Project Franklin

OBJECT DETECTION AND POSE ESTIMATION USING AN XBOX KINECT

SOLUTIONS IN PERCEPTION CHALLENGE 2011

Colin Lea
colinlea@buffalo.edu
Dan Molik
damoli@buffalo.edu

Kevin Yam
kevinyam@buffalo.edu
Lai Lee
lailee@buffalo.edu

ABSTRACT

The following report details an entry into the Solutions in Perceptions Challenge as part of the 2011 International Conference on Robotics and Automation (ICRA). The goal of this competition is to determine the pose and classification of household objects within a series of increasingly complex scenes using an XBox Kinect. Contributions are split between the *training* and *detection* phases of our system. Through comprehensive evaluation we ultimately found the best results using a table-removal-based object detection system, texture-based classification using the Scale-Invariant Feature Transform (SIFT) and pose estimation using Multi-Resolution Occupied Lists (MROL). Preliminary results show high classification accuracy and adequate pose estimation on the training datasets.

Introduction

Computer vision is increasingly important for applications in mobile robotics. The ability to determine information about your surroundings allows for better navigation and situational awareness. For example, if a robot is being used in a rescue scenario where it is trying to find humans in a burning building, computer vision techniques could be used to map the environment and detect people. The robotics company; Willow Garage, specializes in domestic robotics meant for home and business use. For a robot to interact in a home or office setting it is important to be able to recognize different physical objects. For example, in a nursing home a robot could be used to take care of an elderly immobile patient by performing tasks such as getting food or drink from an outside area. Vision is necessary to identify objects and avoid hazardous things in the way.

Another example is autonomous driving. A robot can recognize its surroundings so that it can safely follow a road. Moreover, the vehicle can notice visual indicators such as stop signs and speed limit signs and make corresponding adjustment. Autonomous driving systems will potentially be much safer than current manual driving.

Figure 1 The Microsoft Kinect is used to gather data from our environment. Our data comes in the form of three dimensional pointclouds and two dimensional color and depth images.

With the advent of the Microsoft Kinect, the cost of obtaining three dimensional data has been greatly reduced. Traditionally, costly stereoscopic cameras with on-board stereo-on-a-chip processors are used. These methods are not only expensive but give mediocre data quality [1]. Previous work by one of the group's members shows that the range error and interpolation distance are greater on a traditional stereo setup versus the Kinect [2]. The Kinect can output a three-dimensional pointcloud using a camera-projector pair. The low cost enables a

growing subfield of computer vision, due to the new availability of data. The ICRA Solutions in Perception competition showcases this combination in order to bring forth potential innovations of new pointcloud based techniques that further improve spatial accuracy. ICRA will continue this competition for the near future, progressively setting goals that are more difficult. The intention is to challenge participants to write algorithms to break current barriers of computer vision and robotic perception.

The two key technological requirements are performing object recognition and pose estimation using data from the Kinect, robot pair. *Object recognition* refers to correctly identifying what object(s) are in the scene. *Pose estimation* refers to determining the six degrees of freedom for pose of the identified objects. This includes complete position and orientation data. In the first round there will be one object on a table. The next three rounds get progressively harder by adding more objects to the world scene. The top three teams will then compete by running their system using a real robot that grasps the objects.

Some other challenges include: *occlusion*, dealing with objects that are not fully visible block; similar texture, *classifying objects* in a way that the algorithm will not be confuse with alike objects; *noise*, filtering data that is unimportant so that accuracies are improved; and object movement, *tracking* object as time varies because the objects are not stationary.

Before proceeding into further discussion of our system, it is worth showing imagery related to our final goal. In the following Figure 2, an object that we need to classify is placed on top of a turntable. We must extract the object, classify it, and determine its position and orientation.

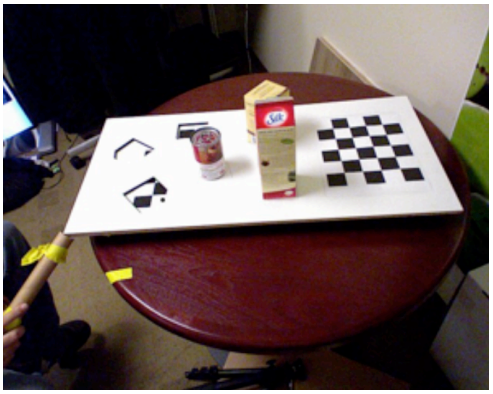

Figure 2 A representative samples of what one of the multi-object data sets will look like.

The team took an iterative approach to finding the best solution to this problem. A selection of algorithms were chosen and tested with comparative results. Ultimately we determined our best strategy to meet the objective of the competition:

Training - Learn new objects into the system
- Features Extraction - Using SIFT
- Model Alignment - Using MROL

Detection - Recognizing and determining object pose
- *Table Removal* using Singular Value Decomposition and Random Sample Consensus
- Feature detection using SIFT which matches the current scene model with models in our database
- Pose Estimation using of MROL

Individual interest varies from mild curiosity in computer vision to pursuit of a graduate degree in this area. General algorithm development and project management is applicable to all group members for continuing education and for corporate employment.

Competition Requirements

A set of rules can be found on the official Solution In Perception web page [3]. First, we summarize these rules and then go into detailed customer requirements and engineering specifications.

The setting of the competition is to imitate an office setting with indoor fluorescent light and object that is located 0.6 to 1.2 meter away. For the competition, the Willow Garage PR-2, see Figure 3,will be stationary but away from the rotating table with the distances mention

above. The algorithm is to be complied in C++ or python so that it is compatible with the robot's operating system (Robotics Operating System). Specifications include dual quad-core Xeons for a total of twenty threads, GNU/Linux with the Robotic Operating System libraries installed, along with numerous other sensors and manipulators a more detailed list can be found here [1]. The computational hardware used is designed to the purpose of this project: recognition of objects. Therefore, the hardware and the setup is optimized for the computing and running of such task.

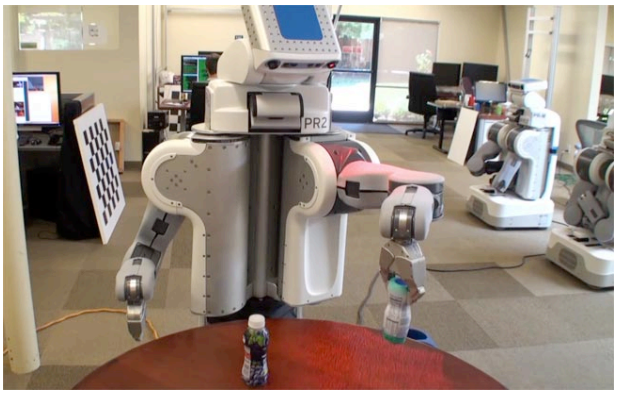

Figure 3 The Willow Garage PR-2 robot

Object recognition is done with a library of 50 known objects. Training data for 35 of these is available online. The data includes color imagery, point clouds, and coordinate transforms necessary for generating models for use. Classification requires learning information about specific features between models that help distinguish between different objects. Note that "learning" is implied in the machine learning sense -- the system learns what an object is, by running algorithms that extract features from the data set. The values returned are important from a statistical sense but essentially come down to a set of numbers.

Figure 4 The 35 objects from the released data set. We have been given training data for all of these objects. These data sets contain each of the objects on a turn table and we are given a pointcloud, imagery, and ground truth pose to train on.

Additionally, 15 new objects will be added at the time of the contest. The system must provide a way for the judges to train the robot to recognize the new objects. Potential challenges include merging this new data into our classification system. New data sets are used to determine if the learning methods are accurate.

The competition takes place at the ICRA in Shanghai, China on May 10th and 11th 2011. While we ultimately aren't able to go to the conference, we will be able to compete. A colleague from CSIRO (Commonwealth Scientific and Industrial Research Organization) in Australia will be at the event and will be presenting on our behalf. Our code was officially submitted and accepted on May 1st. It has been stated that there are six other teams competing aside us.

Customer Requirements

For any products in the market there is always a product life cycle that concerns the useful period time a product can be used. More importantly, this cycle maps out customer groups where surveys can be used to obtain customers' requirements. One of the customer groups for this product is the competition judge, and their requirement is already laid out in the competition guidelines. The second group of customer is the end user, who will try this product without prior knowledge of computer programming. It is assumed that the robot will have to be easy to use. "Easy to use" is interpreted as simple buttons that serve obvious functions. These customer requirements, technical specifications and target settings are shown in the table below.

Customer Requirements	Technical Specs	Target Setting
Learn new objects	Is there a function to add new objects	Yes (and it shouldn't greatly affect runtime)
Identify Objects	Classification accuracy	Maximize the percentage

Find position of the object	Positional Error Tolerance	+/- 2 Centimeter
Fast processing speed	Processing Time	Less than 15 seconds
Works with necessary operating system	Robot's operating system. Programming language.	Robotic Operating System (ROS). Language: C++ or Python
The system should be easy to use	Number of steps to train new object. Number of steps to delete old object. Number of step to run recognition program.	A button to learn. A button to delete. A button to recognize and pose estimate.
Low financial and computational cost	Cost of equipment (hardware) Computational cost of algorithms	Lower in cost compared to stereoscopy Minimize computational cost while retaining classification and pose fidelity

Figure 5 Customer Requirements and Technical Specification.

Learn new objects
The computer will have to learn new objects and be able to recognized and identify them in the future. There millions of objects in this world and pre-loading all of them into the robot would be an expensive and futile exercise in computation and storage optimization. Therefore an alternative solution is used; an algorithm that will allow the robot to learn and recognize a new object. The robot will essentially scan the object on a flat table in a room setting. It will then store these new objects for future identification. Also, it will have to store this information in a compact size so that the memory footprint is minimized, effectively increasing the amount of objects it can recognize.

Recognition of the object
Recognition of the object should be accurate such that the object is classified with a set amount of certainty. Ideally, the object must be identify with near 100% accuracy. A central objective of this project is for the robot to find the specified object and identify it, otherwise the algorithm is effectively unimplemented.

Position of the object
The position of the object relative to a world coordinate frame must also accurate, with an error of tolerance of +/- 2 centimeter. This is another objective the project must satisfied as per point distribution in the competition rules.

Processing Speed
The algorithm will respond in the least amount of time possible. Ideally, it should response within 15 seconds. Anymore time and the robot should automatically end the iteration and output an error message, and assume the algorithm should be restarted.

Compatible
The algorithm is written in a language that is compatible with the operating system. It is our prime concern that the algorithm is fully compatibility with ROS. The programming language used should be C++ or Python.

Easy to Operate
Due to the numerous times the program has to run, the time and steps it takes to operate the algorithm inside the robot should be simple and fast. Therefore, the number of steps the robot takes to learn new object is minimized and the number of steps to run the recognition program is minimized. Ideally there should be a command to learn, and a command to recognize object.

Computational complexity
In this aspect, we want to optimize functionality of this computational algorithm without increasing cost. In other words, a program with more or better functions often come with cost of longer run time. Therefore, it is essential in finding a balance in between them.

Problems and Challenges

Unique features of object
Our classification method recognize the object according to it's characteristics. It is direct, straightforward to distinguish between a circular and rectangular object but it is more difficult to distinguish between a set of cylindrical object with different dimension. Therefore, unique features over a set of similar objects must be found in order to get an accurate classification.

Occlusion
Obscured objects will need to be addressed in order to be awarded the most possible points in the competition. On primary examination, some sort of data extrapolation would be a method to fill the data set to be able to use the same algorithm method as before. This could lead to erroneous data that is most likely misleading, a preferable method would be to simply preform a modified sampling, and have a weighting parameter to determine the degree of confidence that a partial data set provides. Of course programming ease and solution accuracy could lead to a different solution entirely.

Noise
Misleading data that is captured from varying lighting and changing background, which affects the results of the output. This input data will have to be treated so that it does not carry the error to the result. This process is known as filtering, where some of the data is minimized and some of the data are amplified in the effort to improve accuracy.

Potential and Planned Approaches

For our competition, there is a hierarchy of potential directions. On a high level we can tend towards a two dimensional or three dimensional pipeline while working with data, as described in the next section. On a lower level, we must consider a variety of different approaches for each sub-section of our project, which are *Segmentation, Pose Estimation, Classification,* and *Tracking*. In the following two sections, we discuss some potential solutions as well as our current planned path.

High Level
On a high level, we much choose between two-dimensional and three dimensional data pipelines. A two dimensional pipeline implies that we are working directly with images while a three dimensional pipeline means we are using spatial "pointcloud" data. A pointcloud is a datatype that stores the three dimensional location and color of all points in an environment. By working with images, we can leverage many computer vision tools such as OpenCV (Open Computer Vision library), NumPy (Numerical/Matrix Library) and SciPy (Scientific+Vision Library). Because we have these tools available we can prototype different methods faster, without having to implement each algorithm directly. Because the idea of using three-dimensional data is relatively newer there are fewer tools available.

Ultimately, we chose to take a hybrid approach that uses two-dimensional techniques for classification and three-dimensional techniques for pose estimation.

Low Level
Segmentation is the process of separating image pixels into different groups. The objective is obtaining a representative output for next step: classification. We have a wide range of choice in segmentation method: histogram-based, part models, active-contour models, and others. We started by tending towards region-based algorithms because of their tendency to keep neighboring pixels together. Later we show that other methods such as histogram-based processes can have problems where different areas of an image are segmented as one area. Ultimately we employ a table-based segmentation method which works predominately on three dimensional data.

Pose estimation is the process of determining the position and orientation of an object. Our original approach involved an iterative method that starts by matching primitive shapes to determine pose. After classification, a more complicated model would be used to refine the pose to increase robustness. Ultimately only a single iteration was performed using MROL due to computational complexity. It was not possible to run this multiple times within time constraints.

The crux of the competition lies in the actual **classification**. The ability to accurately define an object requires detecting notable features and comparing them to our list of potential objects. Recent work shows that matching specified target objects in a scene can be computed with high accuracy using SIFT, SURF, and other feature detection techniques [4]. Using image-based techniques, we must be able to match what is in our scene with a similar representation of known models in our dataset. We create a database that stores information

about each of our objects with corresponding image and feature data for all angles. We are able to attempt to match our current segmented object with the potential objects in the database. Approaches using ensemble classifiers such as random forests or bagged decision trees were investigated but not necessary in the end. Some other methods of accomplishing classification include using parts models, three dimensional model-matching, grammar models, and simple feature extraction. A Scale-Invariant Feature Transform is used in our final system.

Tracking is used to improve the accuracy of position data over a period of time. The position data measured from the Kinect carry noises, which gives an uncertainty of positions. Using a tracking method, the noisy measurement is recorded and calibrated to a new position, which will be closer to the actual position. Tracking is useful in that it allows the robot to know where the object is at all times. There are two approaches this solution, linear and non-linear. A Kalman filter is a linear approach where it assumes that the noise variation is under the assumption of Gaussians distribution. This is favorable because any system that can be simplified to a linear solution is much simpler to work with. The non-linear approach involves the uses of particle filter which provide better accuracy but much more complicated since it deals with random noise variations. Ultimately the use of Kalman filter is preferred because the accuracy between a linear approach and non-learn approach is relatively small. One advantages of using a Kalman filter is that it only requires two steps to do tracking: prediction and update. The system measures the position of the object, predicts the path it will travel, and measures the position again. This process repeats and becomes more accurate as the time step decreases. Unfortunately, due to changes in competition rules, this work was not included in the final system. Regardless, we show results from our preliminary implementation.

Detailed Design

As previously discussed, our software is divided by two major components: Training and Detection. Each of these plays a very different role in the whole picture but works using mostly the same algorithms. In this section we discuss our final approaches in detail and also talk about system-level and other related challenges inherent in our design.

Training:

The training phase is split into three main sections: background removal, texture extraction, and model generation. The underlying premise of this phase is that we are trying to store a model of all of our 35 objects. We want to be able to model each of the objects in both two dimensional and three dimensional representations. Both geometry and texture are important for our detection algorithms, so we must extract them from the input data. The input is our initial data set which includes a colored pointcloud, a two dimensional image, and the ground truth pose. The output is a numerical description of the texture and a complete pointcloud of our object.

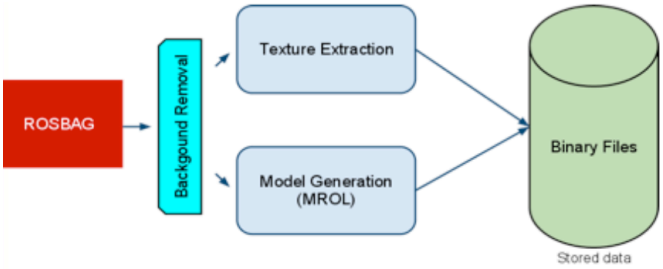

Figure 6 Algorithm pipeline, for use with training new objects.

Background removal:

In order to classify each of the objects in a data set, we must first detect which parts of the scene contain an object and which are parts of the background. Because we are given the ground truth position and orientation of each object in the training data we can easily eliminate the background. Mathematically, this results in transforming all of the points in the scene by the inverse of the ground truth pose. After this transformation the object is at zero position [xyz=0,0,0] and is orientated directly upwards. We simply use a box filter to isolate the object, which eliminates all points outside of a specified bound. In our case we used bounds reflecting the largest object size: x=+/- 0.15 m, y=+/- 0.15 m, z=+0.3 m.

Figure 7 A single projected frame from a data set

See the section on coordinate transformations to get a better understanding of how this procedure works.

Texture Extraction:

In our case, texture is defined as structured change of color in an image. We want to be able to compare the texture of an object in the scene with all of the textures in our database in order to perform object classification. Our biggest problem is that we will see objects at different positions in the scene and at different orientations. Using basic image recognition techniques such as template matching, if we try to match two objects at the same orientation, if one is farther back in the scene than the other then they will appear as two distinct objects. Even if we extracted an object's color image at multiple scales we would still have a problem matching at different orientations. The input data is spaced by 10 degree rotations, so we do not have enough information to generate color at all angles.

Over the past decade David Lowe's work on the SIFT [4] has garnered a lot of attention for it's ability to do image matching. It detects salient features in an image based on grayscale texture. The following figure shows how a stop sign, which can be considered a complex pattern, is matched between two images even though the lighting and angle are not exactly the same.

Figure 8 SIFT matching demoed on a common stop sign.

SIFT works by finding the maxima and minima on a Difference of Gaussians [4]**.** DoG is a method where multiple versions of an image that are blurred and sampled at different resolutions are convolved with each other. By only looking at the extrema the most distinct sections are kept. This algorithm is also rotationally invariant due to the way the descriptors are stored. There are 128 numerical values stored that represent cues surrounding the feature. These are stored starting at the dominant orientation surrounding the point and traverse clockwise. This generalizes the descriptor so that it can work at varying orientations

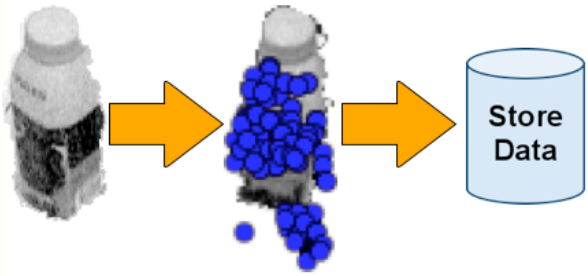

Figure 9 SIFT feature extraction pipeline

There is an efficient C++ implementation provided by the author which has been used. Note, however, that the matching algorithm has been implemented by us. This is discussed in the feature matching part in *Detection*. The previous picture depicts the process of taking the extracted image, finding the SIFT descriptors (the blue circles), and storing them in a database. All of the keys from each viewpoint in the data set are merged together into one file per object.

Model Generation

Each frame from a data set represents a different viewpoint of the object. In order to generate a complete model we must merge the pointclouds together. Given that we have the ground truth pose, in theory this shouldn't be a problem. However, through experience we found that the models did not line up particularly well. The following image shows the resulting pointcloud after simply merging all of the pointclouds together. There is a lot of noise, which would throw our pose estimation results off.

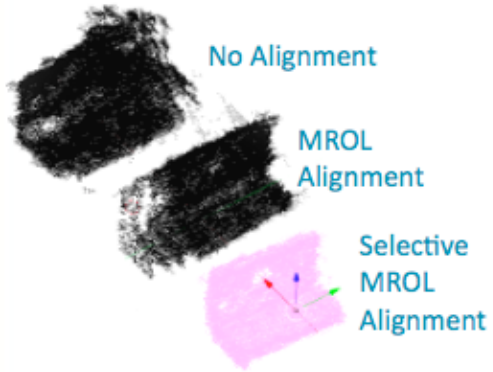

Figure 10 MROL model generation and refinement.

To compensate for this noise we use MROL, developed by colleagues Julian Ryde and Nick Hillier [5]. This is six degree of freedom (three position, three orientation) localization method using a probabilistic multi-resolution approach with occupied voxel lists. Note that voxels are the three-dimensional equivalent of pixels. They can decrease the amount of space necessary for storing sets of points by a large amount while only slightly decreasing the accuracy.

As shown in the image, just using the matching algorithm explicitly does not provide great results. It does a better job than simply merging the points together but there is still significant noise, especially in the Y direction. By selectively adding new pointclouds we were able to achieve much more accurate results. This works by calculating the deviation between the centroid of a new aligned pointcloud and the centroid of the combined pointcloud. If the deviation is too large (greater than 0.5 cm) then the object is not merged. A similar procedure is used for rotation. Figure 11 shows the output of the first 10 objects from the data sets.

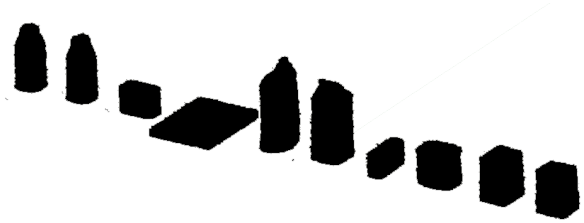

Figure 11 Objects 1-10 out of the 35 objects in the data sets

Detection

The underlying algorithms in the detection phase are the same as for training, but the way they are used is different. Instead of generating new models, we are able to classify and determine the pose of objects in a scene. Aside from SIFT and MROL we have developed a method for eliminating the background by determining the location and parameters of the surface that the objects are stationed on. This algorithm has gone through several different iterations. The process is documented in more detail in the *Segmentation* section later.

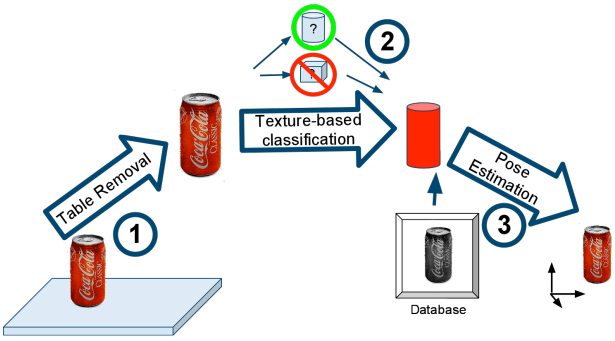

Figure 12 Detection pipeline for classifying object(s) within a scene

Table removal

It has been documented in the competition rules that the objects will be located on a flat surface. From all of the example data sets we see that this surface is the dominant plane in the scene. It was hypothesized that if we can identify and mathematically define the table then we can remove it from our scene.

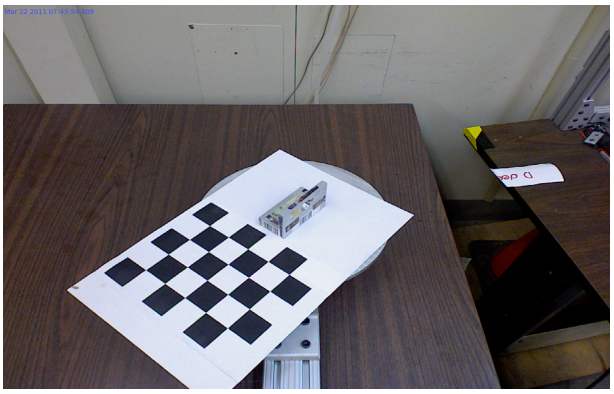

Figure 13 Input image

Figure 14 Segmented image used as input in SIFT

In order to mathematically model the table we must be able to locate it in the scene. A plane is defined by Ax+By+Cz=D where x, y, and z are any points in a scene and A, B, and C are defined by the surface normals. D is the intercept in our case will be set to zero. Given our three dimensional pointcloud we are able to estimate the surface normals for each point. This is calculated using Singular Value Decomposition (SVD) using a set of neighboring samples. SVD outputs a factorized set of matrices that split an input matrix into two orthogonal matrices and a scalar matrix. Importantly, the second orthogonal matrix can be used to obtain the surface normals for each patch. This is discussed in greater detail in the *Segmentation* section. In Figure 14 the color blue represents areas that have been removed, yellow is the isolated object, white is area without valid depth data, and red is data above the table but outside of our bounds.

Texture-based classification

Once the foreground has been extracted we must determine the number of objects in the scene and each objects' identity. Similar to the training phase, we use SIFT to extract texture descriptors from the scene. These are compared to the corresponding descriptors for all of the trained models. SIFT matching is performed by calculating the Euclidean distance between the descriptors in two models,

$$D = \sqrt{sum((A-B)^2)},$$

where A is the scene descriptor vector, B is a model vector, and D is the distance. The problem is that there are 128 numbers per descriptor and there are approximately 1000 descriptors per trained model. This takes a long time to compute!

The Lowe paper [4] suggests an alternative metric rather than Euclidian distance which is much more computationally efficient but slightly less accurate. Essentially, instead of taking the sum of the squared differences, the angle between the vectors comprising the 128 numbers can be calculated. This requires taking the inverse cosine of the dot product of the two vectors,

$$D = \cos^{-1}(A \bullet B).$$

While an open source method for doing this is available online, it is not computationally efficient. Running this code in our system resulted in run times of over 15 seconds just for the detection phase! We implemented this algorithm in Python and vectorized it using the numerical library Numpy for far superior performance. We were able to get approximately 1.8x the performance.

Figure 15 SIFT feature matching for "Odwalla" juice bottle

For single object detection, Maximum Likelihood Estimation is used to calculate the most likely object based on the SIFT features. This is mathematically described as:

$$\theta_{estimate} = \arg\max_{i} L(X \mid \theta_i)$$

where θ_i is an individual model description, X is the current scene description, and L denotes likelihood.

The value of the most likely solution is used as the basis for multi-object detection. Currently a threshold is used, so if any object is at least 70% as likely as the most likely solution then it is added as a potential object. If more time was available, parameter estimation techniques could be used to learn what percentage is optimal. This requires taking a large number of data sets, determining the minimum likelihood for successfully classifying all objects in the scene, and determining what threshold minimizes the error in classification.

Pose Estimation

The most likely models are fed into the MROL technique that has been previously discussed. Using our classification technique, we calculate an estimated centroid for the object based on the midpoint of the descriptor locations. MROL uses a conjugate gradient method to attempt to align the current scene with the stored models. The change in position between our aligned position and the initial guess is used to determine the pose in sensor coordinates. Figure 16 shows an example of an "Odwalla" juice bottle found in a scene and compares it to a model from the trainer.

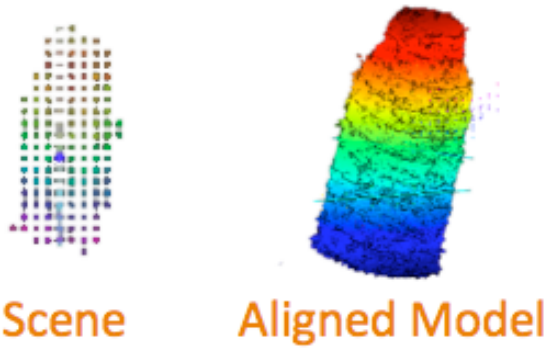

Figure 16 MROL to SIFT matching for classification of an object

The scene pointcloud is much more sparse and thus it can be hard to obtain an accurate alignment. Figure 17 shows how the algorithm does not always find a good match. There are additional problems when the scene is entirely flat. Sufficient geometric texture is required to accurately align the model.

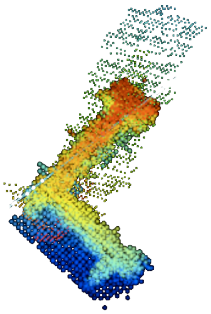

Figure 17 Model alignment.

Supplemental Challenges

Aside from the primary classification and pose estimation techniques, several other areas must also be accounted for. In order to actually use the aforementioned algorithms there must be a way to interface with the Robotics platform. In order to simplify prototyping, there must also be a way of extracting information from the robotics platform to use in programs such as Matlab. Additionally, many of the data interfaces that we're using are in different coordinate systems or use different notation. Finally, we discuss tracking, an area that received significant attention until a change in the competition rules mid-way through the semester.

System

All of the code for the final competition runs in Linux using the Robot Operating System (ROS). In order to compete we must be able to interface our algorithms with this software and output our results to Comma Separated Variable (CSV) text files. This required spending a lot of time learning how to take the data they give us (pointclouds, images, pose data) and be able to actually use it. There are hundreds of lines of "wrapper" code, which takes our algorithm and allow it to work as a complete system. The following figure shows a high level overview: we must input the ROS data into our training and detector code and output the final results to a separate file.

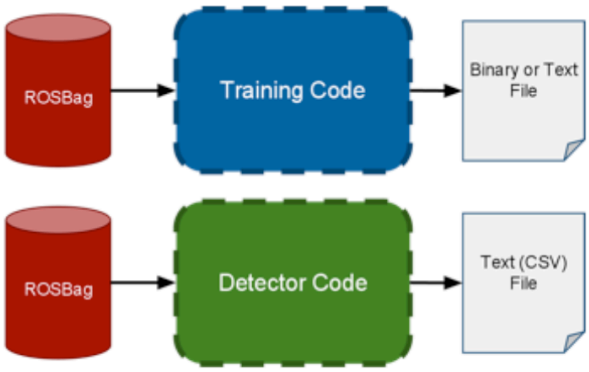

Figure 18 High level system pipeline.

Additional work was done to allow us to use the data outside of the ROS architecture. A batch convertor was created which inputs the ROS data and outputs the pointclouds, images, and poses into multiple formats that we can use directly with Python and Matlab. This process is shown in the following figure.

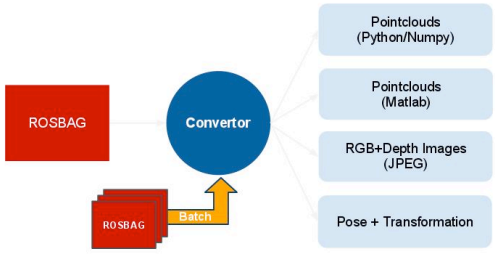

Figure 19 Written utilities for data extraction to enable algorithm simplification

Coordinate Systems:

There are 2 primary coordinate systems: the Kinect frame and the object frame. It is possible to switch between each of the frames using a set of transformations. For our results, it has been determined that the position of the object should be calculated relative to the Kinect frame. For the final round of competition, it is necessary to find the exact object pose so that a robot can uses its arm to move the object. The transformation equation is shown below along with Figure 20 to demonstrate the relationship between the two frames. T is the transformation from the Kinect to the global center position of the object and R is the relative transformation between frames. While the object rotates on the turntable in each data set the T matrix stays static and the R matrix changes every frame to reflect change about the Z axis.

$$Pos_{truth} = inv(T*R)*Pos_{kinect}$$

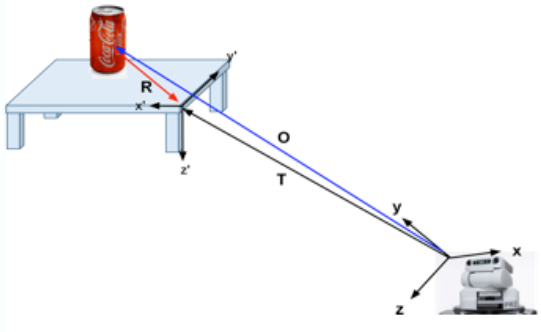

Figure 20 Relationship between Kinect frame and object frame

Additional coordinate problems arose from the use of different coordinate representations. The rotation in MROL uses a vector of length 3 with axis-angle notation. The input into our system is in Rodrigues notation and the output is a 4x4 transformation matrix. The lack of documentation regarding which notation was being used at each step caused trouble. For example, it was assumed that MROL and the input were using the same notation because they are both vectors of length 3. We later found this not to be true.

Tracking:

The competition is in its first year, so it took a couple months before the rules were completely finalized. Originally the robot was supposed to move around a stationary table that had a set of objects. However due to a sudden change in rules, the new set up uses a stationary robot with a table that rotates the objects. In the original setup we anticipated the need to use tracking techniques to refine our pose estimation. Significant work was put into this area, but ultimately tracking lost its important due to a change in rules. Nevertheless, Fig. 21 below shows a graph using our implementation of a Kalman Filter for displacement versus time in X, Y, and Z directions. A Kalman filter assumes there is noise in our measurements and improves accuracy of the expected positions. It assumes there is a "hidden" true value that is being corrupted by the sensor.

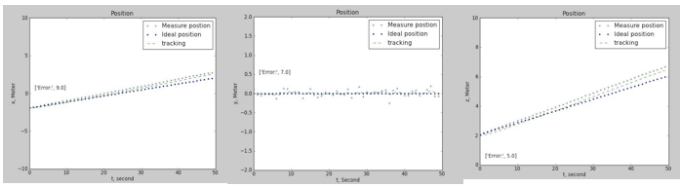

Figure 21 Displacement as a function of time in x,y,z coordinates

Iterative Design Approach

The problems that we are trying to work with cannot be considered "solved." This area of computer vision is being widely researched because the current state of the art is not adequate for real-time applications such as in robotics. While there are some directions for object classification and pose estimation that we have looked towards, there is no known "best" solution. As such, this project required a lot of trial and error. We used an iterative approach with each section until we were satisfied.

Additionally, In efforts to investigate a larger variety of algorithms, we decided to have two team members prototype methods exclusively in Matlab. This leveraged prior experience that these people had doing engineering simulation without requiring them to spend inordinate amounts of time learning a more "complete" programming language.

Segmenting the object(s) from the background imagery is a difficult task because of the complex textures that each object has. We went through three distinct methods of segmentation including two image-based methods and one geometric method using 3D pointclouds. In the end, our final table segmentation technique worked the best. In the following sections we detail each of the methods and explain (1) why we investigated the particular technique and (2) the final results and why we either used it or tried something different.

Method 1: K-Means

The algorithm we initially attempted was the K-Means method of segmentation. It is an efficient method for separating raw data into different groups according to similarity [6]. Since the intensity of pixels in an object is close, we considered this method to be useful for segmentation. Additional reasons for choosing K-Means were its simplicity and computational cost savings. In short, the procedure for K-Means is as follow:

K-Means Algorithm:

1.) Initially the data is randomly selected and the centroid is calculated. (data with 2 properties gives a two dimensional centroid and data with 3 properties gives a three dimensional centroid)

2.) The number of centroid returned depends on the number of group inputs.

3.) The distance between the centroid and each data point within a group is calculated. If the distance exceeds the tolerance, it will be excluded from the group.

4.) The procedure is repeated as a new centroid is calculated. The iterations will be stop if most of the data is assigned to a group.

Testing

We created our program based on the procedure described above in Matlab. After several modifications and improvements of the program, prototyping results can be seen in Figure 22.

Simulation Results

Figure 22 k-Means simulation results

The simulation results were not expected and do not satisfy our requirements for segmentation. Figure 22a is the original input depth image and Fig. 22b is the output image of K-means method. An ideal output should be similar to Fig. 22c so that the shape of a bowl is clearly extracted from the background. However, the output from K-means extracted the shape of the bowl with some part of the table. We believe that it is because the pixel in the lower part of the table share some similarity with the bowl. As a result, we looked at alternative methods to obtain better results

Method 2: Watershed Algorithm

Next we tried a watershed segmentation algorithm. Since the image needed to be divided into different segments instead of simply separating it from the background, this algorithm is more suitable than the K-Means algorithm. However, the procedure for the watershed algorithm is more complex than previous algorithms because it requires additional preconditioning of the input image [7]. The following is the procedure for watershed segmentation:

Watershed Algorithm:

1.) The input image (either depth or RGB image) is converted into a binary image. A binary image is an image format with only black and white pixels.

2.) The transformation mentioned above is often achieved by using threshold segmentation.

3.) A distance transform of the image is the next step. This transformation labels each pixel with the distance from the nearest non-zero pixel.

4.) Several local minimum points are chosen based on grayscale intensity.

5.) A "flooding" process starts from those minimum points by repeatedly increasing the intensity of the neighboring groups of pixels. During this process, a segmented line is formed by any pixels that are merged together.

6.) This line is used to construct the final segmentation.

Testing

We used Matlab's built-in watershed function [8] for testing. However, a tailored algorithm was written for the distance and binary transforms. The result can be seen in Figure 23.

Simulation Results

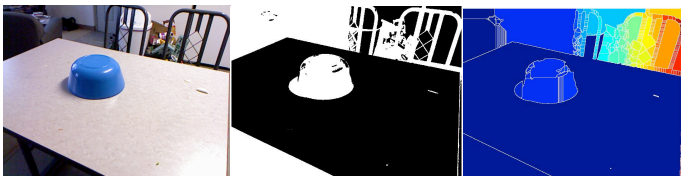

Figure 23 (a) RGB (b) Binary (c) Segmented Image

From Fig 23, we can see that the bowl is clearly distinguished from the background. Therefore, it is considered successful. However, with the multiple objects seen in Figure 24 in can be shown that the result is slightly less clear than the single object seen in Figure 23. The overall results were not acceptable since different objects can not be classified. In addition, it was discovered that the segmented result of watershed algorithm depends heavily on the binary image. In other words, the binary transform must be reliable to ensure the quality of the segmented results.

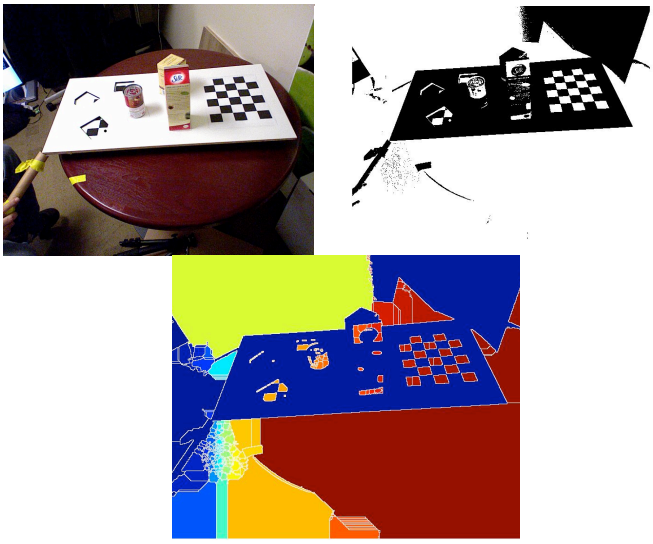

Figure 24 Multi-object scene (a) RGB (b) Binary (c) Segmented Image

Method 3: Table removal

The watershed segmentation has superior performance over k-Means, but ultimately doesn't suit our needs. We later realized that if the table is first extracted from the image then we will be able to easily extract the object.

The table extraction can be done using Random Sample Consensus (RANSAC) where we input points that have normal vectors pointing upwards I the scene. The normal's can be calculated from the depth image by using Singular Value Decomposition (SVD). As stated before, RANSAC can be used to compute the parameters of a mathematical model to the data [9]. The actual model in our case is generated using Ordinary Least Squares (OLS) by finding the minimum error about a linear plane. The advantage to using RANSAC is that it will find the dominant plane in the entire data set. If OLS was used on the entire pointcloud then the result would be very noisy. The following procedure details the RANSAC method:

RANSAC Algorithm

1.) Random data points are selected.

2.) The parameters of a mathematical model (such as a plane) are estimated based on the selected random data using Ordinary Least Squares. Specifically a three dimensional plane was used as a model.

3.) Additional random data points are selected. Then the difference between the current math model and the new data points are computed.

4.) Only the data points within tolerance will be included and a new math model will be constructed with the additional data points.

5.) The procedure iterates until the math model can represent a defined proportion of the data.

Since the table is a flat surface, all the normal vectors of it must point in a similar direction. Therefore, data points with distance close to a plane represent the table in an image.

Testing

We implemented the above method for RANSAC in Matlab. At first, the run time was too slow for final implementation. By changing the code to utilize vector and matrix based math, the run time was greatly reduced.

Simulation Results

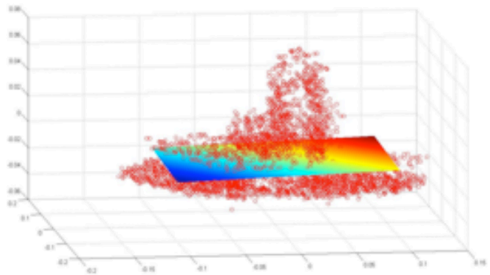

Figure 25 Data points in three-dimensional space and the found plane model

In Fig. 25, the red dots represent the normal vectors of a surface in the image. It is obvious that the set of red dots in the lower region represents the normal vector of the table. As a result, a linear plane model is produced around that region by using our implemented RANSAC algorithm. We were able to extract the table from an image based on these computed parameters (coefficients of a linear plane equation). Figure 26 represents the visual output of the modeled table. The yellow colored located in the center is the found table.

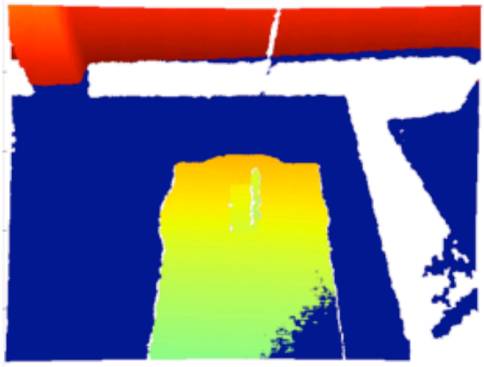

Figure 26 Found table in training data

System Results

While we will not know the end results until the competition on May 10th, we have done preliminary testing on data sets provided by Willow Garage. As a whole, the results are higher than expected.

On the high side, the accuracy of the classification is up to 100% for most of the object on training data. Figure 27 displays these result for 35 objects. From this figure, there are 15 items with an accuracy of 100%. Twelve items have 80% accuracy and there are 27 out of 35 items have accuracy higher than 80%. The average accuracy is 82%. It should be noted, however, that this is all calculated based on the same data sets that we *trained* the objects. This means that on *testing* data we most likely won't get this high. This is because the features we're detecting from these datasets are the exact same ones that are in our SIFT database. In reality the features will be slight variations on what we current have stored.

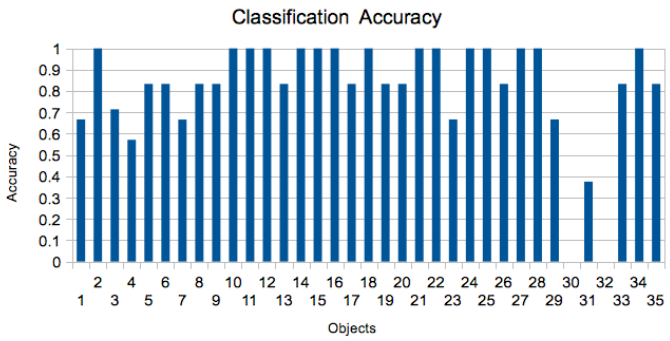

Figure 27 Classification Accuracy of 35 Objects

In regards to pose estimation, based on a small sampling of data sets the position accuracy is roughly 2cm to 5cm. In the competition anything below 2 cm gets full credit

and anything between 2cm and 5 cm receives partial credit. The rotation accuracy is harder to quantify. It has not be stated exactly how rotational accuracy will be determined. Because of different coordinate systems and way to measure it, we cannot determine how much credit we will likely receive. Visually, it looks like our orientation is in range of acceptable values (there is partial credit up to 90 degrees off axis). In general, our orientation is off by +/- 45 degrees. The running time of our system varies but is approximately 15 seconds on average. According to the judges, as long as the system run time is not significantly more than 15 seconds then full credit will be received.

However, our system does have significant problems in some areas. The classification accuracy for scenes with multiple objects is bad as 5% sometimes. Sometimes this is because there are objects on the table that are never trained on our system; these are here to trick us. There are other problems when the Kinect is oriented on its side, as shown in Figure 28. We believe that this has to do with the surface normal calculation done in the object detection phase.

Figure 28 Detection on a "sideways" data set

In both single and multi-object scenes, if our detection method fails then the classification and pose are usually completely wrong.

Conclusion and Future Work

The product has achieved a complete system capable of our the main objective: object recognition and pose estimation. While there is significant variation in result performance, most of the requirements mentioned before were met or below target. Since most of the material in this project is new to group members, we learned that testing is necessary to evaluate results of any conceptual approach. This led to an iterative approach which allows you to try many ideas but ultimately increases the development time.

Going forward changes to the system pipeline will have to be made to increase algorithm performance and to provide a set of filtering schemes to increase system performance.

There are several high level future improvements that may help obtain better results. Firstly, we can improve the poor classification accuracy of object in multi object scenes by using an alternative classifier pipeline. This new set of classifiers will combine additional method to avoid mismatching in order to improve overall accuracy. Additionally, we can decrease the run time of our system by migrating code to multi-threading and by incorporating more code optimizations. This can be done by either programming for multiple CPU cores and or massive parallel processing on a GPU. Since a GPU has many more cores than a CPU it can spread the computation between its cores to get a faster performance.

While the final results are not yet know, the experience gained by working on this project has been instrumental in increasing abstract problem solving skills, written ability, and analytical thinking abilities.

References

[1] D. Scharstein, R. Szeliski, and R. Zabih, “A taxonomy and evaluation of dense two-frame stereo correspondence algorithms,” in Stereo and Multi-Baseline Vision, 2001. (SMBV 2001). Proceedings. IEEE Workshop on, 2001, pp. 131 –140.

[2] (In review) Wong, U., Morris, A., Lea, C. et al. “Comparative Evaluation of Range Sensing Technologies for Underground Void Modeling.” International Conference on Intelligent Robots and Systems 2011.

[3] “Solutions in Perception Challenge.” Willow Garage: http://opencv.willowgarage.com/wiki/SolutionsInPerceptionChallenge

[4] Lowe, D. G., “Distinctive Image Features from Scale-Invariant Keypoints”, International Journal of Computer Vision, 60, 2, pp. 91-110, 2004.

[5] J. Ryde and N. Hillier. Alignment and 3D scene change detection for segmentation in autonomous earth moving. ICRA 2011.

[6] Kardi Teknomo. “K-mean Clustering Tutorial” :http://people.revoledu.com/kardi/tutorial/kMean/index.html , 1994

[7] Serge Beucher. “Image segmentation and mathematical morphology.” May 18, 2010: http://cmm.ensmp.fr/~beucher/wtshed.html.

[8] The Math Work Inc. “Matlab - watershed transform”: http://www.mathworks.com/help/toolbox/images/ref/watershed.html , 1994.

[9] Jose Luis “Ransac C++ example”: http://www.mrpt.org/RANSAC_C%20%20_examples , 2007