# STAIR Subcomponent:
# Learning to Manipulate Objects from Simulated Images

Justin Driemeyer

CS229 Term Project

December 15, 2005

## Overview

For my project, I am working with Ashutosh Saxena on a subcomponent of the STanford AI Robot, i.e. STAIR. One of the goals for STAIR is to be able to pick up and manipulate objects in its environment using a robotic arm using a camera mounted on the end, which we are approaching with machine learning. However, in order to learn hundreds of images will be needed, all with information on camera orientation and object grasping point. So instead of taking a ton of pictures with a digital camera, the labeling of which would be tedious to say the least, we decided to train using simulated ray traced images, where the coordinate system is well known, then test on real images.

Although the final goal is to implement this work on a robotic arm with a variety of objects, that seemed beyond a reasonable scope for what could be accomplished in the time frame of a class project. So I focused on the classification problem at the heart of the project – can we train using only simulated images with the grasping point identified and get good results on real images?

## Related Work

There are several factors at play in this approach. For one, we are using just a single camera on the end of the arm and attempting to control a robotic arm in 3d space using a monocular image. Although we could use two successive frames in sequence and try to derive depth information from that, the images are not just taken from the top of a moving robot, but even worse from the end of an arm on top of it. So, the odometry information on the separation of the two images is error prone, not to mention computation time and simple mathematical limits to bifocal depth resolution. So instead once we start working on the robot we will be using work done earlier by Ashutosh on learning depth from monocular images.

It also builds on work described in the High Speed Obstacle Avoidance using Monocular Vision and Reinforcement Learning paper, where they trained the algorithm first using simulated images to get basic driving skills, then trained further in the real world. Their results using just simulated images showed that you can learn from simulated images, and the simulated images they used were pretty coarse open GL, worlds away from the most realistic scenes which can be constructed using a ray tracer.

## Simulated Images

Assuming that the more realistic an image the more applicable learning on that image will be to the real world, we decided to work with ray traced images instead of Open GL. All work in this area was done using POV-Ray and open source extensions in MegaPov. This also allows us to build objects using constructive solid geometry instead of having to model everything using polygons, but of course a ray tracer can use polygons too if we want, but for many objects CSG should work just fine. Furthermore CGS allows simple representation of things which in a polgonalized space would be very complex, such as a mug. Thus we have

a great degree of flexibility in object creation.

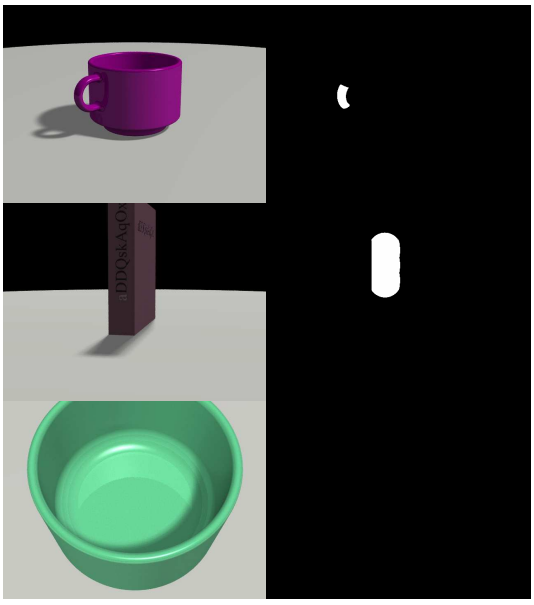
Initial work was done just in generating some sample images and balancing the degree of accuracy vs. render time. The current implantation has mainly used a simulated image of a mug, but after observing performance on that I tested the approach on a book object as well. Our current graphics take ~30-60 seconds to render per image, allowing a thousand images to be rendered overnight. However, based on experimental results, I believe in many cases 100-200 images will be sufficient.

The method we are using to identify points of interest is a smattering of feature detectors, and relies on the object having some bit of texture. So a bit of texture was added to the sample objects through a high frequency "bozo" texture, which has the effect of giving the object a little bit of noise. This approach can be applied to any object, mitigating the need to define specific texture mappings.

Each image is in a random position, with the main light casting a shadow at a random point along a general ceiling area. The mug is always in the same location on a flat white tabletop, but is of a random color. Some sample training images follow:

When designing the book object I decided to investigate the effect of randomizing the dimensions of the object. So, for the book the title is a random 10 character string, the height, width, and depth are also all randomized between certain parameters (i.e., the depth ranges from 0.5" to 1.5") in addition to the random light position, camera location, etc. Some example book images:

For training each image has a corresponding grasp image, which is black except at the regions where the object should be held. For the mug this is defined as the handle, for the book it is along the center of the spine. To define grasp points spheres were used, although other primitives could be used as well. Here are some sample training images along with their corresponding target images:



Note in the third image no handle was visible, thus no part of the image is labeled as the target.

Production of the labeled target images it entirely handled by the ray

tracer, so there is never any need for handling labeling the target region of the images after they are produced.

**Learning**

All learning was done using Matlab. Although far too slow for actual deployment on a robot, it is much easier to experiment with different methods and parameters there, and once those have been nailed down the program can be implemented in something faster.

The approach for the learning was to break each 480x640 down into 48x64 10x10 patches then apply a smattering of 51 feature detectors, which I received from Ashutosh. Each patch is then labeled as 1 if any part of it contains a grasp region. Even so, since many images don't even contain the grasping region due to camera angle or occlusion, nearly 98 % of regions ended up labeled 0 when training on the mug. This significantly weights the negative examples, so to even out the dataset, and to speed convergence by reducing the extreme number of training examples (20 images yield 20 * 48 * 64 = 61440 examples) negative examples are randomly trimmed until they are about equal. I also experimented with different ratios, up to 3 negative examples to 1 positive, but found that somewhere between 1:1 and 3:2 seemed to yield the best results. I also experimented with simply replicating the positive examples with some perturbation, but trimming the negative examples yielded better results and was also easier to parameterize.

For the actual learning algorithm I tried both logistic regression, which I wrote, and kernalized SVM from http://asi.insa-rouen.fr/~arakotom/toolbox/index.html. However, after spending a whole day on SVM attempting different kernels and different parameters, even when I got something that would converge if I changed the number of training examples it no longer converged. Besides, the SVM results were mediocre at best. So I decided to focus on logistic regression instead.

To fine tune the parameters, I experimented on the mug images. I tested varying the number of training images from 20 up to 300, and varying the ratio of negative to positive images from 1:1 up to 3:1.

**Results**

I tested two objects in each category, one which should be easier to identify, and one which should be more difficult. For mugs I tested one dark blue mug which would stand out well against the white background and which had a handle somewhat similar to the handle in the training example – large enough to fit the whole hand. The second mug was white so it would be harder to resolve against the white background, and it had a handle which only fits one finger, very different from the handle in the training example. For the book object, the first book is a C++ reference manual, very rectangular and with corners which should be easy to resolve. The second book is a hardcover yearbook with a rounded spine and very dark to the corners are hard to resolve.

Before trimming the number of negative examples, training and testing on the training images led to 99.8% accuracy in classifying regions as handle

or non handle overall, but only 73.6 % accuracy correctly classifying the handle regions. This of course translated to even worse performance on the real test images. After trimming, there was a significant improvement in grasp point detection, although the false positive rate also increased. However, this is not an issue as long as the grasp point is detected consistently and false positives are spurious. I found the best performance training on 80 examples and a negative to positive example ratio of 1:1 or 1.2:1. Regions identified as grasp point regions are highlighted by having their red component inverted. (which shows as a red or blue highlight) Note the consistently excellent detection of the easier target, while detection of the second "challenge" target fails in a few images (represented by image 6 in both sets) but is detected in most images.

81 Training Images, 1.2:1 Ratio, Mug

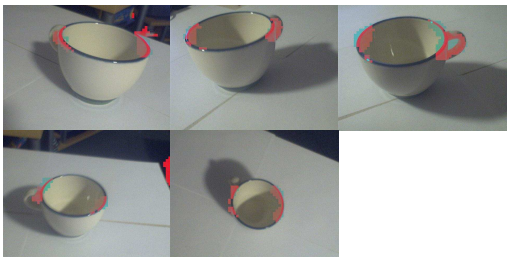
81 Training Images, 1.2:1 Ratio, Book

These images are a representative cross section. In the bottom right book image, note the spine is on the right side. This was the worst performance in terms of false positives, most were more similar to the third image. Full results for these parameters are on the next page.

**Future Work**

Continuing the project will involve producing more simulated objects, as well as experimenting with using multiple simulated objects for one class of objects (i.e., having several models of a mug instead of just one). Also, orientation of the robotic pincher needs to be incorporated as well, and then it needs to be implemented in a real-time language instead of Matlab.

Full Mug Results 81 Images 1.2:1 Ratio



Full Book Results 81 Images 1.2:1 Ratio