CS 6758: Finding and Classifying Object Interaction Using Stereoscopic Data

Shahrukh Mallick, CS M.Eng 2011, skm47@cornell.edu

Andrew Spielberg, CS M.Eng 2011, aes89@cornell.edu

Abstract

Autonomous understanding of the kinematic structure and interaction of objects in an agent's environment has important applications in the fields of robotics and computational physics. In this paper, we present a novel means for determining the kinematic model of objects (rigid and non-rigid) and how they interact with one another in an environment using stereoscopic data of the agent's world. Our algorithm consists of two phases. The first is an unsupervised link detection phase, wherein we determine the set of links that compose a kinematic body. Here, we employ 2D tracking algorithms coupled with 3D spectral clustering methods. The second phase is to determine where and how these links interact with one another. The locations are determined using assumptions about the locality of the links. We employ a soft max regression on our features, which were determined heuristically to discern the interaction types. Clustering was correct on data in 22 out of 36 cases, yielding 61.1% accuracy.

1 Introduction

The task of identifying objects and understanding how to utilize these objects is a growing area of interest in the fields of machine learning and robotics. Specifically understanding the kinematic model of an object will illuminate the functionality and uses of the object (e.g. a robot can learn how scissors move without any prior knowledge about the structure of the scissors, and can use this knowledge to cut paper). This extra layer of info will provide more valuable information to a computer. This is a very simple idea, and this type of learning can be applied to a wide scope of tasks and objectives.

Several other studies have addressed this problem. Oliver *et al* [1] suggest a method for tracking motion of linked bodies over time and determining the planar kinematic structure. While it is generalizable to n-linked objects, it has two main shortcomings. One, it requires that the object's motion be projectable onto a 2D plane (and manipulated on the plane), and two, it only can identify rotational motion (revolute). J. Sturm [2] used artificial markers and learned the articulation models of objects in 3D, focusing on rigid bodies and classifying the model into rigid transformations or into a joint model consisting of a prismatic, rotational, or an LLE/GP model. Sturm expanded on this work [3] using stereovision to learn the articulation of models without any markers. However, this implementation is limited in that it only addresses objects that can be fit by rectangles.

In this paper, we will expand on the previously mentioned works by identifying where objects interact with each other, and classify them into one of several different types. We will address deformable objects as well. We will use 3D stereographic data to develop the methods for determining the kinematic structure of the objects. In addition, no markers will be used to aid in tracking, as this is

important when considering applications to the real world. Our goal is to develop an algorithm that is reasonably fast and highly accurate.

2 Dataset and Methodology

In developing our dataset, we sought to provide data on a variety of interactions and movements in the objects and environments, including prismatic, revolute, screw, rigid contact, 2D planar constraint, 1-manifold deformable, 2-manifold deformable, and separable. Most items we considered contained one core interaction, but most cases contained several different types of interactions. Table 1 below provides a summary of the interaction types, a brief description of it, and the number of videos recorded of this type.

Interaction Type	Description	# of Videos
Prismatic	Two links related by a prismatic joint	4
Rotational (Revolute)	Two links related by a revolute joint	5
Rotational (Screw)	Two links related by a screw joint	2
Rigid Constraint	Links constrained by a point of contact (i.e. hangar on hook)	6
2D-Planar Constraint	An item constrained to some plane (i.e. item in card sleeve)	2
1-Manifold	1D deformable object (i.e. wire)	2
2-Manifold	2D deformable object (i.e. cloth)	8
Separable	No constraint between objects (i.e. item on surface)	7

Table 1: Overview of data set: interaction type, description of type, and # of videos for each item shown

For a very detailed view of the data set, a table has been provided in the appendix that specifies the contents of each video.

Recording of data was done with the Microsoft Kinect, which captures a 3D point cloud of the object and provides a corresponding image. The point cloud data contains the x, y, and z coordinates of points on non-background surfaces that the camera was able to see, and its correspondence with the 2D image. We simulated movement through stop motion, since the Kinect cannot collect store data in real time. Our stop motion "videos" per item were kept to a standard of approximately 15 frames.

3 Algorithm

Here we outline the algorithms for detecting areas of interaction and classifying them into types, both of which build upon a link segmentation algorithm which we implement. We first discuss the link segmentation algorithm before discussing the detection and classification algorithms.

3.1 Link Segmentation

In order to determine any information of the kinematic structure of a given system, it is first imperative that we determine what "links" the objects possess, that is, their kinematic structure. To do this, we require three steps – image segmentation, feature generation, and feature clustering. The former two happen during each time step, but the clustering only occurs after the algorithm has observed the entire video. We now consider each of these steps in turn.

3.1.1 Image Segmentation

The principal technique used for the feature generation step is feature tracking, which is very sensitive to abrupt variation in color intensities. We define the foreground as the structure we wish to learn about, and everything else as the background. Image segmentation of the scene was necessary for reducing the amount of noise in the system. Here, we define noise to be details of the structure that provide no information about system structure or background imagery. For instance, dirt on the camera lens, polka dots on a texture, or hair on a human forearm, none of which provide any information about the structure of the object, can add abrupt changes in color intensity to the system, which in turn can provide misleading information to the tracking algorithms. Additionally, background imagery, such as a shadow, which is not part of the object, can waste time and memory by causing our algorithms to track irrelevant parts of the image, or perhaps even mistake a point as moving from the foreground structure to the background. Image segmentation can eliminate localized but abrupt, noisy details, and further weaken the already gentle background gradients, thus reducing these sources of noise.

We implemented a mean-shift segmentation, as it was a relatively fast method which removed most of the noise in our videos; we used the openCV [4] implementation with a spatial radius of 20 pixels and a color radius of 5 in standard RGB color space.

However, there is a draw back with the use of image segmentation. Because we were using distinct features to track our corners (discussed in 3.1.2), image segmentation would smooth out these features and make the tracking not perform well. For this reason, we opted to not perform segmentation as it was more critical to assure the tracking was working correctly. This would insure better clustering later on. There likely is a happy medium that can be reached (very little segmenting to get rid of noise without losing all sharp detail in object). However, this study did not delve into this area.

3.1.2 Feature Generation

The algorithm moves to the step of cluster feature generation. First, features of the object must be tracked from frame to frame, which we do by first generating “nodes” on the image corresponding to points on the object to track. This sub-step only occurs at the first frame. The points we choose to track occur at “corners,” that is, where there are large color intensity gradients in orthogonal directions. For this, we employed the Shi-Tomasi algorithm.

After selecting the nodes to track, the Lucas-Kanade algorithm was used to track the location of the nodes from one frame to another. Both the Lucas-Kanade algorithm and the Shi-Tomasi algorithm operate on the 2-D image. Figure 1 below shows the image with the features that are being tracked.

Figure 1: Bike Pump data with feature tracking. Dots represent the corners that are being tracked.

Once we have the nodes tracked, we must develop a feature vector for each node. The goal is to develop a feature vector for each node, describing its movement with respect to the other nodes. From this, we hope to eventually cluster the nodes by the links to which they belong.

We make a key assumption about nodes in order to aid with designing a suitable feature vector: if two nodes occupy the same link, then the 3-D distance between the two nodes doesn't change throughout the course of a video. We can weaken this assumption by saying that with noise, the distance between the two nodes doesn't change very much.

From here, the feature vector seems natural: we record the relative change in distance between each pair of nodes (including the reflexive pair) over all adjacent time frames. In total, this gives feature vectors of length NM, where N is number of nodes and M is number of frames. This size is fine for many cases, but could cause memory problems for large N and M.

It's worth noting here that Oliver [1] uses a different methodology to determine the joint structure. Oliver measures the changes in the distance between nodes over time as well, but uses this data to generate a graph. Each node is represented by a vertex, and two vertices are connected by an edge if the change in the distance between their respective nodes is 0 for the entire sequence. From this constructed graph, a min-cut algorithm [5] is used to segment the graph into links. For us it was unclear how to decide how many times to run the min-cut algorithm here. Furthermore, we were worried about this method's sensitivity to recording noise. For example, what if it appears to the Lucas-Kanade algorithm that nodes on the same rigid are moving some very small distance? For this, it would be hard to generate the graph since we'd need to know an additional parameter, some tolerance on which we'd need to pass before we no longer consider two nodes' respective distances stationary. For these reasons, we opted to use clustering algorithms.

3.1.3 Feature Clustering

Using our nodes and generated features, the last step is to cluster the nodes. We use a spectral clustering algorithm based off the work of Ng et al [6]. In particular, we use the Spectral Library MATLAB implementation [7] [8] and use Ng spectral clustering in conjunction with Ward hierarchical clustering as

a mapping method. Figure 2 below shows the bike pump data after it has been clustered. As seen, the pump handle is clustered together.

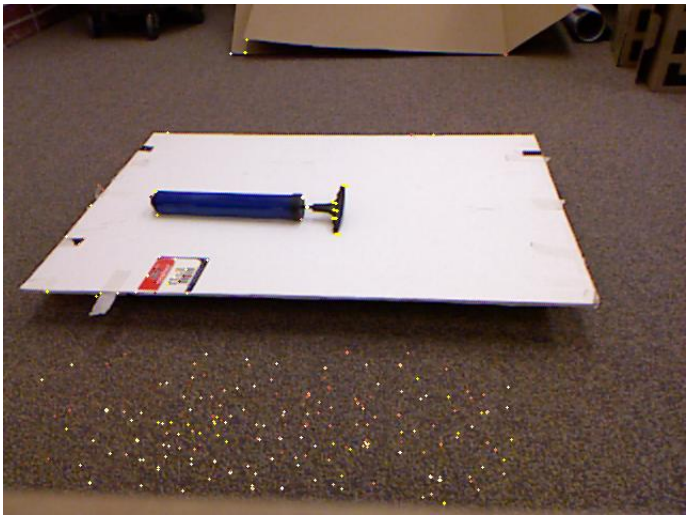

Figure 2: Clustering of bike pump data. Pump handle is clustered together. Algorithm cannot distinguish rest of image since it is motionless and groups it at random

3.2 Location of Interaction

We could consider each pair of possible interactions, but there are ways to eliminate some interactions based on some heuristic features. The reason for doing this is obvious, which is to simplify the classification process. We aim to prune out potential interactions which appear unlikely. We do this by removing potential interactions which are very far away. This distance was determined to be a value proportional to the standard deviation of the size of the cluster. However, there is no guarantee for this pruning to be accurate. Therefore it is better to be conservative here, and not prune too much, because we can classify pairs of clusters as "interactionless" later.

We label the location of interaction as the midpoint between the two nearest nodes.

3.3 Classification of Interaction

We naturally generate several features to track, which we partially base on existing literature. After the links have been segmented, we generate centroid of the points which we are tracking. We mainly use the motion of this centroid in order to track features.

In particular, for each potential interaction (as calculated earlier) between centroids, we record vectors corresponding to its frame-by-frame motion in each of the three Cartesian directions. We have four vectors, $x, y, z,$ and t, where the last vector just represents the time lapse between frames, which we keep constant. The $x, y,$ and z vectors are calculated by subtracting the motion of one of centroid from the other. From these vectors of data, we run an isomap algorithm, attempting to embed the data either into 1 or 2 dimensions, and keep track of the one with the lower residual variance.

The motivation for using isomaps is that constraints often force the motion down to lower dimensional surfaces and contours. For instance, consider a revolute joint. The centroid will sweep out motion that lies on some portion of a circle, which is a 1-dimensional object. Thus, the isomap will determine that this is clearly a 1-dimensional embedding. The same will be true for things like tracked motion, and prismatic joints, which also sweep out 1-dimensional motion. Meanwhile, the motion that sweeps out from something like a ball-and-socket joint will lie on a 2-dimensional spherical surface. We postulate that deformable objects, which will be represented by many closely coupled small clusters, will lead to non-correlated motion and thus will not permit a good lower-dimensional embedding.

Previous work by Sturm [2, 3] used 6 degrees of freedom of the object; linear and rotational motions were both tracked. For now, since we are using centroids, which are points, rotation is untrackable; however, we will consider ways to use our entire set of tracked points to approximate rotational motion in the later stages of this project.

After we're able to identify the correct embedding, we extract features from our isomap. In particular, we extract the average curvature of the motion and the variance in the curvature. We also measure the covariance matrix of the motion in the original feature space, and pass that along as one of our features. We also include the residual variance of our embedding as a feature.

Another two features we use is the number of tracked elements in neighboring clusters, and the standard deviation of the distances within the clusters, averaged between the two clusters and over all time steps. The hypothesis here is that rigid bodies will have large numbers of tracked elements spread out over large distances, while deformable objects are approximated by lots of little clusters that are tightly packed.

4 Results and Discussion

4.1 Clustering Accuracy

Generating the link structure was not as accurate as originally hoped. Figure 3 below shows the result of clustering on the backpack data. As expected, no real links were found together because the object was deformable and had no real structure to the movement. Thus, it led to a very poor guess as to the areas of interaction. In essence, it ended up roughly being an average of all the points in the image. Overall, this algorithm was able to cluster 22 out of 36 images correctly, yielding 61.1% accuracy. However, this can be improved.

Since there's no way to distinguish between the background and pieces of an object that are not moving (i.e. the base of a pump as the handle is moving), we attempted to manually define a region of interest for our image. Essentially, we'd focus on clustering the images into a predefined area in the image, which was hard coded in by us. Figure 4 shows this result after running our clustering and interaction detection on this restricted data. As one can see, the results are much better than what is seen in Figure 2. The base is strictly its own cluster and the handle is its own cluster, and the rest of the image is ignored. However, this approach takes too much human involvement, and thus, was not pursued for the entire

dataset. It is here to illustrate the algorithm can work more effectively with more restrictive settings and human intervention.

Figure 3: Clustering of the nodes denoted by the colored dotes and lines on backpack data. The large white circles represent the location of interaction

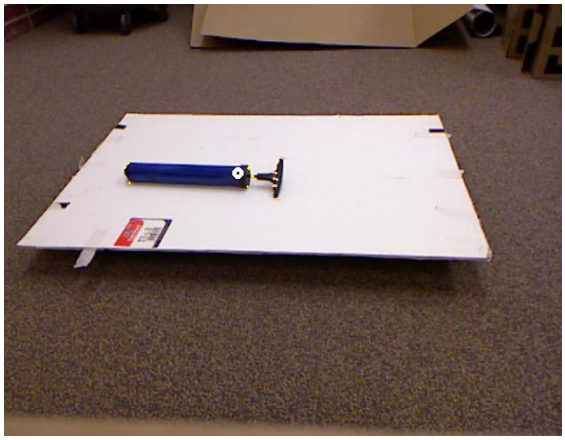

Figure 4: This is the clustering after the region of interest restrictions have been applied to bike pump data

4.2 Interaction Classification

Unfortunately, there was no enough time to complete this step. The idea was to run soft max regression on features described in section 3.3. However, the training step for even one iteration was taking an indefinite amount of time. Hence, performing LOOCV on the data set would have taken an unknown amount of time. No alternative was able to be performed with the time left.

4 Future Work

There could be many things to help improve this algorithm. As one can see, defining a region of interest seemed to yield the best results for clustering and interaction detection. However, that takes a lot of

human involvement. Developing a way to automatically define a region of interest based on movement in image could be one way to automate this problem.

Though we opted not to use image segmentation, applying it might be beneficial in future cases. It is worthwhile exploring applying very minimal segmentation to find a good medium between segmenting without losing details of the object that needs to be tracked. Doing so would remove the noise of the image and prevent the tracking algorithm to needlessly track or be erroneous to small details.

Lastly, finding an alternative for interaction classification is very important. An essential goal with many projects involving robotics is the ability to perform the algorithm online. Soft max was not the right approach as the training was taking too long. Several fixes could be to prune our training set and features more to reduce cost of training. Alternative would be to find another classification algorithm, such as SVM-Multi [9].

5 Conclusion

We have presented a new method for both detecting and classifying object interactions using stereoscopic data. The use of spectral clustering, as a means of determining the segments and joints, allowed for a completely unsupervised method of using this data to build a link structure of the object in its environment. However, the results were not nearly as accurate as hoped, and ways of improving this have been discussed in section 4. We believe that be finding a way to automate this region of interest will prune out much of the data by throwing out unnecessary information. Finding a faster classification scheme will also allow this algorithm to run online with the intended goal to be run without any human intervention. This combined with some very minimal segmentation can lead to a very viable method for quickly understanding the interactions of an object in an environment.

Acknowledgments

We'd like to thank Professor Saxena and Akram Helou for their helpful input on this project.

References

[1] D. Katz and O. Brock. Extracting Planar Kinematic Models Using Interactive Perception. 2007

[2] J. Sturm, V. Pradeep, C. Stanchiss, C. Plagemann, K. Konolidge, and Wolfram Burgard. *Learning Kinematic Models for Articulated Object*. 2009.

[3] J. Sturm, K. Konolidge, C. Stanchiss, and W. Burgard. *3D Pose Estimation, Tracking and Model Learning of Articulated Objects from Dense Depth Video using Projected Texture Stereo*. 2010.

[4] Intel. http://www.intel.com/technology/computing/opencv/.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Introduction to Algorithms. MIT Press and McGraw-Hill, 2001.

[6] Ng, Jordan, et al. On Spectral Clustering: Analysis and an Algorithm. 2001.

[7] SpectraLIB - Package for symmetric spectral clustering written by Deepak Verma. Updated by Tatiana Maravina.

[8] MATLAB version 7.8.0. Natick, Massachusetts: The MathWorks Inc., 2009.

[9] T. Joachims. *Support Vector Machine for Complex Outputs* version 2.00. 2004

APPENDIX

Detailed listing of data recorded. Items listed in italics have not been recorded yet.

Items	Description	Interaction
Calipers	Analog calipers opening and closing	Prismatic
Bike Pump	A small bicycle pump, pumping action	Prismatic
Umbrella	Closed umbrella moving up and down	Prismatic
Helicopter	4 propeller helicopter, spinning propellers	Rotational (Revolute)
Book	Book opening from a closer position	Rotational (Revolute)
D-Link	Wireless router with antenna, antenna being rotated	Rotational (Revolute)
Pliers	A pair of pliers opening and closing	Rotational (Revolute)
Bottle Caps	Bottle cap being spun open and closed	Rotational (Screw)
Hangar	Hangar being removed from a hooked position	Rigid Constraint
Rope Hook	Rope being pulled by hooked contraption	Rigid Constraint, 1-Manifold
Mouse Cord	Cord being manipulated randomly	1-Manifold
Bunjie cord	Cord being stretched	1-Manifold
Rope Wire	Wire being bent around	1-Manifold
Jacket	Jacket being zipped up and down	2-Manifold
Backpack	Backpack being zipped open to closed	2-Manifold
Magazine	Magazine page being folded and distorted	2-Manifold
Paper Towel	Paper towel being distorted	2-Manifold
Card Sleeve	Dot object moving around in a card sleeve	2D-Planar Constraint
Cone	Cone moving around on surface	Separable
Bottle	Bottle moving around on surface	Separable
Object Assortment	Multiple objects moving around on surface	Separable
Mouse	Mouse moving around on surface	Separable
Seal and Gnome	Seal and gnome objects moving on surface	Separable