Probabilistic Discovery of Articulated Object Kinematics Using Trajectory Matching with a pseudo-Riemannian Metric on $SE(3)$

Alex Burka[1] and Daniel D. Lee[2]
{aburka,ddlee}@seas.upenn.edu

Abstract— **We demonstrate a probabilistic approach for automatically discovering the latent kinematic structure of an articulated object from three-dimensional input (e.g. stereo vision, structured light, LIDAR or some other source of point cloud data conducive to object tracking). The key thrust is using standard function optimization algorithms to find the low-dimensional manifolds (nonlinearly embedded in $SE(3)$) which describe the pairwise relative motion of parts of an articulated object. We review the mathematical techniques necessary to develop this optimization, especially the distance metric on $SE(3)$ that is necessary for the objective function, and evaluate the resulting learner in simulation and using video captured using augmented reality (AR) markers for tracking.**

I. INTRODUCTION

A. Motivation

Consider the problem of navigating and interacting in unstructured, unknown environments. Actually, this is faced by office workers every morning. A person going to work may put on and take off several pairs of glasses. Operating a motor vehicle is another matter, but even on foot there are doors to open, some with keycards, others by turning keys; coffee and breakfast to buy; and the continuous allocation problem of accomplishing these tasks with only two hands. Once in the office, there remains a perception problem: locate a swivel chair, adapt to any furniture that was moved overnight. Described in human terms, this hypothetical morning seems simple. Yet we struggle to put a robot through these paces.

If we expect autonomous robots to accompany and assist us in our daily lives, they must be able to act in the human world. This includes many complex manipulation tasks. The sequence described above is completely unremarkable for a human, but involves serious heavy lifting and several open problems if it is to be completed by a robot. This work will focus on an important subproblem applicable to this situation: namely, extracting the kinematic structure of an articulated object from visual input. In the context of the current paper, an *articulated object* is a coherent object in the world that consists of rigid interlocking moving parts (for example, a tape measure, desk lamp, swivel chair, etc). The *kinematic structure* describes the relationships between the parts and how they move; for example, a swivel chair might be described by a tree with the seat at the root. The seat has two children: the backrest (which might be rigidly connected, or recline via a revolute joint), and the wheeled base, which turns on a revolute joint with respect to the seat. The seat might also be vertically adjustable, which would be represented by a prismatic joint in the same place. Note that, in general, kinematic structures are graphs, but in the following they are assumed to be trees to simplify the computation. The wheels of a swivel chair may turn independently, but usually they are all in contact with the floor and so move in concert. Therefore the chair has fewer degrees of freedom than is suggested by a kinematic tree with the wheels as leaves. However, in many cases the analysis is similar and with the tree assumption the structure discovery is much easier.

*We acknowledge the support of NSF and the ARL for this project.

[1]A. Burka is a graduate student with the Department of Electrical and Systems Engineering, University of Pennsylvania.

[2]D. D. Lee is a professor in the same department.

B. Literature Review

Interactive perception, and discovery of latent kinematic structure, is not an incredibly new idea. It has, however, become somewhat more tractable recently with the widespread availability of accurate RGBD cameras. Recent work includes the efforts of Dov Katz et al at UMass-Amherst [1], [2] and later at CMU [3], as well as Yan and Pollefeys at UNC [4] and Jürgen Sturm et al at the University of Freiburg [5].

Katz et al present a system for "interactive segmentation" as well as kinematic modeling in [1]. They use three-dimensional (RGBD) vision to track a large cloud of feature points on an articulated object of interest. Then, various metrics are used to group feature points into object parts, including 3D proximity, color and texture consistency, and relative 3D motion (that is, rigidly connected feature points will be close together, possibly the same color, and should have very little relative motion). Once the object of interest is segmented into rigid bodies, they aim to discover the underlying kinematic structure of those bodies. This is done using heuristics specific to each built-in joint type. Specifically, the relative 3D trajectories of feature points on a pair of prismatically connected rigid bodies should fit straight lines, and likewise arcs of circles for a revolute joint. These heuristics are somewhat limiting in that there is no unified system for deriving a heuristic from a kinematic definition. However, in a subsequent paper Katz et al explore the use of reinforcement learning to choose manipulator actions that probe the environment for the sole purpose of perception – an interesting direction that should be considered for this work as well [2].

Yan and Pollefeys take a more mathematical approach, attempting to discover the motion subspace described by each feature trajectory and linking them by looking for dependencies between the subspaces [4]. That work is principally applied to non-rigid objects, which puts it outside the scope of this paper, but the approach of considering relative motion as a subspace and representing the object as a graph is relevant.

Sturm et al provide the most direct inspiration for this work in [5]. They seek to create a top-to-bottom system that can go from visual input to a kinematic tree, taking into account mathematical joint models and remaining robust to outliers. Joint types considered are rigid, prismatic, revolute and a catch-all Gaussian process model that can capture several hard-to-model joints, such as a garage door.

C. Paper Outline

In the rest of this paper, we will present a system for building kinematic trees from visual input, implemented from scratch using MATLAB and following from the techniques of [5]. Section II introduces the mathematical foundations, and Section III follows with a schematic view of our implementation. We validate the system with several experiments in simulation and with real-world objects, detailed in Section IV. Finally, Section V discusses the many avenues for future research.

II. Kinematic and Probabilistic Models

A. Kinematics of Articulated Objects

In the context of this paper, an *articulated object* is a composite object consisting of one or more parts, which are connected by joints which define their relative motion. The joint between two parts might be rigid, in which case the two parts are effectively one. More interesting are joints with at least one degree of freedom. In this paper we consider only prismatic and revolute joints, which have one degree of freedom.

Mathematically, the relative motion between two parts of an articulated object can be described as a trajectory through $SE(3)$, the space of rigid motions in 3D space. This is a group with six degrees of freedom. However, the presence of a 1-DOF joint restricts the relative motion to a one-dimensional manifold nonlinearly embedded in the six-dimensional configuration space. The problem of finding the joint is thus rephrased as choosing the shape of the manifold (i.e. the joint type) and the nature of its embedding (the joint parameters).

Zooming out from the relative motion of two parts to the articulated object as a whole, we can describe its structure as a directed[1] graph, where the nodes are the parts of the object and the edges are annotated with joint types and parameters. An example graph describing the structure of a swivel chair is shown in Figure 1. In this simplified swivel chair model, the main parts are the seat, base, backrest and wheels. The wheels turn, the seat swivels and is adjustable in height, and the backrest leans back (P stands for prismatic and R for revolute). Note that, with the seat as the root, the swivel chair can be kinematically described as a tree. This greatly simplifies the joint fitting for reasons that will become clear later, so we will make the assumption in the following that all kinematic graphs are trees. This assumption is, of course, false; even in the swivel chair example the wheels are normally on the ground so their rotations are locked together. For another example, humans have two hands which are normally independent, but when gripping something like a steering wheel or a ladder rung they are kinematically connected. This further reduces the degrees of the system as a whole (thereby simplifying the search space), but it cannot be represented in a tree, so the tree-based algorithm presented here will miss the simplification.

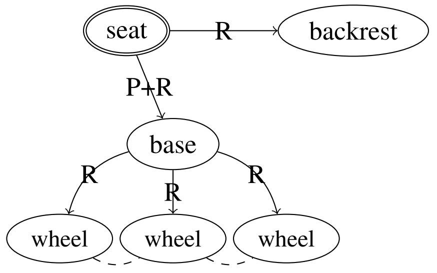

Fig. 1. A possible model for the kinematic structure of a swivel chair

B. Probabilistic Formulation of the Problem

To mathematically formulate the problem, we express the input data as a set of trajectories of K distinct points on an articulated object:[2]

$$X = \{\boldsymbol{x}_t^k \in SE(3) \mid k \in \{1..K\}, t \in \{1..T\}\} \tag{1}$$

The points could be feature points from camera data, or VICON tracking output, or GPS logs; the only requirement is that they are tracked in position and orientation. Envisioning the object as a graph $G = (V, E)$ where $V = \{1..K\}$, we want to find a set of edges

$$E = \{M^i = (J, \theta, \sigma)^i \mid i \in \{1..N\}\} \tag{2}$$

such that the graph is connected and the trajectory data is satisfactorily explained. Each of the N joints in the model is described by the above 3-tuple M^i where J identifies the joint type, while θ and σ describe the joint parameters and configuration (see Section II-D).

Let S be the set of all spanning trees of the connected graph of size K (which will all have $K-1$ edges), and let $\boldsymbol{\Delta}_t^{i:j} = (\boldsymbol{x}_t^j)^{-1}\boldsymbol{x}_t^i$ be the relative transformation between object parts i and j at time t. For a given model $\widehat{E}$ we want to evaluate its likelihood $P(\widehat{E} \mid X)$ but in general this is hard to calculate, so we invoke Bayes' rule

$$P(\widehat{E} \mid X) = \frac{P(X \mid \widehat{E})P(\widehat{E})}{P(X)} \tag{3}$$

[1]Note that the graph is directed in that reversing the direction of an edge requires inverting the geometry of the joint; however, the joint type does not change.

[2]In this paper, superscripts are used for numbering object parts or feature points, while subscripts are reserved for indexing in time. Where it makes sense, operations may be considered to be implicitly vectorized (i.e. $(x_{1..T}^a)^{-1}$ refers to $\{(x_1^a)^{-1}, (x_2^a)^{-1}, \dots\}$). Also, elements of $SE(3)$ will usually be shown in boldface ($\boldsymbol{x}$), matrices in $\mathbb{R}^{3\times3}$ in uppercase (M) and vectors in $\mathbb{R}^3$ with an arrow ($\vec{v}$).

which reformulates the problem in terms of $P(X \mid \widehat{E})$ (easier to calculate), $P(\widehat{E})$ (the model prior), and $P(X)$ (irrelevant for model selection). So we can select the best model using an argmax:

$$\begin{aligned}
\widehat{E} &= \max_{E} P(X \mid E) P(M) && (4)\\
&= \max_{E\in S} \prod_{i=1}^{K-1} \max_{M^i} P(X \mid M^i) P(M^i) && (5)\\
&= \max_{E\in S} \prod_{i=1}^{K-1} \max_{M^i} \prod_{t=1}^{T} P(\boldsymbol{\Delta}_t^{a^i:b^i} \mid M^i) P(M^i) && (6)\\
&= \max_{E\in S} \sum_{i=1}^{K-1} \max_{M^i} \sum_{t=1}^{T} \log P(\boldsymbol{\Delta}_t^{a^i:b^i} \mid M^i) + \log P(M^i) && (7)\\
&\approx \min_{E\in S} \sum_{i=1}^{K-1} \min_{M^i} \sum_{t=1}^{T} ||\boldsymbol{\Delta}_t^{a^i:b^i} - fk_{J^i}(\theta^i, \sigma_t^i)|| + |\theta^i| && (8)
\end{aligned}$$

As per usual in model selection over a large search space, we make several dubious independence assumptions in order to make the search tractable. In this case, we assume that (a) separate joints are independent (5), and (b) the likelihood of a given model is independent across time (6). Notice that maximizing likelihood is the same as maximizing log likelihood, so we can switch the products to sums without changing the outcome (7); lastly in (8) we switch from maximizing likelihood to minimizing the Akaike Information Criterion [6]. In this final form, the choice of $g \in G$ can be executed by building a complete graph of all K parts, weighting each edge with the AIC of the model best explaining the corresponding joint, and then finding the minimum spanning tree (see Figure 3). The definitions for the distance metric $||x - y||$ in this space and the forward kinematics $fk(\theta, \sigma)$ are given in sections II-C and II-D respectively.

C. A pseudo-Riemannian Distance Metric on $SE(3)$

A central problem in the mathematics of trajectory fitting is defining a distance function in trajectory space. Sturm did not discuss this problem in [5], but as implemented[3] that system uses a zero-mean, two-dimensional Gaussian distribution with diagonal covariance on the rotation angle and translation magnitude. That is, to find the "distance" $l_S(\boldsymbol{u}, \boldsymbol{v})$ between two $SE(3)$ transformations, first find the relative transformation $\boldsymbol{u}^{-1} * \boldsymbol{v}$. Any such transformation can be expressed as the composition of a rotation R and a translation $\vec{t}$, and taking the axis-angle view we can extract a single angle θ from the rotation, so that Sturm et al.'s metric is

$$l_S(\boldsymbol{u}, \boldsymbol{v}) = \frac{1}{2\pi\sigma_r\sigma_t} \exp\left\{-\frac{1}{2}\left(\frac{\theta^2}{\sigma_r^2} + \frac{||\vec{t}||^2}{\sigma_t^2}\right)\right\} \tag{9}$$

[3]see the functions `GenericModel::getInlierLogLikelihood` and `my_df` in `http://alufr-ros-pkg.googlecode.com/svn/trunk/articulation/articulation_models/src/models/generic_model.cpp`

with suitably defined variances σ_r and σ_t. In this section, we will derive a similar distance metric on $SE(3)$ from first principles.

$SE(3)$ is not a Euclidean space because three of its six degrees of freedom describe rotations, which are topologically distinct from translational dimensions. Therefore it is not so easy to find a natural distance metric on $SE(3)$. However, we need one, in order to evaluate the likelihood of a candidate joint model with regards to data (8). In this work we will use the scale-dependent left-invariant metric derived by Park [7], though other choices are available (e.g. Belta and Kumar [8], Larochelle et al [9]).

A distance metric must be symmetric, positive definite, and satisfy the triangle inequality. Additional desirable properties are scale-invariance ($||a-b||$ should not depend on the units of each dimension) and bi-invariance (left $||a-b|| = ||ca-cb||$ and right $||a-b|| = ||ac-bc||$). Unfortunately it can be shown [7] that in $SE(3)$ scale invariance is impossible, and we have to choose between left- and right-invariance. Luckily, however, $SE(3)$ is a Lie group, so we can use techniques from calculus to derive an acceptable metric. The main idea is, given two points $a, b \in SE(3)$, the distance is defined as a line integral along a geodesic from a to b (the shortest path, or an approximation thereto) *in the Lie algebra* $\mathfrak{se}(3)$. This is called a Riemannian metric. (Since the Lie algebra is comparable to a derivative, imagine measuring the length of an airplane's flight path by dividing it into many short segments and summing the lengths of each segment as if the Earth were piecewise flat.)

An element of $SE(3)$ may be considered as a 4×4 matrix containing a rotation and a translation

$$\boldsymbol{u} = \left[\begin{array}{c|c} u_R \in \mathbb{R}^{3\times3} & u_T \in \mathbb{R}^{3\times1} \\ \hline 0 & 1 \end{array}\right] \tag{10}$$

and its Lie algebra $\mathfrak{se}(3)$ has six-dimensional elements (ω, v) with a correspondence defined by the exponential mapping

$$\exp\left[\begin{array}{c|c} \omega_\times & v \\ \hline 0 & 0 \end{array}\right] = \left[\begin{array}{c|c} e^{\omega_\times} & Av \\ \hline 0 & 1 \end{array}\right] \tag{11}$$

where $\omega_\times$ is the skew-symmetric cross product matrix and A is defined thusly

$$\omega_\times = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \tag{12}$$

$$e^{\omega_\times} = I + \frac{\sin||\omega||}{||\omega||}\omega_\times + \frac{1-\cos||\omega||}{||\omega||^2}\omega_\times^2 \tag{13}$$

$$A = I + \frac{1-\cos||\omega||}{||\omega||^2}\omega_\times + \frac{||\omega|| - \sin||\omega||}{||\omega||^3}\omega_\times^2 \tag{14}$$

These formulae are from Park [7], and serve to show that the correspondence between $SE(3)$ and $\mathfrak{se}(3)$ is well-defined, but it will not be necessary to use them explicitly. The metric used in this work, which is scale-dependent and right-invariant, is

$$l(\boldsymbol{u}, \boldsymbol{v}) : SE(3) \times SE(3) \longrightarrow \mathbb{R} \tag{15}$$

$$l(\boldsymbol{u}, \boldsymbol{v}) = \sqrt{c||\log(U_R^T V_R)||_F^2 + d||\vec{u}_T - \vec{v}_T||^2}$$

where $||\cdot||_F$ denotes the Frobenius norm and $||\cdot||$ the familiar L^2 norm.

Since this is only to be used as an objective function for optimization, the quantity of interest is the squared distance

$$L(\boldsymbol{u}, \boldsymbol{v}) = c|| \log(U_R^T V_R)||_F^2 + d||\vec{u}_T - \vec{v}_T||^2 \tag{16}$$

The gradient, derived in Appendix A, is

$$\frac{\partial L}{\partial \boldsymbol{u}} = 2c \log(U_R^T V_R) V_R^T U_R V_R + 2d(\vec{u}_T - \vec{v}_T) \tag{17}$$

Since the general formula for converting a rotation matrix R to axis-angle form involves extracting the angle $\sqrt{2}\theta = || \log(R)||_F$, our metric (16) bears certain similarities to the logarithm of Sturm et al.'s metric (9). So using (16) as a stand-in for log likelihood is quite natural. Furthermore, having derived (16) from the geometry of the problem, the gradient comes naturally as well.

D. Joint Types

For the purposes of this work a *joint* defines an embedding of a low-dimensional manifold into $SE(3)$. A joint has a set of *parameters*, θ, which characterize the embedding, and *states*, σ, which form local coordinates in the manifold. In the current paper only rigid, prismatic and revolute joints are considered, but these are powerful enough to describe many real-world objects. Figure 2 shows the joints schematically.

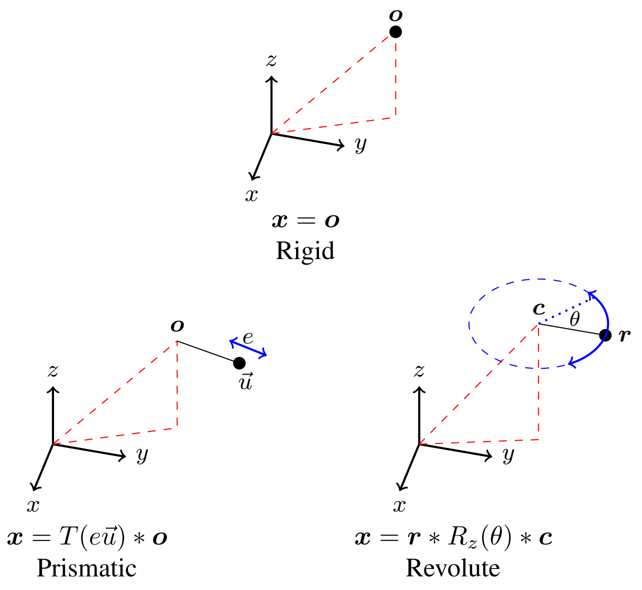
$\boldsymbol{o}$
z
y
x
$\boldsymbol{x} = \boldsymbol{o}$
Rigid
$\boldsymbol{o}$
e
$\vec{u}$
z
y
x
$\boldsymbol{x} = T(e\vec{u}) * \boldsymbol{o}$
Prismatic
$\boldsymbol{c}$
θ
$\boldsymbol{r}$
z
y
x
$\boldsymbol{x} = \boldsymbol{r} * R_z(\theta) * \boldsymbol{c}$
Revolute

Fig. 2. Joint types

To be useful in our system, a joint must have associated forward and inverse kinematics functions. A forward kinematics function fk_J processes θ and σ to produce a relative transformation in Euclidean space, while an inverse kinematics function ik_J takes such a relative transformation, along with θ, and recovers σ (often easier said than done, especially in the presence of sensor noise). The definitions of our three joint types are:

- **Rigid**: a rigid joint has one parameter, which is an offset in $SE(3)$, and no state. Acoordingly, its kinematics are trivial.

$$fk_{ri}(\theta, \sigma) : SE(3) \times \mathbb{R} \longrightarrow SE(3) \tag{18}$$
$$fk_{ri}(\langle \boldsymbol{o} \rangle, \sigma) = \boldsymbol{o}$$
$$ik_{ri}(\boldsymbol{x}, \theta) : SE(3) \times SE(3) \longrightarrow \mathbb{R} \tag{19}$$
$$ik_{ri}(\boldsymbol{x}, \langle \boldsymbol{o} \rangle) = 0$$

- **Prismatic**: a prismatic joint has two parameters, an offset and a unit vector pointing in the direction of extension, and one state which specifies the length of extension. In the following T creates a pure translation in $SE(3)$, given a vector in $\mathbb{R}^3$, and T^{-1} extracts the translation vector from a full transformation.

$$fk_{pr}(\theta, \sigma) : SE(3) \times \mathbb{U}^3 \times \mathbb{R} \longrightarrow SE(3) \tag{20}$$
$$fk_{pr}(\langle \boldsymbol{o}, \vec{u} \rangle, \sigma) = T(\sigma \vec{u}) * \boldsymbol{o}$$
$$ik_{pr}(\boldsymbol{x}, \theta) : SE(3) \times SE(3) \times \mathbb{U}^3 \longrightarrow \mathbb{R} \tag{21}$$
$$ik_{pr}(\boldsymbol{x}, \langle \boldsymbol{o}, \vec{u} \rangle) = T^{-1}(\boldsymbol{x} * \boldsymbol{o}^{-1}) \cdot \vec{u}$$

- **Revolute**: a revolute joint has two parameters, a center of rotation and a "radius" which defines a transformation to apply after the rotation, and one state which specifies the rotation angle. In the following R_z creates a pure rotation around the z axis in $SE(3)$, given an angle, and R_z^{-1} recovers that angle.

$$fk_{re}(\theta, \sigma) : SE(3) \times SE(3) \times \mathbb{R} \longrightarrow SE(3) \tag{22}$$
$$fk_{re}(\langle \boldsymbol{c}, \boldsymbol{r} \rangle, \sigma) = \boldsymbol{r} * R_z(\sigma) * \boldsymbol{c}$$
$$ik_{re}(\boldsymbol{x}, \theta) : SE(3) \times SE(3) \times SE(3) \longrightarrow \mathbb{R} \tag{23}$$
$$ik_{re}(\boldsymbol{x}, \langle \boldsymbol{c}, \boldsymbol{r} \rangle) = R_z^{-1}(\boldsymbol{r}^{-1} * \boldsymbol{x} * \boldsymbol{c}^{-1})$$

In the future, it would make sense to add more joint types so that our kinematic trees can be more expressive. For example, ball joints (which have three states) are often seen in the wild, as are screw joints. Note that these could also be represented by combining prismatic and revolute joints with "virtual" (invisible) nodes in the tree.

III. IMPLEMENTATION

Currently the algorithm presented here is implemented in MATLAB. The code will be available online.[4]

A. Algorithm Overview

An overview of the major stages in the learning algorithm is presented in Figure 3. The procedure is designed to be modular, with most components interchangeable – for example, the perception could use a Kinect or a laser scanner, the smoothing could use a Kalman filter instead of the current generic low-pass, et cetera.

Algorithm 1 corresponds to the dotted box in the flowchart, which is the core of the learning. Here, the initial parameters $\hat{\theta}$ are computed differently for each joint type:

- First we choose three (different) time points in the trajectory from which to initialize the parameters.

[4]`http://www.alexburka.com/penn/manip.html`

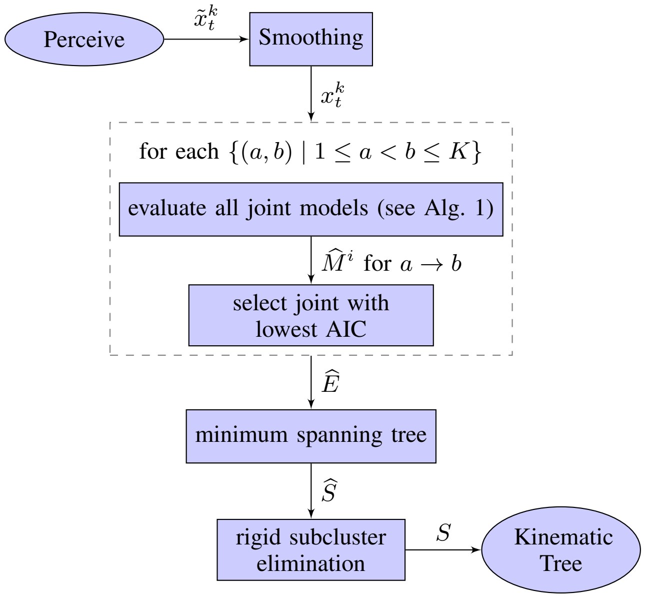
Perceive
$\tilde{x}_t^k$
Smoothing
x_t^k
for each $\{(a,b) \mid 1 \leq a < b \leq K\}$
evaluate all joint models (see Alg. 1)
$\widehat{M}^i$ for $a \to b$
select joint with lowest AIC
$\widehat{E}$
minimum spanning tree
$\widehat{S}$
rigid subcluster elimination
S
Kinematic Tree

Fig. 3. Block diagram of the optimization algorithm

- For a rigid joint, we simply set the offset to one of the transformations (which are all approximately the same, if this is really a rigid joint).
- For a prismatic joint, we set the offset to one of the transformations and calculate the unit vector as the direction of translation from one transformation to another.
- For a revolute joint, the calculation is more involved. First we fit the plane of rotation, from the entire trajectory (the plane is defined by a center point, the mean of the trajectory translations, and two basis vectors which come from the two largest principal components of the trajectory translations). Then we find the circumcenter of the three sample transformations, restricted to this plane; that is the translation component of $\boldsymbol{c}$. The rotation component of $\boldsymbol{c}$ is constructed from the basis vectors of the plane P_{d1} and P_{d2}. Lastly, $\boldsymbol{r}$ is simple to extract from $\boldsymbol{c}$ and one sample transformation.

The cost function is AIC, the Akaike Information Criterion, which is chosen to penalize models with too many degrees of freedom [6]. The full formula is

$$AIC = -2 \log \mathcal{L}_{max} + 2k \tag{24}$$

where $\mathcal{L}_{max}$ is the maximum likelihood and k is the number of model DOFs. To calculate AIC we use the error calculated by the objective function (line 19 of Algorithm 1) as a measure of the negative log likelihood, and add the number of model parameters.[5]

[5]In Section II-D, elements of $SE(3)$ and $\mathbb{R}^3$ were counted as whole parameters. Here we count degrees of freedom instead. An element of $SE(3)$ has six degrees of freedom, and an element of $\mathbb{R}^3$ three. Using this counting method, a rigid joint has 6 DOFs, a prismatic joint has 8 DOFs, and a revolute joint has 12 DOFs. In truth, for implementation expedience the prismatic joint is considered to have 9 DOFs (ignoring the unit vector constraint for AIC purposes), but prismatic joints are frequent false positives anyway, so this should not adversely affect the results.

After the loop, which finds the most plausible joint model for each pair of object parts, we still need to decide which parts are actually connected and which joints are superfluous or misleading. The minimum spanning tree fulfills this step, choosing the joints with the lowest cost while keeping all of the nodes connected.

The last step is *rigid subcluster elimination*, which eliminates all rigid joints from the tree, by modifying the surrounding joints to obviate the rigid offset. Table I shows the simple parameter adjustments necessary to absorb a rigid offset $\boldsymbol{q} = (Q_R, \vec{q}_T)$ on either end of another joint. The reason for including rigid subcluster elimination is for the case in which the input is from an unsupervised feature tracker, where there will be multiple features tracker per object part. The final tree thus contains only the movable joints.

Joint type	Parameters	Prepend rigid $\boldsymbol{q}$	Append rigid $\boldsymbol{q}$
Rigid	$\boldsymbol{o}$	$\boldsymbol{o} \leftarrow \boldsymbol{o} * \boldsymbol{q}$	$\boldsymbol{o} \leftarrow \boldsymbol{q} * \boldsymbol{o}$
Prismatic	$\boldsymbol{o}, \vec{u}$	$\boldsymbol{o} \leftarrow \boldsymbol{o} * \boldsymbol{q}$	$\boldsymbol{o} \leftarrow \boldsymbol{q} * \boldsymbol{o}$, $\vec{u} \leftarrow Q_R * \vec{u}$
Revolute	$\boldsymbol{c}, \boldsymbol{r}$	$\boldsymbol{c} \leftarrow \boldsymbol{c} * \boldsymbol{q}$	$\boldsymbol{r} \leftarrow \boldsymbol{q} * \boldsymbol{r}$

TABLE I

INCORPORATING RIGID OFFSETS INTO JOINTS

Algorithm 1: Joint learner

```
   Input: $a, b$ tree edge, $\boldsymbol{x}^{a,b}_{1..T}$ object part positions
   Output: Joint $(J, \theta)$, states $\sigma_{1..T}$, cost $c$
1  $\boldsymbol{\Delta}_{1..T} \leftarrow \boldsymbol{x}^b_{1..T} * (\boldsymbol{x}^a_{1..T})^{-1}$
2  for $q$ in $\{$rigid, prismatic, revolute$\}$ do
3      $i, j, k \leftarrow \textsc{Rnd}(1..T)$
4      switch $q$ do
5          case rigid
6              $\hat{\theta} \leftarrow \{\boldsymbol{\Delta}_i\}$
7          endsw
8          case prismatic
9              $\hat{\theta} \leftarrow \{\boldsymbol{\Delta}_i, \frac{T^{-1}(\boldsymbol{\Delta}_k - \boldsymbol{\Delta}_i)}{||T^{-1}(\boldsymbol{\Delta}_k - \boldsymbol{\Delta}_i)||}\}$
10         endsw
11         case revolute
12             $P \leftarrow \textsc{FitPlane}(T^{-1}(\boldsymbol{\Delta}_{1..T}))$
13             $\vec{\delta}_i, \vec{\delta}_j, \vec{\delta}_k \leftarrow \textsc{Proj}(P,\ \boldsymbol{\Delta}_{\{i,j,k\}})$
14             $\vec{c} \leftarrow \textsc{Unproj}(P,\ \textsc{Circumcenter}(\vec{\delta}_{\{i,j,k\}}))$
15             $\boldsymbol{c} \leftarrow T(\vec{c}) * \begin{bmatrix} \vec{P}_{d1} \\ \hline \vec{P}_{d2} \\ \hline \vec{P}_{d1} \times \vec{P}_{d2} \end{bmatrix}$
16             $\hat{\theta} \leftarrow \{\boldsymbol{c}, \boldsymbol{\Delta}_i * \boldsymbol{c}^{-1}\}$
17         endsw
18     endsw
19     $(\theta_q, c_q) \leftarrow \textsc{Min}(\hat{\theta},\ \lambda\theta.\ ||fk_q(\theta, ik_q(\boldsymbol{\Delta}, \theta)) - \boldsymbol{\Delta}||^2)$
20 end for
21 $J \leftarrow \min_q c_q$
22 $\sigma_{1..T} \leftarrow ik_J(\boldsymbol{\Delta}_{1..T}, \theta_J)$
23 return $(J, \theta_J), \sigma_{1..T}, c_J$
```

IV. EVALUATION

A. *Simulation*

Software was written to simulate the motion of arbitrary articulated objects in 2D or 3D. A graphical interface is provided for constructing a kinematic tree, which is represented as a list of tuples

$$S = \{(a, b, J, \theta, \sigma, \sigma_{min}, \sigma_{max})^j \mid j \in \{1..N\}\} \tag{25}$$

In this tuple, similar to that of (2), a and b specify the object parts (graph nodes) connected by the joint, J is the joint type configured by θ, and σ, the joint state, varies between σ_{min} to σ_{max} during the simulation.

The algorithm for generating rigid motions of the full articulated object is simple: the root of the kinematic tree is placed at a random position and perturbed a small amount at every timestep. All the other object parts are placed by walking the tree (depth-first, using the recursive PLACEOBJECTS function) and chaining the joint forward kinematics functions. The parameters σ are perturbed so that the joints move, and a small amount of noise is also simulated.

Algorithm 2: Articulated motion simulator

```
    Input: $S$ kinematic tree (25), $(\epsilon_T, \epsilon_R)$ noise amplitude
    Output: $\boldsymbol{x}^{1..K}_{1..T}$ object part trajectories
 1  $\boldsymbol{x}^1_1 \leftarrow$ T(RND($\pm 5$)) $*$ R(RND($\pm 2\pi$))
 2  for $t \leftarrow 1$ to $T$ do
 3      $\boldsymbol{x}^{1..K}_t \leftarrow$ PLACEOBJECTS($\boldsymbol{x}^{1..K}_t$, $S$, 1)
 4      $\boldsymbol{\Delta x} \leftarrow$ T(RND($\pm\epsilon_T$)) $*$ R(RND($\pm\epsilon_R$))
 5      $\boldsymbol{x}^{1..K}_t \leftarrow \boldsymbol{\Delta x} * \boldsymbol{x}^{1..K}_t$
 6      $\boldsymbol{x}^1_{t+1} \leftarrow \boldsymbol{x}^1_t * \boldsymbol{\Delta x}$
 7      for $j \leftarrow 1$ to $N$ do
 8          $\Delta\sigma \leftarrow$ RND($\pm\frac{1}{10}(\sigma^j_{max} - \sigma^j_{min})$)
 9          $\sigma^j \leftarrow$ BOUND($\sigma^j + \Delta\sigma$, $\sigma^j_{min}$, $\sigma^j_{max}$)
10      end for
11  end for
```

Function PLACEOBJECTS (subroutine for Alg. 2)

```
    Input: $\boldsymbol{x}^{1..K}_t$ object part positions, $S$ kinematic tree
        (25), $p$ parent part index
    Output: $\boldsymbol{x}^{1..K}_t$ updated part positions
 1  for $j \leftarrow 1$ to $N$ do
 2      if $a^j = p$ then
 3          $\boldsymbol{x}^{b^j}_t \leftarrow fk_{J^j}(\theta^j, \sigma^j) * \boldsymbol{x}^{a^j}_t$
 4          $\boldsymbol{x}^{1..K}_t \leftarrow$ PLACEOBJECTS($\boldsymbol{x}^{1..K}_t$, $S$, $b^j$)
 5      end if
 6  end for
 7  return $\boldsymbol{x}^{1..K}_t$
```

B. *Input from Augmented Reality Markers*

In order to leave the simulation and use the tree learner on real data, we need a set of feature points to track. In the simulator, this comes for free. In order to postpone the hard problem of image segmentation and coherent feature tracking, we instrument the articulated objects under study with fiducial markers, specifically Aruco markers, which are small QR-like patterns that encode 5 bit numbers [10]. An external detection library locates these markers in an image and outputs position and orientation. (See Figure 4 for an example.) It is a simple matter to translate this into our standard trajectory format (i.e., a sequence of homogeneous transformation matrices) and so the initial perception problem is effectively circumvented (but see Section V for ideas to improve this situation).

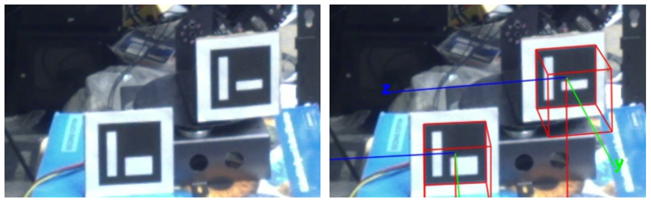

Fig. 4. Detection of Aruco fiducial markers

C. *Experiment 1: Simulation*

1) Setup: For this experiment we drew several kinematic trees representing imaginary objects and fabricated trajectory datasets by generating random trajectories in state space. The trees are not supposed to represent any real-world objects, but rather to exercise the capabilities of the joint matching algorithm. The simulator has two interesting hyperparameters:

1) Noise is applied to the generated trajectory data. The noise is uniformly distributed in the Lie algebra $\mathfrak{se}(3)$, with different magnitudes on the translational and rotational components. Those magnitudes are (ϵ_T, ϵ_R), as in line 4 of Algorithm 2.
2) Normally the simulation generates one trajectory per object part, which replicates the input using the fiducial markers. However, with unsupervised feature tracking there will be multiple (rigidly-connected) trajectories per object part. The simulator can fake this using an "inflation" factor I: before simulation, we add KI rigid joints connecting each node to I nearby virtual nodes.

2) Results: Figure 5 shows the effect of changing the simulator noise parameter on matching performance. The experiment was run 4 times at each of 4 noise settings and 4 inflation settings; results are shown with no inflation. At the top of the figure is the tree used to generate all the simulations. The plots show the algorithm attempting to model the prismatic joint, at three noise levels. With low noise, this is successful; at high noise levels the fit incorrectly produces three revolute joints.

Real-world trajectory measurements will always be noisy, especially when recovering 3D pose from 2D cameras, so robustness to noise in simulation is promising. The assumption that noise will be uniformly distributed in $\mathfrak{se}(3)$ is perhaps a naïve one, so future work should vary this distribution.

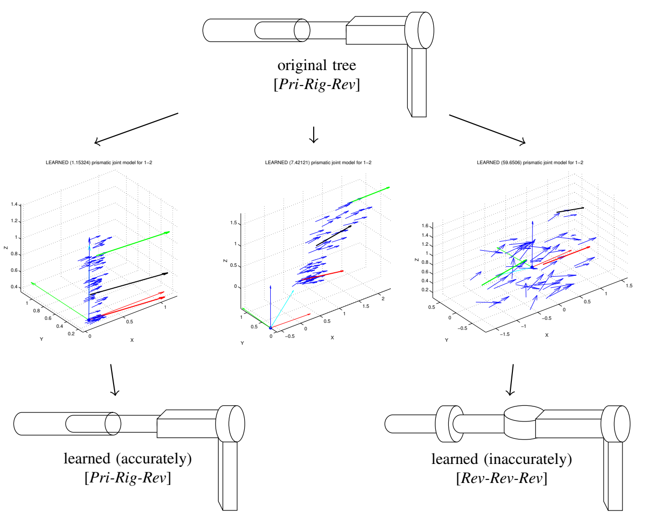

Fig. 5. Simulation robustness to noise. Visualizations show the best-fitting attempts to model the leftmost joint as prismatic, at different noise levels.

D. Experiment 2: Objects in the lab

1) Setup: In the first experiment, the learner was applied to simple articulated objects found in the robotics lab: a swing arm desk lamp and a Dynamixel robotic arm (shown in Figure 6). The lamp has three revolute joints and a two-bar linkage which does not correspond to any of the models in the current system (in fact it has two degrees of freedom), but for certain motions it appears as a prismatic joint which can be detected. The robotic arm has four revolute joints (and a prismatic joint in the gripper which we are not using).

For the experiment, the lamp and robotic arm were both actuated by hand to capture two datasets each. For a final dataset, the robotic arm was driven by a computer through a set of scripted motions; accordingly we can construct a ground truth for comparison. The objects were instrumented with Aruco markers (approximaely 2 inches in width) at the points marked in red on the figure. All videos were recorded with the built-in webcam on a 2012 Apple MacBook Air, using OpenCV, and are available online.

Once parsed by the Aruco tracking library, we have a set of markers identified in each frame, and coordinates $\langle t_x, t_y, t_z, r_x, r_y, r_z \rangle^{1..K}$ for each one (frames where one or more markers go undetected are simply discarded). Since the library unfortunately uses a different version of Euler angles than the rest of this work, we perform the transformation

$$R^k = R_z(\frac{\pi}{2})R_y(r_x^k)R_z(-\frac{\pi}{2})R_y(r_y^k)R_z(r_z^k) \tag{26}$$

The Aruco output is rather noisy, especially the rotation components, so we smooth it with a low-pass filter. After that, the trajectories are processed by the matching algorithm as in Figure 3.

2) Results: Figure 7 shows a trace through time of the commands sent to the base joint of the robotic arm, and the angle detected for that joint (note that of course the commands are sent instantaneously, but the arm speed is heavily limited so that the camera does not lose track of the Aruco tags). We see that the detection works fairly well as the base joint rotates around the z axis. At high angles, the Aruco tags are at an oblique angle with respect to the camera, so there is not much resolution and the detection accuracy is reduced.

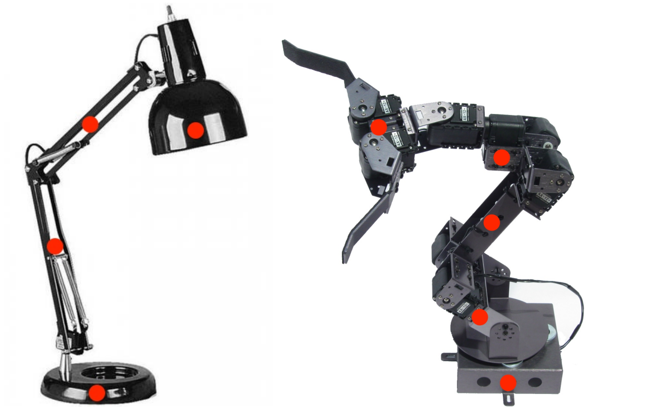

Fig. 6. Representative images of the objects used for Experiment 2. Fiducial markers placed at red dots. Sources: Home Decorators Collection and CrustCrawler Robotics

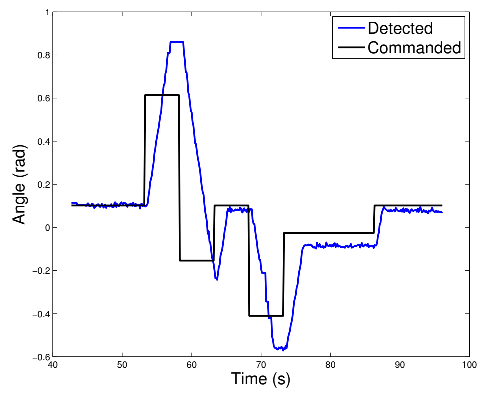

Fig. 7. Commanded and detected angles for one joint of the robotic arm in Experiment 2.

V. CONCLUSIONS AND FUTURE WORK

In the current paper we have presented a working system that takes in raw trajectory input and produces a kinematic tree as output. The system's effectiveness is demonstrated in simulation and on real-world objects using fiducial markers for tracking, though we envision swapping out the fiducial markers for 3D feature tracking with a stereoscopic camera in the near future. The key contribution of this work is a modular framework for kinematic tree fitting applications, since nearly every stage in Figure 3 can be replaced with alternative implementations to try different joint matching strategies; additionally, adding another joint type requires implementing only forward/inverse kinematics and a few ancillary functions.

There are many avenues for continuing research with this modular platform as a base.

One area that invites inquiry is better object tracking: that using a stereoscopic camera we can achieve real-time position and orientation tracking without using fiducial markers. Given a stationary camera, motion can be used directly for segmentation.

Another avenue is to re-examine the feedforward nature of Figure 3. Other popular vision algorithms, such as SLAM, use probabilistic structures to keep track of a belief state about the world, which in turn informs the interpretation of new sensor data. We could even incorporate a structure akin to a particle filter in order to track multiple kinematic tree hypotheses [11].

Looking further ahead, it is somewhat limiting to confine the modeling effort to software. Humans and animals routinely modify the environment to test hypotheses and ease perception. Robots may nudge, squeeze, pick up or even throw and drop objects in order to learn about their kinematic and dynamic properties. The "Ripley" robotic arm system described in [12] performs these exploratory manipulations. Applied to articulated objects, this kind of "interactive perception", where the current belief state of the kinematic tree determines the most informative manipulator action, could be very helpful.

In conclusion, the system presented here is modular and extensible, and we see many interesting directions for extending its functionality.

APPENDIX

A. Gradient of the Objective Function

If we re-cast (16) as a chain of nested functions

$$L(\boldsymbol{u},\boldsymbol{v}) = c * g(h(k(\boldsymbol{u},\boldsymbol{v}))) + d * m(n(\boldsymbol{u},\boldsymbol{v})) \tag{27}$$

where the individual functions are

$$\begin{aligned} g(x) &= ||x||_F^2 & \mathbb{R}^{3\times3} &\to \mathbb{R} \\ h(x) &= \log x & \mathbb{R}^{3\times3} &\to \mathbb{R}^{3\times3} \\ k(\boldsymbol{u},\boldsymbol{v}) &= U_R^T V_R & SE(3) &\to \mathbb{R}^{3\times3} \\ m(x) &= ||x||^2 & \mathbb{R}^3 &\to \mathbb{R} \\ n(\boldsymbol{u},\boldsymbol{v}) &= \vec{u}_T - \vec{v}_T & SE(3) &\to \mathbb{R}^3 \end{aligned}$$

the gradient decomposes as a combination of Jacobian matrices:

$$\frac{\partial L}{\partial \boldsymbol{u}} = c\frac{\partial g}{\partial x}\frac{\partial h}{\partial x}\frac{\partial k}{\partial \boldsymbol{u}} + d\frac{\partial m}{\partial x}\frac{\partial n}{\partial \boldsymbol{u}} \tag{28}$$

Computing the Jacobians and calculating as above,

$$\begin{aligned} \frac{\partial g}{\partial x} &= 2x & 1\times9 \qquad \frac{\partial h}{\partial x} &= x^{-1} & 9\times9 \\ \frac{\partial k}{\partial \boldsymbol{u}} &= v_R & 9\times16 \qquad \frac{\partial m}{\partial x} &= 2x & 1\times3 \\ \frac{\partial n}{\partial \boldsymbol{u}} &= 1 & 3\times16 \end{aligned}$$

$$\begin{aligned} \frac{\partial L}{\partial \boldsymbol{u}} &= c * 2h(k(\boldsymbol{u},\boldsymbol{v})) * k(\boldsymbol{u},\boldsymbol{v})^{-1} * V_R + d * 2n(\boldsymbol{u},\boldsymbol{v}) \\ &= 2c\log(U_R^T V_R)V_R^T U_R V_R + 2d(\vec{u}_T - \vec{v}_T) \end{aligned} \tag{29}$$

and this will be used to speed up the gradient descent optimization (see Algorithm 1).

B. Source Code

The MATLAB, Python and C++ code underlying the algorithms described in this paper may be found at `http://www.alexburka.com/penn/manip.html`.

ACKNOWLEDGMENTS

Thanks are due to Prof. Thomas Hunter (Swarthmore College) for a crash course in Lie group theory.

REFERENCES

[1] D. Katz and O. Brock, "Manipulating articulated objects with interactive perception," in *2008 IEEE International Conference on Robotics and Automation*. IEEE, May 2008, pp. 272–277. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4543220http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4543220

[2] D. Katz, Y. Pyuro, and O. Brock, "Learning to manipulate articulated objects in unstructured environments using a grounded relational representation," *In Robotics: Science and Systems*, 2008. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.177.9616

[3] D. Katz, M. Kazemi, J. Bagnell, and A. Stentz, "Interactive Segmentation, Tracking, and Kinematic Modeling of Unknown Articulated Objects," 2012. [Online]. Available: http://www.ri.cmu.edu/pub_files/2012/3/InteractiveSegmentation.pdf

[4] J. Yan and M. Pollefeys, "Automatic kinematic chain building from feature trajectories of articulated objects," …*Vision and Pattern Recognition, 2006 IEEE* …, 2006. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1640824

[5] J. Sturm, C. Stachniss, and W. Burgard, "A probabilistic framework for learning kinematic models of articulated objects," *Journal of Artificial Intelligence Research*, vol. 41, pp. 477–526, 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=2051252

[6] A. R. Liddle, "Information criteria for astrophysical model selection," 2008.

[7] F. Park, "Distance Metrics on the Rigid-Body Motions with Applications to Mechanism Design," *Journal of Mechanical Design*, no. March, 1995. [Online]. Available: http://cat.inist.fr/?aModele=afficheN\&cpsidt=3501402

[8] C. Belta and V. Kumar, "Euclidean metrics for motion generation on SE (3)," *Proceedings of the Institution of Mechanical* …, no. 3, 2002. [Online]. Available: http://pep.metapress.com/index/778684U55U713X10.pdf

[9] P. M. Larochelle, A. P. Murray, and J. Angeles, "A Distance Metric for Finite Sets of Rigid-Body Displacements via the Polar Decomposition," *Journal of Mechanical Design*, vol. 129, no. 8, p. 883, 2007. [Online]. Available: http://link.aip.org/link/JMDEDB/v129/i8/p883/s1\&Agg=doi

[10] R. Muñoz Salinas, "ArUco: A minimal library for Augmented Reality applications based on OpenCV," 2012.

[11] S. Thrun, "Robotic Mapping : A Survey," no. February, 2002.

[12] K.-y. Hsiao and D. Roy, "A Habit System for an Interactive Robot Ripley : An Interactive Manipulator Robot," 2005.