# 3D OBJECT CLASSIFICATION AND GRIPPING POINT LEARNING

CS / ECE / MAE 4758 – ROBOT LEARNING

*Sebastian Castro [sac77]* *Nathan Lloyd [nsl6]* *Rui Wu [rrw32]*

**Abstract:**

**There are many challenges present in the field of robotic manipulation. When manipulating an object using a gripper, there is more to consider than simply finding the location of the object and moving the robotic arm towards that point. We have developed a descriptive feature based machine learning platform that views a 3D point cloud and classifies it as one of three different object types: Mugs, Plates/Bowls and Glasses/Bottles. In this project, we use a database of classified 3D mesh objects as a Support Vector Machine (SVM) training set for new objects to be classified. Based on this classification we then apply different gripping strategies that best suit the type of object to be manipulated. This point and vector then serves as a reference command for a real or simulated manipulation platform.**

## I. PROJECT OVERVIEW

Our motivation for this work stems from discussion about ways to manipulate different kind of objects. When we consider an object with a handle, such as a mug or coffee cup, we can pick it up in several ways. The mug can be picked up by its center of gravity, by grasping its upper rim or by grasping the handle. Intuitively, because the object has a handle, we can conclude that the handle would probably be the best gripping strategy for a mug-type object. For something like a plate or bowl, however, there is no handle present to repeat this same strategy. In this case, grabbing the edge of the plate would be the best procedure. Similarly, for an object resembling a bottle, we can picture our optimal gripping strategy as grasping the middle of the object by its minor axis. While we can instruct a computer to follow these rules, we must first figure out what type of object a set of 3D points really represents.

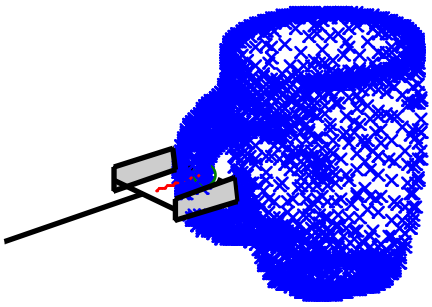

**Fig. 1: Computed Gripping Point for a Mug.**

In order to classify objects in one of the three predetermined categories we must develop a set of features that can best explain the qualitative differences between objects. We use ratios between dimensions in X, Y and Z directions to roughly determine the general shape of an object. To find handles, as in the case of a mug, we use an iterative circle-fitting algorithm which gives us a notion of symmetry between opposing sides of an object. We also look at the boundaries of 2D projections of the point clouds to find the relative sizes of objects' opposing ends.

With these features, we generate a vector that describes any given 3D object. Using machine learning techniques, namely Support Vector Machines, we can then classify models based on their feature vectors. For every object class we define, we have a separate SVM that classifies an object as positive or negative – positive meaning that the object is a candidate for that particular class. We then evaluate which class our object most closely matches. Based on this label, we then compute the gripping point and gripper approach vector specific to objects of that class.

We have written a C++ class that can interface with Robot Operating System (ROS), namely with the Willow Garage PR2 Robotic simulator. This class employs a GNU Octave-based package that we wrote on top of existing MATLAB packages to read and manipulate 3D mesh objects. All the SVM building, pre-

processing, and image displaying is done directly in MATLAB. The ROS-compatible section of our project takes in a 3D matrix of points or model filename (in **.ply** format) and classifies an object based on three SVM models. The final output is a gripping point and vector for the gripper to approach the object.

## II. APPROACH

### *A. Features*

We iterated through several possible numerical features that we could use to pinpoint a 3D model to a specific class. Our final result used the feature list as shown in Table 1. These features constitute a 14-dimensional vector that characterizes a particular object and is then used for SVM training and classification. We have assigned equal weight to these features by constraining each of them to take a value between 0 and 1.

| Feature | Description |
|---|---|
| 1 | Minor Axis Length/Major Axis Length |
| 2 | Middle Axis Length/Major Axis Length |
| 3 | Minor Axis Length/Middle Axis Length |
| 4-9 | Ratio of number of points on lower and upper ends of X, Y and Z axes. |
| 10-11 | Circle fit to 2D projected ends. |
| 12-13 | Rectangle fit to 2D projected ends. |
| 14 | Handle detection feature. |

**Table 1: Current Classification Attribute List.**

The first 3 attributes simply give us the general "envelope" shape of the object. For the object classes that we are considering, and extending to even further possible work, it is important to know how "long and thin" or "short and wide" an object can be described as. In the case of a bottle of perfectly circular cross-section, for example, we will find that the first 2 features will be small and close to equal, and feature 3 will be close to 1 because the middle and minor axes lengths describe the circular cross-section. For a spherical object, on the other hand, we expect that all 3 features will take a value close to 1.

Features 4-9 can quantitatively describe the degree of "taper" of a 3D object. This is done by "slicing" the mesh along the upper and lower bounds of the Cartesian (XYZ) axes with a certain threshold value and finding the number of points present at the two ends. For instance, if we are observing a conical object with a centerline along the Z-axis, we expect that the lower end (with the circular cross-section) will have many more points than the upper end, which is just the tip of the cone. To ensure considerably accurate information about slices, in the case of tips and points, we continue to raise the threshold as needed until at least 10 points are found in each mesh slice.

Features 10 through 13 are quantitative descriptors for the cross-sectional shapes of an object. A 2D projection of the object is created such that all the points are "squashed" down to a single plane. We then iterate through different sizes of circles and rectangles (respective to the feature) and find the RMS error of the best geometric fit to these projections. In order to ensure that this feature takes a value between 0 and 1, we have placed the RMS error value through a logistic regression sigmoid function with a design parameter **c** as follows. This parameter is chosen depending on the desired sharpness of the logistical classifier.

$$\phi_{fit} = \frac{1}{1 + e^{-c \cdot RMSE}}$$

Feature 14 is the most elaborate feature we designed for this project. In fact, we developed two different strategies for detecting handle-related asymmetry in objects. The first one is a 2D projection, Hough Transform-based feature which creates best-fit boundaries of an object's main body and typically does a good job in isolating handles outside of this boundary. The second technique, which is the one we decided to employ, is a rotation-based cylinder fitting algorithm using the 3D model's convex hull. Both of these algorithms double as handle classifying features and Mug-type object gripping point/approach vector computation functions and will be discussed in the gripping point computation section that follows.

### *B. Object Alignment*

We initially assumed that we could use as input any object regardless of its alignment in 3D Cartesian space. We developed two possible alignment techniques, both of which we decided to discontinue because of their shortcomings in dealing with asymmetric objects like mugs, and because the **.ply** objects we use are already aligned with the X, Y and Z axes.

The first alignment technique we employed was a custom iterative cost function algorithm which iterates through different rotation angles about the X, Y and Z axes in order to maximize the length of a major axis and minimize the length of a minor axis. For example, we rotate about the Z axis until we maximize the X-dimension and minimize the Y-dimension based on the following cost function.

$$J_{ALIGN} = \max(|x_{MAX} - \bar{x}| + |x_{MIN} - \bar{x}|) - \max(|y_{MAX} - \bar{y}| + |y_{MIN} - \bar{y}|)$$

*where:* $\bar{x}$ *is the mean x-coordinate*
$\bar{y}$ *is the mean y-coordinate.*

The second alignment technique we tried used the built-in MATLAB functions **princomp.m** and **convhulln.m**. We compute the convex hull of an object and then rotate it using Principal Component Analysis (PCA). This ensures that our major axis is the X-axis and our minor axis is the Z-axis. These functions in conjunction worked very well for symmetric objects, but not for irregular shapes so we could not rely on them when computing the object vectors.

*C. Support Vector Machine*

To perform the classification of objects we used **SVMLight** [1]. The objects were divided into three basic classes:

Class 1: Glass, bottles, juice boxes, etc.
Class 2: Plates and bowls.
Class 3: Objects with handles such as mugs and pitchers.

We assigned a class to each object and three SVMs were developed for classification of a new object. Given any new point cloud, we create the associated attribute vector and feed it into the classification tool in **SVMLight** for each of the three class identifiers. Each SVM determines if an object does or does not belong in a single class. For classification and testing, the **.ply** data is split into training and testing sets. Three-quarters of the data was put in the training set, while the other quarter into a testing set. The same data was put into each set for each of the three classifiers; however, when fed into **svm_learn**, the classification labels were reassigned to suit the appropriate class.

Given the set of three trained SVMs and the testing data, final classification was then done as follows:

1) If there is only one positive classification, then assign the new object that class, and proceed to compute the gripping point.
2) If there is more than one positive classification, then we assign the class of the object to be that which is "most positive". Specifically, the class of this object will be the one where the object lies deepest in the positive classification region.
3) If there are no positive classifications, then assign the class of the object to be that which is "least negative". Specifically, we have all three classifiers saying the object does not belong, so we assign it the class where it is closest to the SVM division.

For many objects that are misclassified, an acceptable grasping point can still be calculated. For example, while you would like to grasp a mug on the handle, if classified as a cup, the grasping point would be computed at the centroid, as discussed in *Grasping Point Computation.* Additionally, a cup or mug, if misclassified as a plate or bowl, could be picked up on the brim. While not optimal, these grasping points can still allow for the object to be picked up; however, the grasping points are either awkward or simply unintuitive, so the result can be considered a failure. We will discuss the results of the SVM classifications paired with the gripping point strategies in *SVM Results*.

*D. Grasping Point Computation*

As previously discussed, we designed a different manipulation strategy for each of the object classes we have defined. For the Glass/Bottle category, the gripping strategy is very simple since these objects are symmetric and have centroids easily approachable by an end effector. For the Plate/Bowl category, the gripping algorithm is non-trivial since the approach direction to the rim of a plate can vary with different types of objects. The most complicated gripping strategy was used to approach handles in the Mug/Pitcher category. Again, we attempted two different strategies and chose the one with the better overall performance.

Glass/Bottle Category:

The gripping point for this object class is simply the "center of mass" of the object. This is easily computed by averaging the X, Y and Z coordinates of all the points in the object. The gripper approach direction is parallel to the X-Y plane, but its direction within that plane is not important since the cross-section of these objects is symmetric. Therefore, we define the approach angle arbitrarily as the (normalized) difference between a point (60, 60) on the X-Y plane at the centroid height and the actual center point of the object.

Plate/Bowl Category:

For plates and bowls, we have already discussed that our optimal gripping strategy is not to reach the centroid of the object as this would be difficult due to the flatness of a plate or the curvature of a bowl resulting in a centroid that is not even on the object itself. This strategy assumed that any bowl or plate we consider is wider than its height, so we set the Z axis as the minor axis prior to gripping point computation. We then divide the 3D model into six (6) buckets spaced evenly in Z-coordinate value. We then find the maximum X-coordinate of all points in each bucket. The gripper approach direction is the vector defined between the *maximum* X-coordinate maximum and the *minimum* X-coordinate maximum. The gripping point itself is the point of maximum X-coordinate of the whole object. The schematic below visually represents this algorithm.

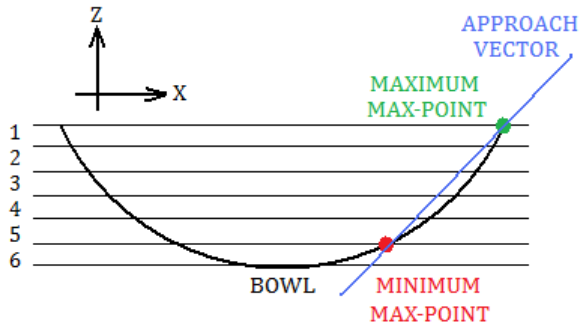


**Fig. 2: Plate/Bowl Grasping Strategy Schematic.**

Mug/Pitcher Category:

The first strategy we tried for handle grasping built on the Hough Transform algorithm. The idea behind this is that due to the higher point concentration on 2D projections of edges, we can use lines that pass through the most number of points to accurately represent edges. In the case of a mug, we could then separate the handle from the main body through these Hough Transform generated lines. The boundary line is generated by iterating through different values of **R** and $\boldsymbol{\theta}$ as shown in the schematic below. The origin of this line is simply the center of mass of the entire Mug object.

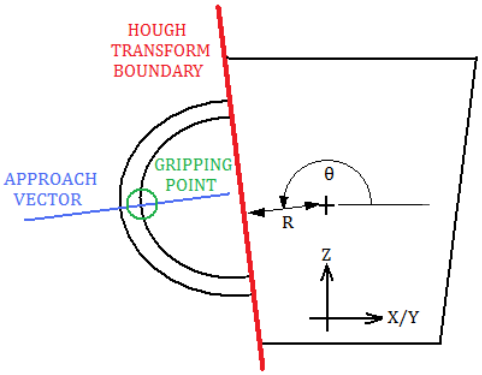


**Fig. 3: Hough Transform Mug Grasping Strategy.**

To find the grasping point with the algorithm above, we take the average of all the points outside the boundary (in the ideal case, this would be just the handle). The approach vector then passes through the grasping point and is perpendicular to the Hough Transform boundary line. Note that the line picture above is ideal, and while it looks like this for some objects, it does

not fully isolate handles all the time. This algorithm deals with handle rotation too, since it uses the handle symmetry descriptor to figure out which 2D projections (X-Z, Y-Z or both) should be used. Symmetry is defined with the ratio of points outside the Hough Transform boundary versus the total number of points. If symmetry is "low" in only one of the 2D projections, we use only the boundary for that projection. If symmetry is "low" for both X-Z and Y-Z projections, we look at points to the left of *both* boundaries, and the approach vector is a linear combination of the perpendiculars to both boundaries.

The second strategy we used to find the mug grasping point does not show any increased performance in directly computing the grasping point, but the algorithm runs significantly faster than the first one. Here, we rotate the top (X-Y) view of the mug until it is as symmetric as possible about the Y-axis – i.e. the handle points directly upwards or downwards in this projection. We then find a circle centered about the Y-axis which best fits the X-coordinates of all the points. This circle is then moved up and down along the Y-axis until the RMS error of the circle fit is further minimized. The center of the circle is kept track of and the entire structure is rotated back to original coordinates.

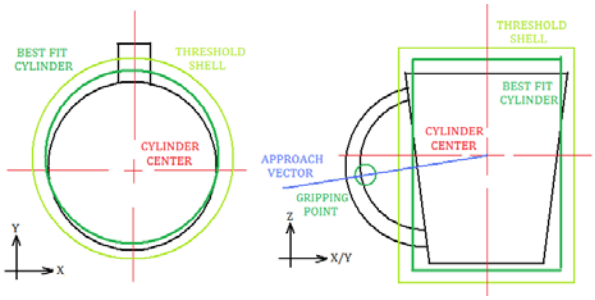


**Fig. 4: Cylinder Fit Mug Grasping Strategy.**
**a) Plan View (left) b) Lateral Elevation (right)**

To account for slight error in cylinder fitting due to the eccentricity induced by the handle, we define an additional threshold "shell" of points outside the cylinder which we still consider to be internal to the main body and not the handle. The gripping point is computed as the average of all the points outside this shell, and the gripping approach vector is defined from the gripping point towards the center of the best fit cylinder. The symmetry feature here is defined as follows, again with a design parameter **c** involved. We made **c** large enough for our purposes so that the feature is close to a binary value for good SVM classification performance.

$$RATIO = \frac{Points\ Outside\ Shell}{Total\ \#\ of\ Points}$$

$$\phi_{handle} = 1 - e^{-c \cdot RATIO}$$

*E. Code Structure*

While our algorithm was initially developed in MATLAB, the final product is a C++ class (**Classify.h**) which makes calls to both Octave functions and the SVM executable files. This allows for easy integration with any ROS controller wanting to perform object classification or grasping point computations.

To use the **Classify.cpp** and **Classify.h** files in a ROS controller, we must first include all the SVM and Octave files. The **svm_light** folder contains the SVM model text files created from our learning data set, as well as the **svm_learn** executable. All the Octave **.m** functions will simply be placed into the source directory of the controller.

At this point, any ROS controller only needs to work with our **Classify** class. A typical object classification would proceed as follows: The controller wants to compute a gripping strategy for an unknown object. It creates a new instance of **Classify**, where our Constructor will initialize the connection to Octave. The controller supplies the point cloud by calling **setPointCloud()**. In the **setPointCloud** function, we call the **getVector.m** function and write the output feature vector to file. The **svm_classify** executable is then run for each SVM model file; we have one SVM model file for each object class – mugs, plates, and cylinders. The **svm_classify** executable writes the SVM margin of the point cloud to three different text files. The **getObjectClass()** function reads these text files and returns the object class with the largest positive margin. Depending on the object classification, the functions **getGrippingPoint()**

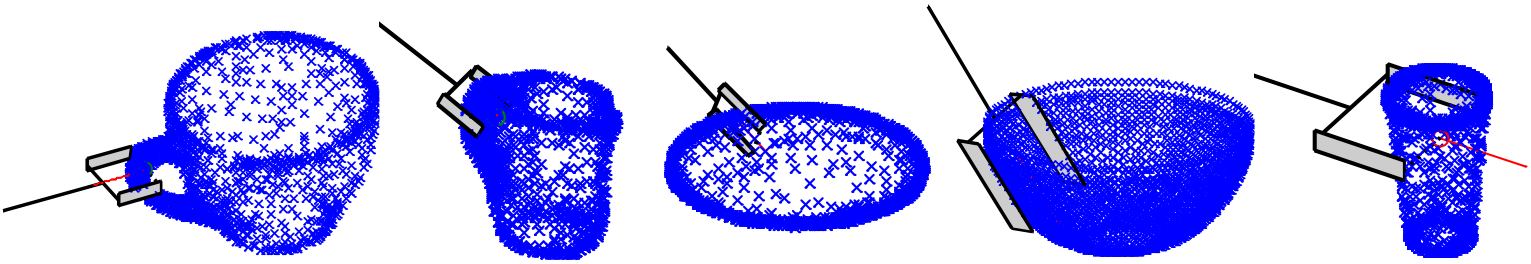

**Fig. 5: Sample gripping strategies for various objects in the test set.**

and **getGrippingVector()** call an Octave function (**findMugGraspingPoint.m**, **findPlateGraspingPoint.m**, or **findCylinderGraspingPoint.m**) to return the resulting grasping point and vector, respectively.

## III. RESULTS

### *A. Grasping Point Results*

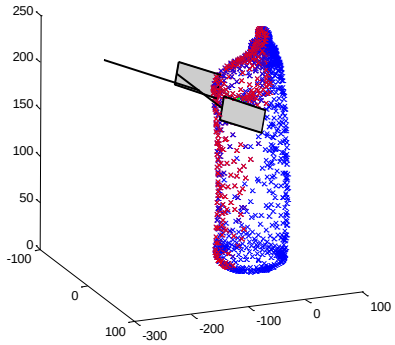


**Fig. 6: SVM Non-optimal gripping point calculations on misclassified objects. a) An acceptable alternative grasping point (above) b) An invalid grasping point (below)**

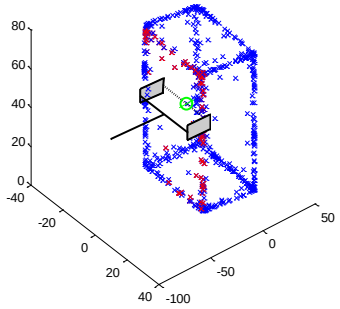


Fig. 5 above shows some gripping strategies for objects in our test set. Our category-dependent grasping algorithms performed very well, as all correctly classified objects displayed feasible gripper positions and approach directions in the MATLAB visualization. We cannot quantitatively assess how good these gripping strategies are, but all our 3D plots displayed viable solutions to the manipulation problem. The SVM classification performance was the bigger issue, as the gripping points and vectors calculated for misclassified objects do not give good results (as we will later see).

One object which is misclassified is a chemical bottle, shown in Fig. 6a. There are no objects in the training set that resemble this exact shape, although the classification clearly should put it in the same class as any bottle or glass with no handle. Incidentally, the resulting classification is that of class 1: mugs and objects with handles. Fortunately, while this is considered a failure in that our algorithm was unable to perfectly identify the object type and compute an optimal grasping point, the grasping point computed via the handle grasping methods returns a non-optimal point which would also be valid.

Another object that is misclassified as a mug is the box shown in Fig. 6b. Note that because our code tries to look for a "handle" in the box, the gripping point is close to the edge of the box and a real grasper would probably slip if attempting to manipulate the box in this fashion. Additionally, the approach vector is not perpendicular to the object's minor axis.

### *B. SVM Results*

A major obstacle in this project is the lack of "real" .ply object data. As a result of this, the SVM kernel used could not be overly sophisticated, as it would over fit to the training data and cause large testing errors. Over fitting the training data is not an issue for a linear or quadratic kernel.

When tuning constants and choosing an appropriate kernel, we examined five critical values. The training and testing errors are always important, but more importantly we watched the change in precision, recall, and F-Values for a

given SVM training setup. Precision is the ratio of true positives classified positive by the SVM versus the total number of examples classified positive, while recall is the number of true positives classified positive by the SVM versus the total number of positive examples in the data set. In our case, we strive to correctly classify all positive examples, since we don't want any objects to be rejected by all three SVM classifiers. This means that we weight a higher recall slightly more than precision when attempting to tune the SVM classifiers to optimize the grasper. Figure 6 shows the results of testing a linear and quadratic SVM classifier on the training and testing sets.

Proceeding with the three SVM model files output by **SVMLight** using the quadratic kernel, we compute the classification for an unknown object, and subsequently the grasping point. For our test set of size 35, 4 objects were incorrectly classified, for an 11.4% overall testing error on grasping point calculations. Most importantly, the objects which were misclassified using this method were objects which were under represented in the training set. Commonly found objects such as plates, bowls, cups, and wine glasses were all correctly classified by the three SVM classifiers.

| | Class 1 | Class 2 | Class 3 |
|---|---|---|---|
| Training Error | 11.22% | 2.04% | 10.20% |
| Testing Error | 14.29% | 0.00% | 14.29% |
| Test Precision | 63.64% | 100.00% | 93.75% |
| Test Recall | 87.50% | 100.00% | 78.95% |
| $F_1$ Value | 73.69% | 100.00% | 85.72% |

**Table 2: SVM Testing results. a) biased linear kernel (above) b) biased quadratic kernel (below)**

| | Class 1 | Class 2 | Class 3 |
|---|---|---|---|
| Training Error | 6.12% | 0.00% | 8.16% |
| Testing Error | 11.43% | 0.00% | 8.57% |
| Test Precision | 70.00% | 100.00% | 94.44% |
| Test Recall | 87.50% | 100.00% | 89.47% |
| $F_1$ Value | 77.78% | 100.00% | 91.89% |

### *C. Data Quantity and Format*

While our training and testing results were fairly good, the lack of available 3D model data at our disposal had a great effect on the results and depth of our work. We were limited to only 133 models from the .ply object database [2], and then had to divide these into training and testing data. We remedied this limited availability of .ply models by cutting down the number of object categories to three very distinct classes.

In addition, we decided not to utilize the original .3DS models we were working with because the data point structure is much less "real" than the .ply formatted objects. The concentration of points is not as uniform for .3DS objects like it is for .ply objects, and the general 3D point distribution is very sparse, especially for simple objects. While the .ply objects are formed from real data, the points on .3DS objects are structured to just define the vertices of the model. Therefore, for features such as straight handles, a typical .3DS point distribution would be virtually empty everywhere except the edges of the handle. Therefore, our point density-based features showed little success with the .3DS object format.

### *D. Object Alignment*

Another initial element of this project which we decided to discontinue for the final product was the notion of object alignment. The alignment techniques described earlier did not yield good results, particularly for asymmetric objects. Again, we encountered an advantage in using the .ply data format database because all objects were already satisfactorily aligned with the Cartesian axes. The .3DS objects were not, and even though we could align simple objects like boxes and pens, objects like cups and ladles could not be rotated to align with any of the major co-ordinate axes. If our work focused more on grasping of misaligned simple objects we would surely employ PCA alignment techniques.

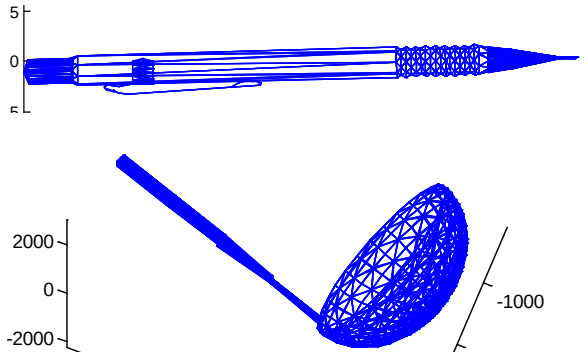


**Fig. 7: a) Effectively Aligned *.3DS* Object (top)**
**b) Misaligned *.3DS* Object (bottom)**

## IV. CONCLUSION

We developed a feature-based classifier for 3D point cloud models using Support Vector Machine techniques. The features were designed judiciously to create a 14-element vector that can quantitatively separate different object categories based on mathematical expressions for their unique characteristics. By training the SVM with a set of 3D models we can then introduce novel objects and classify them in a category that most accurately describes them. Then, based on the classification of this new object we can apply class-specific gripping strategies which can serve as a guideline for a real robotic manipulator device.

For future work, we plan on finding more .ply formatted 3D models so that we can expand our data set. This well-needed expansion will improve our SVM results and also allow us to introduce more object classes to describe objects. Other plans involve using the C++ class we have written and testing our gripping strategies on an actual robotic platform by interfacing with Robot Operating System (ROS) through the **roscpp** package. Another idea we had for further implementation was the use of apprenticeship or reinforcement learning instead of hard-coded gripping strategies. However, this would require the use of a robot or simulator in order to be able to evaluate gripping performance.

## V. ACKNOWLEDGEMENTS

We would like to thank Joachims Thorsten [1] for making available his SVM software SVMLight, which we used to train and test our feature-based object classifications. In addition, we would like to thank Pascal Getreuer [2] and Steven Michael [3] for their MATLAB code packages that allowed us to easily read and manipulate .ply and .3DS objects (respectively). Finally, we wish to thank Professor Ashutosh Saxena and the entire Personal Robotics group at Cornell University for helping us develop our work into the final product we have presented.

## VI. REFERENCES

1. Joachims, T. [2008]. SVM$^{light}$ – Support Vector Machine. <http://svmlight.joachims.org/>
2. Getreuer, P. [2004]. Read & Write PLY Files – MATLAB Central. <http://www.mathworks.com/matlabcentral/fileexchange/5459-read-write-ply-files>
3. Michael, S [2005]. Model3d – MATLAB Central. <http://www.mathworks.com/matlabcentral/fileexchange/7940-model3d>
4. Willow Garage Vault [2010]. PLY Object Database. <http://vault.willowgarage.com/wgdata1/vol1/wgdb/>
5. Architectural Home Design, 3D Models, Quality Textures. Online Interior Design. <http://www.archibaseplanet.com/>