A Gentle Introduction to Grasp Planning

Author:
Bikramjot Hanzra
Tae-Hyung Kim
Rushat Gupta Chadha
Tushar Agrawal

Instructor:
Dr. Siddhartha Srinivasa

February 12, 2016
Contents

1 Introduction 2

2 Challenges in Grasp Planning 3

3 Pregrasp 4

4 Controller 6

5 Previous Work on Grasp Planning 7

5.1 Shape Primitives ......................................................... 8
5.2 Inspiration from Humans ............................................... 8
5.3 Eigengrasps ............................................................... 9

List of Figures

1 2D Barrett Hand ............................................................. 4
2 Enveloping Grasp ........................................................... 6
3 Barrett Hand in dimension ............................................... 6
4 2D Coffee Mug primitized using shape primitives ................. 8
5 Mapping Hand Postures in a lower dimensional (2D) space .... 9
6 Eigengrasp planner test using 5 hand models to grasp each of 6 objects ................ 9
1 Introduction

In these section of notes we will be discussing about Grasp Planning, which is sort of asking this simple question –

How do we generate good grasp candidates?

Think of the grasp verification as a black box that takes in input some configuration of the hand say $C_H$ and some configuration of the object $Q_o$. The output is either a success or a failure and in addition, it can give us a score. The score is a measure of Grasp Quality. We want to find good hand and object positions such that when we pass the grasps through the grasp verification, we get a lot of success.

We will first discuss about the nominal algorithm for grasp planning which is in simulation and the reason we want to start doing it in simulation is that –

• Firstly, running something on the real robot is expensive.
• Secondly, we don’t want to fail with the object in the real world in which case the object might break.

So, we run a lot of simulations to find good grasp candidates. Good grasps are not always guaranteed to succeed in the real world and they do fail but at least it means that we have done the best we can in simulation such that we give the robot a good chance of succeeding.

The first thing in grasp planning we do is that we pick a pregrasp configuration. We will talk about this in detail during our subsequent discussion. Once we have the pregrasp configuration, the next task is to run a controller. A pregrasp configuration is a sort of an attack direction or an attack pose of the hand with respect to the object. A simple example would be picking a pen in front of the robot. The hand would first come in some pose relative to the object and then the controller will be executed i.e. the fingers will squeeze with feedback control and at the end we will have the hand and the object touching each other. At this point of time, the controller will pass this information to the grasp verification and if this succeed, we execute the grasp. At a high level, the algorithmic steps along with the issues involved with each (in italics) are shown in Algorithm 1 below.

Result: Execute grasp

while not SUCCESS do
  1. Select Preshape – High-dimensional hand posture space
  2. Select Pose – Diversity (Clutter, Arm reachability)
  3. Select Controller – Realism
  4. Run Controller – Computation Time
  5. Verification – Post-grasp stability, Uncertainty

end

Algorithm 1: Process in Simulation with issues involved

The key issue with doing this again and again is that this process is slow. There is a finite amount of time that is consumer before the grasp verification is done. Approximately, one evaluation can take up to .5 seconds. This is because in general rigid body dynamics is hard. This is also partly because solving for contact mechanics is particularly hard. This is the same reason why animated character in computer animation do not interact that well with the environment and simulating contacts is an open area of research.
This brings us to one more fundamental question –

**How many successes we should have before running it on the real robot?**

The first thought that comes to mind is to iterate the process until we find the first stable grasp and then we try to execute it. This approach might not be the right thing to do because this particular grasp might not be reachable by the arm or their might be an obstacle in the way during the grasping that was not considered earlier. So, we need a wide range of choices.

The first idea and probably the best idea is what is called a **Grasp Tables**. The idea is really simple and elegant. *We come up with a precomputed set of successful pregrasp configuration for an hand-object pair.* Their is a lot of nuances involved with this sentence and we will explore each of these very carefully. The grasp tables are precomputed before we actually see the object. We store a set of attack directions that we know in physics will work. When we see the object in the real world, we select the best set of grasp based on some metric of closeness and we only simulate those to see if they work well.

### 2 Challenges in Grasp Planning

The big challenges here are –

- **Generalization** – Imagine we calculated 10 different grasps of an object and we cached them. Now in the real world, we see the object not in one of these 10 cached configurations. We are then forced to compute the same thing again. So, we need to have enough grasps in our grasps tables so that we generalize well across as many scenes as possible. It is the same problem that we try to solve in Machine Learning where we want more and more data to perform well on new examples. One possible solution is that why not store a ton of these configurations – hundreds, thousands or even millions. In the age of big data, it is actually not a bad idea. One possible problem with this strategy is that recall in real time then becomes pretty hard. Another problem is that evaluation is a big challenge. This is sort of a fundamental issue with anything we store.

- **Clutter** – Another challenge is clutter. Imagine we create a grasp for an object. In the real world there will be other objects also near the object and then the problem becomes complex. One possible solution is to consider other objects also in the grasp table. The downside of this approach is that the search time becomes almost twice by adding one more object. Indeed, the computational time is exponential with the number of objects added to the grasp table. So, there is a trade off between how accurate we are and how fast we are! This brings us to another key requirement which is no matter where the clutter is, our algorithm should find a way out. In our grasp table we want to have approaches that attack the object not just from one side but from every side. This is exactly what we end up doing in application as we want diversity in all possible approach direction. The grasp table is created with respect to the coordinate frame of the object. It is calculated without any clutter and the good side is that it can generalize pretty well in cluttered environments.

- **Noise** – Throughout our previous discussion, we made the assumption that the pose of the contact points is known perfectly but it is hardly the case in the real world. Another unanswered question here is that – how due we dealing with noise? There exist super complicated ways to deal with it One of the simple solution is to do a statistical analysis. Think of the
grasp verifier as something that produces a boolean output and we are interested in finding the expected value. If we have some noise in the positions of the hand and the object, we can simply run simulations or do something more involved like Monte-Carlo Sampling and then we count the number of zeros and ones we get. We can think of it as a loaded coin which will either land heads or tails and we are interested in knowing how often we get heads.

- **Configuration of the arm** – One of the key things that is missing from this grasp table is the arm itself. The positive of this is that this will work for any arm and it works well for any relative position and orientation of the object w.r.t. to the arm. We want a sufficiently large grasp table as there will be grasps that look pretty good but the arm might not be able to get there due to maybe clutter or even the constraints of the arm itself.

- **Divorces arm from the hand** – Basically, we are saying that the hand is flying around in the space and then we create a grasp table. We then attach the arm to the hand to see what happens assuming that there is an smooth isotropic uncertainty in the hand. The solution to this problem might be to include the arm’s configuration in the grasp table.

That’s fair amount of theory and now let’s dive into a bit of mathematics of grasp tables and as promised previously, we will discuss the pregrasp in detail now.

## 3 Pregrasp

We will first define what a **Pregrasp Configuration** is and then we will consider a floating hand in $SE(2)$ and $SE(3)$. In $SE(2)$, the position and orientation is given by $(x, y, \theta)$. The degree of freedom of the hand is defined by $C_H$. So, we have the position, orientation and shape of the object. Our hand has at least 21 degrees of freedoms. This is already a challenge, since we are dealing in a $21 + 6 = 27$ dimensional space. This is a very large space and a motivating problem here is to solve the dimensionality of space.

![2D Barrett Hand](image1)

Let’s look at an example. We will use the square that we love to grasp in 2 dimension. Remember that all grasp tables are object centric i.e. there is a coordinate frame attached with the object. All the positions and orientations are measured w.r.t. this coordinate frame. We also have a 2D
Barrett Hand which is the 2 dimensional version of the three finger hand that CMU’s HERB uses. It has 2 degree of freedoms – $\theta_1$ and $\theta_2$. We can then write $C_H$ in terms of $\theta_1$ and $\theta_2$ as –

$$C_H = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$$

It has a cool clutch mechanism where the distal joint rotates at a rate proportional to the proximal joint. This is very similar to human fingers. The nice thing is that with a single motor we can get a curving in like behavior. The Barrett hand also has a nice mechanism by which it will stop as soon as it touches something.

An important thing while doing grasp planning is to reduce the dimensionality of the search space. In our example, we are in a 2 dimensional space, so it’s relatively easy. Since it is important to us to reduce the dimensionality in case of more complex examples, we will define what we call a Preshape. Preshape is a subspace of $C_H$ considered for grasping. So we say that even though our finger can do all sorts of funky things, we will restrict the set of things we do to a subspace of $C_H$. For the 2D Barett Hand example, we will look at the subspace where $\theta_1 = \theta_2$ i.e. we will assume symmetry and by doing this we have reduced the dimensionality of the space from 2 to 1. This comes as an advantage in that we only have to search a smaller space for grasps that might be valid but at the same time it comes with a disadvantage. It is obvious to note that starting with non-symmetric shapes is not allowed. So, now the question that arises is that how this preshape is created? Guess what, it’s a bit of black magic!

We will now try to unravel the black magic to a certain extent. Imagine we have an object that needs to be grasped and we have the hand at some $(x, y, \theta)$. To reduce the dimensionality of the pose space, instead of searching completely in $(x, y, \theta)$, we only look for grasps that are at some $(r, \theta)$ with the normal of the hand pointed inwards. The intuition here is that if we want to grab something, then the hands should point towards it. Having said this, there are grasps where the hands don’t point towards the object to be grasped and in our meticulously disciplined world we are going to give up on these cases. This is again to encourage the grasp planner to come up with good grasps.

So our subspace is given by –

$$q = (\theta, r)$$

where

$$(\theta_1 = \theta_2) \Leftrightarrow a$$

Here $a$ is called the aperture of the hand. So, we are parameterizing our subspace by the gap between the fingers. This is an important parameter because it gives us an intuitive way to decide how wide we can open the hand to grab an object. It is usually tough to define a measure in terms of the joint angles but extremely convenient in terms of a single parameter like $a$. 

5
4 Controller

Until now, we went from a pregrasp position in the space which could be anywhere in the real world to a more constrained lower dimensional parameterized subspace. But we haven’t yet discussed about the controller and this is what we will be discussing now. Once the Grasp Verifier tell the robot to grasp the object, what should the robot do?

The interesting part here is that we have some sort of control over the controller. There is a delude of ways in which the controller can approach the object e.g. all fingers approach the object together or one finger approaches the object first. After the contact is made we might want to close the hand. We want to make sure that our controller can reasonably execute some grasps.

If we have a much larger hand, we can envelope the object with the hand. This is called an Enveloping/Cover grasp. Figure 2 shows an enveloping grasp. In our parameterized world for our example, it means a bigger aperture. On the other hand, if we have a tiny aperture we get a Precision/Finger tip grasp.

The particular normal vector say $\hat{n}$ is called the approach direction. So, if we have a preshape and an approach direction, we are all set to run our controller. The search algorithm that we will look at essentially says –

- Sample an approach direction.
- Sample a preshape.
• Run the controller.
• Iterate if fail.

This will give us a set of good grasps. Here another fundamental question that pops out is that how do we sample approach directions. The approach direction should cover a unit sphere because we want to attack the object from all possible directions as we never know where the clutter might be, where the arm might be, if we could run the controller or not. Previously, we have discussed about force closure grasps. We want to make sure that any evil entity that can exert a unit ball can be resisted. If we are doing this in 3D as shown in Figure 3 we sample uniformly in \( SE(2) \). There is actually a cool algorithm to generate uniform points on a unit sphere which basically says generate normal samples –

\[
x_1 \sim \mathcal{N}(0,1)
\]

\[
x_2 \sim \mathcal{N}(0,1)
\]

where \( \mathcal{N} \) is a Normal(Gaussian) Distribution with zero mean and unit variance.

and we just project that on to the unit sphere i.e. we normalize it.

\[
\hat{n} = \left( \frac{x_1}{|x|}, \frac{x_2}{|x|} \right)
\]

where

\[
|x| = \sqrt{x_1^2 + x_2^2}
\]

This approach also generalizes well to \( n \) dimensions. This approach sort of covers everything but what are the drawbacks of this kind of approach. Think about using this approach to grab a coffee mug. Let’s ask the simple question, how well this approach will be practice?

It will work well for a sphere because it is isotropic in all directions. We will sample along the contact normals and then we will use them as the approach directions. In the case of the coffee mug, this idea produced some really nice grasps in simulation but in real world it failed every single time. The reason was that due to the direction of the contact normals the robots was trying to grasp the mug from under the table because it does not have the diversity that other algorithms have.

Why don’t we approach the outer point normal and the intuition is that we want to grab a box with the palm facing the side of the box rather than some arbitrary direction. Typically what we want is a set of grasps that cover our unit spheres but we also want a set of grasps that are good along the contact normals. There is no clear winner and we want to come out with a judicious combination of the two.

5 Previous Work on Grasp Planning

Grasp Planning is an open area of research. One of the biggest problem of grasp planning with grasp tables is that it assumes that the the exact geometry of the object is known. Below, we have
discussed some of the approaches that have been proposed before.

5.1 Shape Primitives

So, the question that this paper\[1\] asked was that what happens when we see an object that we haven’t seen before. They proposed to model objects using some predefined shapes like spheres, boxes, cones etc that are initially cached. When we actually encounter an object, we try to break them down into these set of primitive shapes. For the coffee mug example, the intuition is that we have never seen this coffee mug before but we know it is kind of a combination of a cylinder and a box. So, we will recall all the grasps of both to try them out. In reality, it fails in most cases but it is a good thing to start with. Figure 4 shows the actual and the primitized shape of the coffee mug. A big problem that they sidestepped was that the optimization problem of primitizing an object is ill-posed. Also, when we break the set of grasps of a coffee mug into two primitive types – a cylinder and a box, we are actually ignoring a certain set of grasps e.g. the union of a cylinder and a box.

![Figure 4: 2D Coffee Mug primitized using shape primitives](image)

(a) Actual Shape of the object  (b) Object reconstructed using shape primitives

5.2 Inspiration from Humans

A lot people have looked at how do we prepare grasps for robotic hands both from a computer graphics point of view but also from how people grab stuff. This particular paper [2] addresses this issue. The human hand has 21 degree of freedom. That number is gigantic! One of stunning result from this paper was that hw we grasp the object does not matter, rather it is the preshape of the hand before the grasp that matters. They were able to show that over 98% of the grasps that humans execute are done in 2 dimensions. Figure 5 shows the mapping of hand postures while grasping different objects in a lower dimensional (2D) space.
5.3 Eigengrasps

Researchers have tried to generalize the above behavior to robotic hands which is known as eigengrasps[3]. These eigengrasps are generated by hands and they have then mapped to robotic hands. These have somewhat shown to work well. Figure 6 shows the best hand postures found in a two-dimensional eigengrasp space using simulated annealing optimization.
To conclude, Grasp Planning is still an active area of research and there is a lot to explore in the uncharted waters of robotic grasp planning.

References

