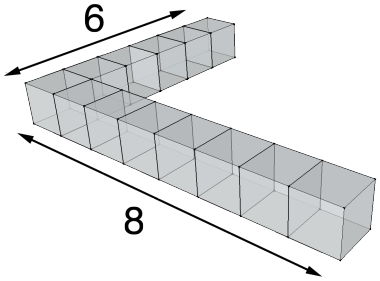

Fig. 4: Another path with ratio $a : b = 3 : 4$

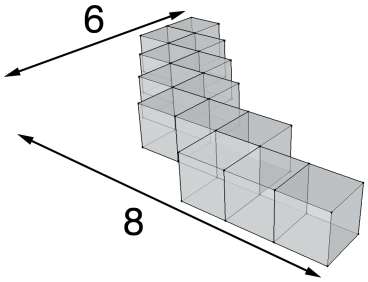

Fig. 5: Alternate path also with ratio $a : b = 3 : 4$

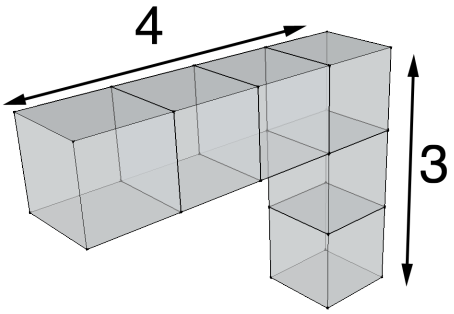

Fig. 6: Path that after rotational transformation has ratio $a : b = 3 : 4$

Figs. 3-4 show ratio equivalent paths, that is, the ratio of the number of moves made in each of the six directions is the same. Note than these ratios are invariant under transformation by scale. Fig. 5 depicts a path that makes the same ratio of moves in a different order, and therefore all paths with identical ratios in the edge cube are likewise path ratio equivalent.

An interesting case is exhibited in Fig. 6, where the ratio is the same but along another edge. We consider this path to be *rotationally* ratio equivalent to the paths in Figs.3-5a. There are 24 rotational symmetries for a cube (one is the identity), so searching for ratio equivalence given two paths is tractable.

Cube ratios can also be useful for measuring the similarity of paths. If the edge ratio is not identical but approximately the same, the two paths are *approximately* edge ratio equivalent, to the extent of the measure of the difference of their ratios.

4. DESCRIPTION OF TOOL

In this section, we describe the implementation of our tool for reasoning about spatial properties. In the previous section, we used the language of *cubes* to discuss the general methodology. In discussing the tool, we use the language of *boxes*, *box traces* and *box sizes* as a way to differentiate the theory from the concrete implementation, which differ in small ways. Our tool,

written in Ruby, takes trace files, a box size, and bounding dimensions as input, and returns spatial-temporal properties inferred from the traces.

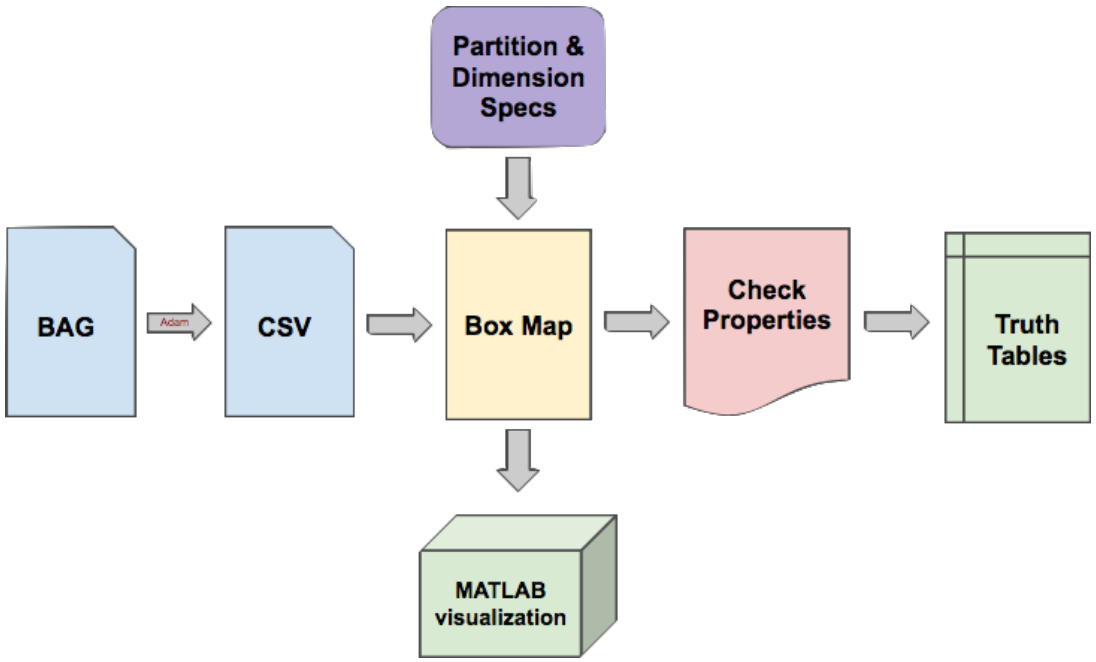

Fig. 7: Overview of tool workflow.

This description elaborates on the high-level workflow diagram shown in Fig. 7. The trace files will be one or more .bag files. A bag is the file format used by the Robot Operating System (ROS) for storing ROS message data. Via messages, various sensors can write their values to these bags during an execution. These values may be examined after any run. In our current implementation, we are only considering the values of time stamps and x, y, and z positions.

We first convert the .bag files to .csv files with four columns which contain a timestamp and the associated x, y, and z position values (in meters). Our tool currently makes the assumption that we receive one *complete* trace of some run, so we are not stitching together partitions of lengthy runs; this may need to be done in the future. Each time frame is mapped to some box, so to temporally sync the runs, we are considering the start of the run to be the global time frame of the greatest initial time frame of all given traces.

The box size is given as a single positive floating-point integer (will be a cubic box), representing the desired length of the box edges in meters. The bounding dimensions are given as a 3-tuple of positive floating-point integers, representing the bounds of the space in which the actors move. Given the box size, we map the position of each time frame of a single trace to the box which corresponds to an actor's position in $\mathbb{R}^3$. This produces a total ordering of abstracted trace positions given as a succession of boxes, which we will call a box-trace.

The output will be a single truth table corresponding to the property you wish to check. You may optionally output a file used by MATLAB for 3D visualization of the box-trace, as shown in Fig. 8.

If we are considering a single actor, given m properties, the truth table will just be one row with m columns containing the truth value of the corresponding property. If we are considering

n multiple actors, the output will be a truth table given as a $1 \times n$ or $n \times n$ matrix, depending on the property considered. For example, the returns home property only depends on a single actor, not their interactions, so the table will be a $1 \times n$ matrix with the truth value of each actor. But a property like box-independence considers interactions between all the actors, and so its truth table will be a $n \times n$ matrix with the truth value for box-independence between (box_i, box_j) for all $1 \leq i, j \leq n$.

For many of the above properties, it will be important to know specific details of violated properties (which box, which time frame, etc.); these details are being stored but we are currently only outputting binary truth values.

Our tool does not yet derive a directed multigraph from the traces; this will be done in the coming weeks. The tool currently detects a subset of the properties mentioned in the previous section, some of which are dependent on examining a multigraph.

5. APPLICATION TO REAL DATA

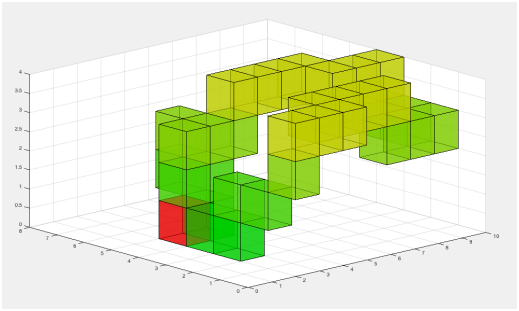

Fig. 8: Trace modeling property $\phi_{returnsHome}$

In this section, we apply our tool to cases of real data. Both examples utilize spatial traces gathered from a small flying robot (a quadrotor unmanned aerial vehicle, or UAV), flown inside a VICON motion capture room. The trace file is originally a ROS .bag file, and is processed as discussed in Sec. 4. Although not depicted in these visualizations, these paths are directed in ascending time.

The first example considers one actor $\bigstar_i$ and the property $\phi_{\{returnsHome\}}$. As shown in Fig. 8, this property is true for a trace T when an actor revisits the cube from which T begins. The UAV begins on the ground, flies up and across the space, translates in y, and then returns back to where it started from. The cube at which $\phi_{returnsHome}$ became true is highlighted in the figure. The unit cube size used in this example is $0.5m$, about the size of the UAV. We chose this size for the cube because at this size, if the 'fuzziness' property is 1 and there is no intersection, then they could not collide.

The second example involves two UAVs, that together model the property $\phi_{\{independence\}}$. This property is true of one or more traces T when they crossed each other's space during any interval of the traces. As shown in Fig. 9, one UAV's path is shown moving from bottom left to upper right, and a second UAV's path moves from right to left. The cube where $\phi_{\{independence\}}$ becomes false is highlighted in the figure.

Although these visualizations depict simple properties, they demonstrate that our tool correctly implements the technique and show the application to real data. Note also that $\phi_{\{independence\}}$ is equivalent to $\phi_{\{k-independence\}}$ for $k = 0$. Both visualization are generated in MATLAB from a simple structured text file generated by *Abstractus*.

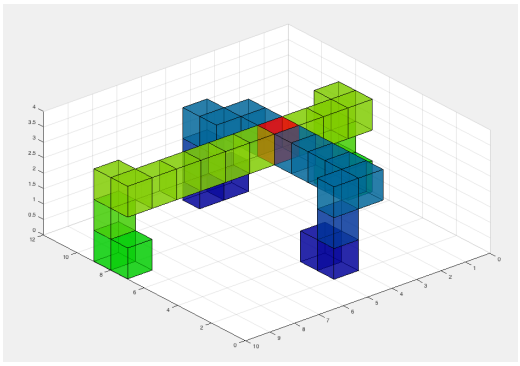

Fig. 9: Trace modeling property $\phi_{\{independence\}} = false$

6. RELATED WORK

Our work is inspired by a software invariant analysis technique developed in the Ph.D thesis of Ernst [Ernst 2000] that became the tool Daikon [Ernst et al. 2007]. Daikon takes as input a trace of variable values and then outputs properties ("invariants") about the values seen in that trace. Likewise, our method takes as input spatial traces and outputs properties observed during those traces. Unlike Daikon, which first instruments source code to generate traces, our method assumes that spatial traces are already generated by some logging mechanism of the system, in our case, ROS, the Robot Operating System [Quigley et al. 2009]. We are repurposing the ideas of Daikon, as has previously been done in hardware error detection [Sahoo et al. 2008].

In the robotics domain, using unit cubes to reason about spaces dates back at least as far as occupancy grids [Moravec and Elfes 1985]. Their unit space denotes the presence of obstacles detected by sonar, and this approach became a canonical in motion planning and obstacle avoidance [Elfes 1989; Borenstein and Koren 1989; Siegwart et al. 2011]. We likewise partition spaces into coarser, regular volumes with simple occupancy properties, but additionally, we embed a graph to track movement. Occupancy grids have recently been refined by the efficient OctoMap [Wurm et al. 2010]. In the future, We hope to leverage OctoMap in large spaces with multiple simultaneous scales. Unlike all known previous work with unit cubes, we reason about abstract properties of past movements rather than planning into the future.

We chose unit cubes because they are easy to work with and span $\mathbb{R}^3$. In 1885, Fedorov [Fedorov 1885] showed that there are exactly five ways to tessellate $\mathbb{R}^3$, and we examined the other shapes but decided that none offers advantages outweighing the ease of working with the unit cube. Further, only the unit cube can be extended to any higher dimension.

Blank *et al.* use video to trace the path of human movement in 'space-time shapes' and then identify actions based on the shapes [Blank et al. 2005]. We likewise consider the space occupied by an object as is moves through space, but whereas Blank *et al.* examine a highly-detailed image with individual moving parts as inferred by pixel changes, and the speeds of those parts, we consider an object to be a point in space occupying a box.

Another approach to spatial abstraction, not involving cube spaces, is explained by Shahar and Molina [Shahar and Molina 1998], who studied the properties of car traffic flow in a two-dimensional plane. They were interested in examining the properties of groups of cars in congestion by modeling the transition from fluid-like behavior to solid. Like their work, we coarsen space and examine properties of objects abstractly. Unlike their approach we work in unit space and search for properties of both individual actors as well as groups.

Gatalsky *et al.* applied space-time-cube visualizations to map a trace of movement over time to a three-dimensional cube [Gatalsky et al. 2004]. In their technique, time is mapped to a spatial dimension (z), and their main purpose is to quickly show the temporal relationships between actors. Like this work we are tracing movement over time, but unlike this work we reasoning about abstract properties and our main purpose is not visualization, but properties of movement.

7. OBSERVATIONS

In working in unit cube space, we encountered questions and peculiarities that we cannot fully explore in the scope of this work. Here we present briefly a few of the things we noticed.

The projection of a trace into cube space is sensitive to the original rotation of the chosen axes. So far, we have chosen real-world situations where the axes are aligned with our cube space, like in our examples with flying robots, where the co-ordinates of the motion capture system and the trajectories we programmed are axes aligned. This would not necessarily need to be the case. We suspect that box trace properties and cube ratios will change in proportion to the rotation angle, until a maximum divergence is reached $45°$s from the original axes.

Further rotation from $45 - 90°$s will decrease divergence until a rotational symmetry is found perpendicular to the original axes. This might mean that there are a number of 'best orientations' for the cube space relative the spatial trace, and that initially searching the space of possible orientations would find a better fit from which to find properties.

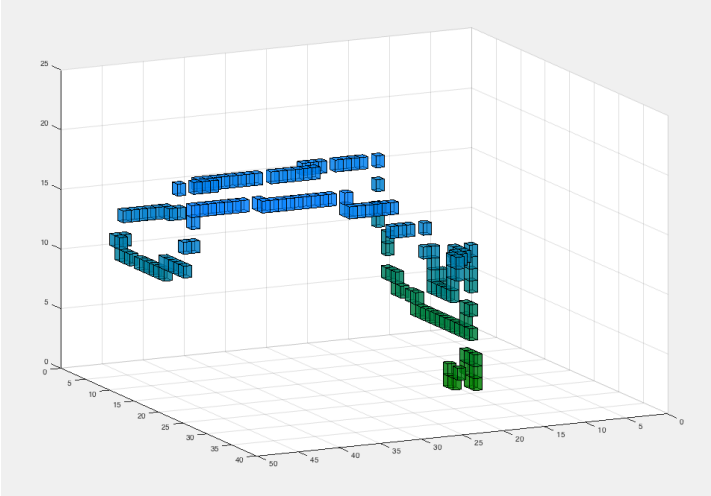

Fig. 10: Example of $10cm$ cubes where $\phi_{teleports}$ is true

Another consideration is choosing a good cube size. Different sizes may generate very different properties. If you choose *too small of a cube*, the trace may model $\phi_{\{teleport\}}$, as shown in Fig. 10, which we define as a spatially disconnected cube-trace. If there are gaps of disconnected cubes, and two entities collide in the unaccounted for space, the violation of cube-independence will not be inferred. If you choose *too big of a cube*, this may be too rough of an over approximation. For an extreme example, an entity never registers "leaving home" if its initial cube is the size of the given dimensions. Sometimes large over-approximations are desired, such as if you only want to examine the movements across two "hemispheres" of an entity's space.

We are curious about moments when trajectories loiter and meander slightly near the intersection of eight cubes. These are literally 'corner cases' where several or even many nodes would be added in the graph even though the movements are very small. It might be worth building in thresholds to ensure a spatial trace has 'really' moved to the next cube.

8. CONCLUSION

This work presents a novel technique for reasoning about movement *ex post facto*. We coarsen x, y, z position data to a unit cube space akin to occupancy grids. Within this space, we identify properties of movement for individuals and groups, as well as properties of paths. We implement this technique in a tool, *Abstractus*, and show examples of the tool's output as it identifies properties from real world position trace files. This work demonstrates the potential of this kind of spatial abstraction for finding useful properties of moving things.

In future work, we would like to explore properties of a host of actors (swarms), apply more efficient representations of cube space such as OctoMaps, and explore the extension of these techniques to software memory shapes.

ACKNOWLEDGMENTS

The authors would like to thank Dr. Matthew Dwyer and Dr. Sebastian Elbaum for facilitating insightful and scintillating conversations.

REFERENCES

Moshe Blank, Lena Gorelick, Eli Shechtman, Michal Irani, and Ronen Basri. 2005. Actions as space-time shapes. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, Vol. 2. IEEE, 1395–1402.

Johann Borenstein and Yoram Koren. 1989. Real-time obstacle avoidance for fast mobile robots. *Systems, Man and Cybernetics, IEEE Transactions on* 19, 5 (1989), 1179–1187.

Alberto Elfes. 1989. Using occupancy grids for mobile robot perception and navigation. *Computer* 22, 6 (1989), 46–57.

Michael D Ernst. 2000. *Dynamically discovering likely program invariants*. Ph.D. Dissertation. University of Washington.

Michael D Ernst, Jeff H Perkins, Philip J Guo, Stephen McCamant, Carlos Pacheco, Matthew S Tschantz, and Chen Xiao. 2007. The Daikon system for dynamic detection of likely invariants. *Science of Computer Programming* 69, 1 (2007), 35–45.

ES Fedorov. 1885. The elements of the study of figures.[Russian](2) 21. In *Zapiski Imperatorskogo S. Peterburgskogo Mineralogichesgo Obshchestva [Proc. S. Peterb. Mineral. Soc.]*. 1–289.

Peter Gatalsky, Natalia Andrienko, and Gennady Andrienko. 2004. Interactive analysis of event data using space-time cube. In *Information Visualisation, 2004. IV 2004. Proceedings. Eighth International Conference on*. IEEE, 145–152.

Yoshio Kataoka, David Notkin, Michael D Ernst, and William G Griswold. 2001. Automated support for program refactoring using invariants. In *Proceedings of the IEEE International Conference on Software Maintenance (ICSM'01)*. IEEE Computer Society, 736.

Hermann Minkowski. 1952. Space and time. *The Principle of Relativity. Dover Books on Physics. June 1, 1952. 240 pages. 0486600815, p. 73-91* 1 (1952), 73–91.

Hans P Moravec and Alberto Elfes. 1985. High resolution maps from wide angle sonar. In *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, Vol. 2. IEEE, 116–121.

Janice M Morse. 1993. Drowning in data. *Qualitative Health Research* 3, 3 (1993), 267–269.

Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. 2009. ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, Vol. 3. 5.

Swarup Kumar Sahoo, Man-Lap Li, Pradeep Ramachandran, Sarita V Adve, Vikram S Adve, and Yuanyuan Zhou. 2008. Using likely program invariants to detect hardware errors. In *Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on*. IEEE, 70–79.

Yuval Shahar and Martin Molina. 1998. Knowledge-based spatiotemporal linear abstraction. *Pattern Analysis and Applications* 1, 2 (1998), 91–104.

Roland Siegwart, Illah Reza Nourbakhsh, and Davide Scaramuzza. 2011. *Introduction to autonomous mobile robots*. MIT press.

Kai M Wurm, Armin Hornung, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. 2010. OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems. In *Proc. of the ICRA 2010 workshop on best practice in 3D perception and modeling for mobile manipulation*, Vol. 2.