Exploring the Motion Manifold for an Articulated Arm

Avikalp Kumar Gupta & Amitabha Mukerjee

November 18, 2015

Abstract

This project is a step towards the extension of visual motion planning algorithms (specifically Debojyoti Dey's M.Tech. thesis work [Dey (2015)]) to 3 dimensions. In this project, we try to generate the *motion manifold* of an articulated arm using only *visual input*. The articulated arm considered in this project has 3 revolute joints, and hence has 3 degrees of freedom.

We use a simulation environment with 3 cameras (vision sensors). Hence, each individual configuration of the robot is expressed as a set of 3 images. We call this set of images, or the configuration it represents, as a *pose* of the robot. A graph is generated over these poses by connecting *k-nearest neighbours* for each pose. The distance between two poses is measured as the Euclidean distance between the pixel values. This graph is supposed to represent the motion manifold of the robot.

When the graph is constructed over the joint angles, it results in a motion manifold (a thick hollow cylinder, $\mathbb{R}^2 \times S^1$) as expected. The visual manifold is corrupted owing to similarity between distant poses, and results in *short-circuits*. The phenomena are discussed and possible approaches to overcoming it are proposed.

1 Introduction

Our ultimate aim is to plan paths of multiple robots. Debojyoti had developed a roadmap composition technique in multi-robot environment. Key traits of the algorithm:

- *decoupled* motion planning (much more efficient than *centralized* motion planning in terms of time)
- oblivious to configuration parameters (The algorithm is implemented on a visual configuration space, and hence is a *generic* method)
- probabilistically resolution *complete*
- Space optimality: *neighbourhood product lattice* used so that Cartesian products of individual roadmaps are dynamically generated.

This algorithm was able to plan paths for 2-D robots.

"Traditionally, the problem has been handled using a configuration space defined in terms of ***motion parameters*** *of the robots. An example of such motion parameters are joint angles of an articulated arm. In this work we propose a novel approach based on* ***visual input*** *alone. Such a model works on a random sample set of images of the robots without knowing the motion parameters concerned."* [Dey (2015)]

Even though the vision based model makes the algorithm more generic with respect to robots, it raises issues regarding the information captured. Debojyoti implemented the model for 2-D robots. In 2 dimensions, all the instantaneous information about all the objects in the environment can be captured by a single photograph. This includes the complete configuration of the robot. The same is not true in 3 dimensions.

The need of images of the environment from multiple views is obvious. Although, the number of cameras that will be sufficient to capture the *complete* information about the environment depends on the environment itself because *occlusion*, which depends on the geometries of the robots and the objects, can give rise to incompleteness. Algorithms, which can generate the a 3D image using a scan through the environment Bhartiya and Mukerjee (2015), can be used to solve this problem in the future.

1.1 3-D Articulated Arm

Since we wanted our method to operate on only visual input, hence we moved forward to work on an algorithm, which might not always be able to find a path, if there exists one, but will only output *correct and collision-free* paths. We started with a 3-D articulated arm with 3 revolute joints (3R) as shown in the figure 1. We installed 3 cameras at orthogonal positions in the environment, which will capture the visual input for our algorithm.

This project uses *v-rep* for object modelling and simulation in 3 dimensions. The arm has been created in the *v-rep* environment, and all data, including joint angles' information used for validation, is extracted using the *remote API* functions provided by v-rep. The code for this project is written in *Python* using multiple libraries, including some created by M.S.Ramiah, IIT Kanpur. Stanford's MATLAB code for ISOMAP was used for visualizing the lower dimensional manifold of the images.

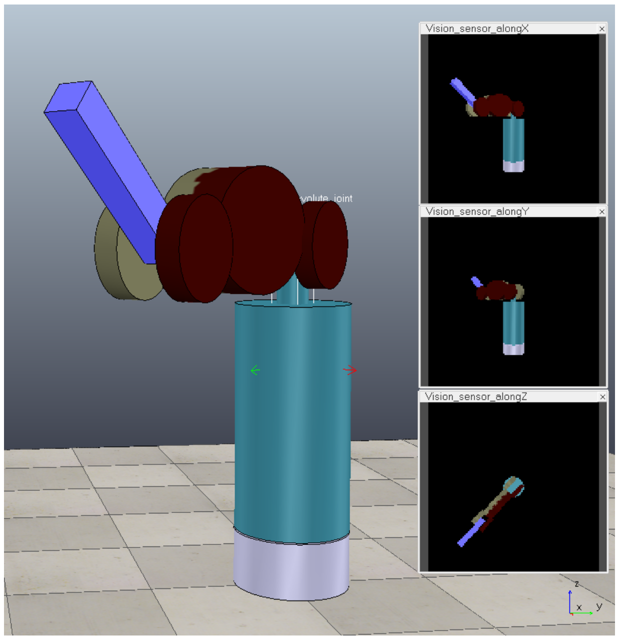

Figure 1: Robotic arm with 3 revolute joints - Created in v-rep. The 3 smaller windows show the image captured by the vision sensors installed in the scene *(image has been generated using v-rep software)*

2 Data Generation

One of the problems, to begin with, is the data generation. In debojyoti's work, the images, which were used to generate the roadmaps for each robot, were generated in the following manner:

1. An image file, similar to the ones illustrated in figure 2, represented the arena.

2. A position on the arena is chosen at random (and an orientation is also chosen at random) to place the robot objects.

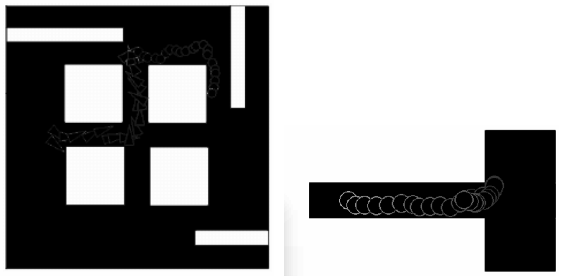
Figure 2: Examples of arenas for Debojyoti's code: Black implies free area and white means obstacle. *Image Source: Grayscaled (to highlight the arena) versions of images from [Dey (2015)]*

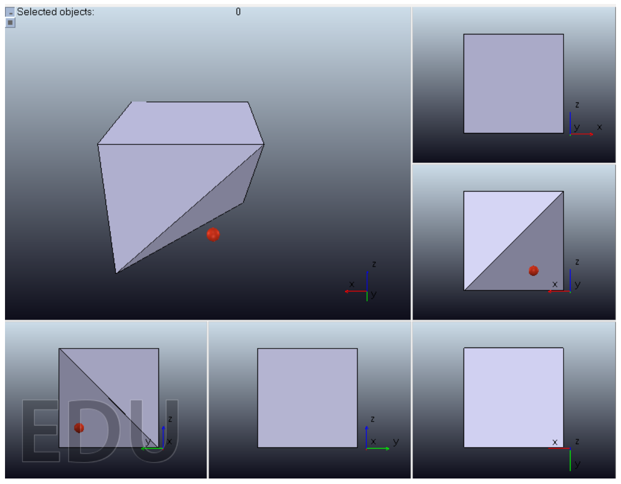

Figure 3: (left-top)A small sphere near a truncated cube; (others) all 5 cameras (placed at orthogonal positions) detect "false" collision *(image has been generated using v-rep software)*

3. Images are validated on the basis of collision with obstacles. Hence, an image, in which any pixel of the robot coincides with any white pixel, will be rejected.

4. Each of the remaining image represents a valid pose of the robot in consideration.

And for the articulated arm:
[**Ask and write here**]

2.1 Image Validation

We have to capture images of the environment using cameras and use the information from *all the images* to extract information. While generating the set of valid poses, for example, if we want to check collision, we have to go through all the images: the pose is valid if there is *atleast* one image in which the objects are not colliding.

But it turns out that we can never have enough (fixed) cameras. Consider the figure 3. The environment, in this case, consists of 5 cameras, strategically placed at orthogonal positions to capture maximum information about the scene. The environment does not have any collision, but still all the cameras together are unable to classify this pose as valid.

Apart from the genericness, the paper also describes the closeness of the vision based model to humans:
"Human beings do not appear to use global coordinates; instead, motion is encoded (both consciously and implicitly) in terms of relative poses. Mammals navigating in familiar environments acquire "place cells" which indicate their position in space, but these are organized in a system of neighbouring columns in the brain that encode nearby places. Thus, the representation appears to encode space in terms of nearby positions rather than a global coordinate."[Dey (2015)]

3 Idea/Approach

The approach is similar to Debojyoti's work, which includes the following steps:

1. Generate images for a particular pose of

a robot from multiple cameras. Repeat for n poses.

2. Ensure the validity of these poses, i.e. handle *"static collision"*
3. Create a graph using these poses (described by the set of images corresponding to the same motion parameter values)
4. Assign weights to the edges of the graph using some metric, preferably distance between the poses
5. Based on these weights, using K-nearest neighbours for each pose, create a graph which represents the manifold.
6. Construct a road-map from the initial to the final pose (Shortest paths in the above graph).
7. Repeat the above steps for all the robots in the scene
8. Create a composite roadmap using the roadmaps of all the robots (this includes handling *"dynamic collisions"*)

3.1 Graph Generation

1. A single set of images for the static obstacles will be captured.
2. Each robot will be simulated in an independent environment. Random *target angles* for all joints will be given during the simulation, and images will be captured when the robot becomes stable. Simulation will be done to avoid impossible poses (figure **??**).
   Drawback: Reduction in the uniformity of the randomness as the boundary configurations become more probable than the other configurations (figure 4).

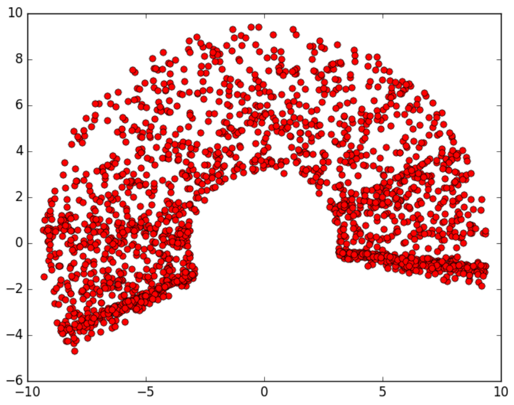

Figure 4: Plot showing accumulation of data points on the boundaries (*Details: Radial distance* $= 2\pi + \theta_2$, *and angle* $= \theta_0$)

3. Poses, in which the robot "seems to be colliding" with some obstacle in **all** the views, will be rejected. A robot "seems to be colliding" with some obstacle in a view if for some pixel, the robot's image and the obstacles' image, both have non-zero values in that view.
4. A *completely connected* graph over all these remaining poses will be created. Each edge will have a weight, based on some metric (eg. distance).
5. Invalid edges (which lead to collision with obstacles during transition) will be dropped using *local planner*.
6. K-nearest neighbours for each vertex (pose) will be retained, and rest of the edges will be dropped.

This will generate the graph which represents the manifold over which the robot can move.

3.2 Roadmap Composition

After the manifold of the movement of each robot is known, i.e. the above graph is constructed for each robot, we will plan simultaneous paths for the robots from a given initial pose to a given final pose, to minimize, either the total distance travelled by the robots, or the net time taken for the transitions:

1. For each robot, get the shortest paths over the manifolds, from their initial to final poses.

2. Detect any collision .... [**see how Debojyoti handles these**]

4 Current Progress

We have used *v-rep software and API* to generate images [see 4.1]. An external python script was written using v-rep remote API which would:

- Start the simulation of the currently active scene on the v-rep window

- Give random values to the target positions of all the joints

- Wait for the robot to reach a stable configuration

- Capture the images from each vision sensor, and store the motion parameters for that pose

Now this script could be used to generate the raw data for any robot [see section 4.2 and **??**].

4.1 Choice of a 3-D modelling software

After going through some 3-D simulation softwares, we chose *v-rep* over the others due to the following:

1. Its free version (for educational purposes) supports most of the relevant features

2. It is a light software and supports 3-D rendering

3. 3-D modelling is supported inside the software, as well as 3ds-max/blender/CAD can be imported

4. Objects and simulations can be externally controlled using its remote API for both, C/C++ and Python. (remote APIs for other languages are also supported)

5. Proper documentation is available, and v-rep is also supported by an online forum

4.2 Image generation for a robot with 3 dofs (3 revolute joints)

The robot in figure 1 was made using primitive shapes in v-rep. It has 3 revolute joints: One at the base, which makes whole system rotate about z-axis. And the other 2 have axes parallel to the $x - y$ plane. This was created because if you consider the system installed on top of the rotating base cylinder, it is the same R1-R1 articulate arm explained in [Dey (2015)]. Hence we have sufficient knowledge and understanding of its behaviour.

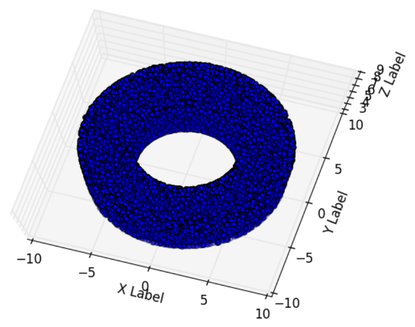

Figure 5: 3-D plot of θ_0, θ_1 and θ_2 is a hollow cylinder (*Details:* $2\pi + \theta_0$ = *distance from z-axis,* $2\pi + \theta_1$ = *distance from x-y plane &* θ_2 = *angular displacement along z-axis*)

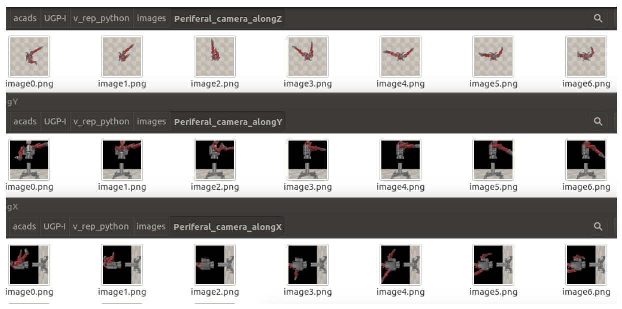

Figure 6: An external python script (which uses v-rep remote API) captures images from various peripheral cameras. Images with the same name belong to the same pose.

4.2.1 Manifold

As we know, the C-space of the system on top of the base cylinder would be a toroid (T^2), if there were no restrictions on the angles which were achievable by the joints. But since both of the top 2 revolute joints cannot rotate 360°, hence the configuration space is homeomorphic to $\mathbb{R}^2$ space, i.e. a rectangle.

The base revolute joint can rotate 360°, and hence is an S^1 joint. Hence we expect the C-space of the complete robot to be a ring-like structure (see figure **??**).

Unfortunately, this is not what we get in our first attempt. Figure **??** illustrates the manifold of movement obtained on the graph generated on the images. You can observe this as the expected graph with short-circuits.

4.2.2 Difficulties

The following were the major reasons of these short-circuits:

- a jump when 2nd link and the base cylinder are in the same spatial region.
- the opposite faces of the middle link get inter-changed (θ_1 change by 180^o, and θ_2 and θ_3 change signs.

4.2.3 Proposed solutions

The solution of the first problem is to distinguish the different ends of the 3^{rd} link using different colors (figure **??**).
The 2nd problem was a fore-seen problem, and it was expected that if we use different colors for the 2 different sides of the middle link, we will be able increase the distance between them. But it turns out that this distance did not increase by much.

5 Future Work

For the direct extension of Debojyoti's work, the first thing to do now is to complete the remaining steps.
But it is quite evident that, due to occlusion, the results produced by it will not have all the traits which the planar model had. Particularly, the algorithm will not be probabilistically resolution-complete.

Hence, now we will proceed by understanding how humans and animals plan their motions. The implementation will be similar to paper by Amitabha Mukerjee and Divyanshu Bhartiya called “A Visual Sense of Space” [Bhartiya and Mukerjee (2015)] in which they have proposed a visual characterization for the complex motions of an unknown mobile system.

Another method could be using body-fixed cameras for capturing information about the environment (example: the case of Baxter) and making the robot learn to find the shortest path. Some study about the various neural network learning algorithms (deep learning and RNNs (including LSTMs)) was also done during the course of this project to proceed in this direction.

6 Acknowledgement

I would like to specially thank Mamidela Seetha Ramaiah, a.k.a. MS Ram who had shared with me some very useful libraries he had developed for processing images and generating graphs over them. He also helped me in understanding concepts related to robotics and manifolds (using Choset’s book Choset (2005)).

I would also like to thank Debojyoti Dey for spending some very precious time in explaining his implementation.

Lastly, I would like to thank Prof. Amitabha Mukerjee, who is the superviser of the project, for his guidance and expert advice through the project.

References

Bhartiya, D. and Mukerjee, A. (2015). A visual sense of space. *Procedia.*

Choset, H. (2005). *Principles of Robot Motion: Theory, Algorithms, and Implementation.* A Bradford book. Prentice Hall of India.

Dey, D. (2015). Visual Motion Planning of Multiple Robots by Composing Roadmaps. Master’s thesis, IIT Kanpur, India.