Institut für
Automatisierungstechnik

Master Thesis

Adapting semantic perception for long-term manipulation using perceptual episodic memorie

Md. Rakibul Islam

November 1, 2017

Tutor:
Ferenc Balint-Benczadi

Examiner:
Prof. Dr.-Ing. Kai Michels

Prof. Michael Beetz

Copyright declaration

I hereby declare that I have finished my thesis without any external help, and that I have not used any sources other than the ones I have indicated.

All passages, taken literally or meaningfully from the publications, I have indicated as such, with reference to the sources.

The thesis will not be modified after submission.

Date: _____ Signature: _____

Acknowledgement

It is the best time in my life to show gratitude towards who are constantly with me such as the almighty God and my parents. I am grateful to my tutor Ference to guide me during the hole thesis work and his guidance helps me to finish the work successfully. Whenever I need any documents in my entire master studies life, I always go to Prof. Dr.-Ing. Kai Michels and he never disappointed me. I am greatly grateful to him to allow me to do my master thesis outside department-1. I want to take the opportunity to thank also Prof. Michael Beetz who allowed me to work in this fascinating area of research.

Abstract

Robots operating in human environment have to deal with complex manipulation tasks. For this robot needs to equipped with perceptual capabilities. One of the most important perceptual capability is how robot recognize and learn about novel objects from its surroundings.

The evaluation of this capability can be achieved by generating the perceptual episodic memory during the execution of a task. The perceptual episodic memory has great impact in robotic perception, because it can enable robot to relive past events and learn from them.

In the manipulation task robot has to perceive and classify different objects, which contains novel objects also. To classify objects, the classifier should be trained with large dataset. The dataset contains the objects or object's group, which must be present in robot working environment. In this thesis work, we used two datasets. One is RGBD dataset from university Washington and the other one is kitchen environment dataset from Institute for artificial intelligent.

Since object classification results is highly influenced by the algorithms for extracting feature from the images. We develop a tool for extracting feature and split the feature into train and test date. As feature extraction algorithm, we use CNN and VGG16 to get the object class and VFH and CVFH for object's shape.

We use popular machine learning algorithm such as random forest, support vector machine, k-nearest neighbor and gradient boost tree. We trained the classifier and evaluate its performance by using both the test and data coming from manipulation task.

Contents

1. Introduction

Semantic perception consists of meaning full informations such as object names, it's geometric informations like object's position and orientation information, understanding the relations between them from the senor data. Different algorithms are used to generate and attach the semantic perceptions to sensor data. Using this semantic perceptions robot perceive about the surroundings world in front of it. Semantic informations also help us to evaluate a robot's limitation to perceive the objects in different conditions. So it important to store them.

In the context of robotic perception, perceptual episodic memory is used to store the semantic perception. Episodic memory helps researcher to get all the necessary informations about a manipulations task that a robot has performed. So the researcher can understand if there is any mistake happened during the task execution. It helps to find the different relations between the different semantic perceptions. As a result, these relations can be used to build new learning problems, which can improve robot perception.

1.1. Motivation

For a robot performing human like perception in general is very difficult and need to cope with many algorithms. Let's consider a simple pick and place task that a robot performing in a kitchen environment. Even for this simple task, the properties of objects vary frequently and therefore the different algorithmic approaches are needed for detecting them. For example, detecting shiny cutlery from a drawer is handle very differently then detecting a cereal box from the shelf. Even this problem get more harder in different lightening condition. Moreover, During the task execution phase, false detection may occurs without noticing.To consider these problems, the perception for long-term manipulation task such ones mentioned above, needs to address on a system level.

1.2. Goal

In this thesis work, we build a perception system using the open source perception framework **RoboSherlock** [3] with active memory. The developed system will be rapped with machine learning algorithms, which classify the objects during the manipulation tasks and also consists of tool to logged task's results in the active memory and query the memory. We also focus on finding the answers of the following questions:
From how far a robot can detect an object perfectly and that is the angel between the robot's camera and the object at that moment.

We mainly focus on finding the answers of the following questions:
From how far a robot can detect an object perfectly and that is the angel between the robot's camera and the object at that moment. And how real world data can be use to classify using turn table data. In order to be able to answer this question we need to store the semantic perception as episodic memory. Everything the robot 'see' during the execution of a task is logged as episodic memories. To achieve our goal, use use Robosherlock a robotic perception framework. The frame work provides perception capabilities for mobile robots performing long-term manipulation task in kitchen environment. From this frame work we use different components called annotator or expert algorithm to generate object hypothesis and annotates the object hypothesis with different kind of semantic perception information from row sensor data (images). We also build tool for extracting features from input datasets(collections of depth and RGB images) and annotators to work with popular machine learning algorithms like support vector machine, random forest, k-nearest neighbor and gradient boost tree classifier. We evaluate the algorithms or classifiers performance using the open source datasets form Uni washington and the kitchen environment datasets from Institute for artificial intelligent.

In this thesis work we consider that the robot is working in the kitchen environment and the task of the robot is like pick and place of the object. We use Washington university dataset [6] and kitchen environment dataset from institute for artificial intelligent. Both datasets are created by taken image of different objects using a turn table with and angle difference 30 degree. We use popular supervised machine learning algorithms like support vector machine, random forest, k-nearest neighbor, gradient boost tree to create the trained model in order to classify the data which is coming from real world robot manipulation task. Before classify the real world data we cross validate our trained model using the test data and evaluate the performance of the classifiers. The real world data is recorded during the robot manipulation task and put it into

a database. The real data is row sensor date and consider as unstructured data. In oder to classify this data we need structure informations from the data. The perception framework robosherlock provide the expert algorithms or annotators which generate structured object hypothesis and annotate the hypothesis with different semantic information. As our interest to classify the real world data so we need to add class label with this objects hypothesis. So we build the classifier as the framework annotator and run together with other annotators in pipe line. This ensemble of annotators is called analysis engine. At this moment our real world data is fit for evaluation because it has enough ground truth informations. The trained model which has been created from the dataset consists of all the possible objects, which are found in a kitchen environment. We again run the analysis engine and logged all the output data into another database. The database contains all semantic perceptions which added to data during the running of the pipe line and database is now consider as the episodic memory. This memory has great impact in robotic perception. As we use the mongoDB database which provide the tools and software to query the database and generate new learning problems which improve robotic perceptions. The over of whole work is presented by a flow charts.

1.3. Related Work

In this chapter we are going to discuss the components of our work which are related to other's work.
In [**?**] authors described how to compute feature from image data using different feature extractions algorithms (especially VFH) and classify them using different machine learning algorithms such support vector machine. Also showed different techniques to ensemble classifiers, which would improved the classification accuracy.

Shape of a object is very good source to classify objects in different categories. object shape can be found from RGB-D images. VFH (View Point Feature Histogram) [11] is very prominent algorithm, which is used to calculates descriptor from depth image. It calculates histogram based on the relative angles between the surface normals of the objects to the normal viewpoint direction of the sensor. It was developed to be robust again noise, which is common in depth image sensors. Based on VFH another algorithm called CVFH(Cluster Viewpoint Feature Histogram) [1] is calculated. It works on object's cluster in the depth image. It useful for object recognition and pose estimation of rigid objects.

Memory for robot is described in literatures [8], [9] [10].Their authors described how to store data and learn form perceptual episodic memory. For working with the data they use to separate memory locations, one for perceptual memory and the second one is for semantic memory. They use non-relational database called mongoDB as central location for perceptual memory.

A general over view of this thesis topic "Adapting semantic perception using perceptual episode memory" is very much similar to approach describe in literature [2], where authors showed how episodic memory can be used to retrospect a robot manipulation task and query the memory to generate new supervised learning problem.

1.4. Overview

The structure of this paper is as follows: Related work is described in Section II. Next, we give a brief description of our system architecture in Section III. We discuss our surface normal and segmentation algorithm in Section IV followed by a discussion of the Viewpoint Feature Histogram in Section V. Experimental setup and resulting computational and recognition performance are described in Section VI. Conclusions and future work are discussed in Section VII The project report is organized ino six chapters. Chapter 2 deals with the theoretical background, chapter 3 describes the existing MATLAB model, chapter 4 covers the reprogrammed model in C++, chapter 5 includes program testing and the final chapter contains the conclusion.

2. Feature Extraction Algorithm

2.1. descriptor

When we work with RGB-D depth image, we calculate descriptors. 3D feature is called descriptors. They are more complex (and precise) signatures of a point, that encode a lot of information about the surrounding geometry. The purpose is to unequivocally identify a point across multiple point clouds, no matter the noise, resolution or transformations. Also, some of them capture additional data about the object they belong to, like the viewpoint (that lets us retrieve the pose). descriptors can be two types local descriptors and global descriptors. Local descriptors are computed for individual points that we give as input. They have no notion of what an object is, they just describe how the local geometry is around that point. Global descriptors encode object geometry. They are not computed for individual points, but for a whole cluster that represents an object. Because of this, a preprocessing step (segmentation) is required, in order to retrieve possible candidates. Global descriptors are used for object recognition and classification, geometric analysis (object type, shape...), and pose estimation.

2.1.1. Viewpoint Feature Histogram

The VFH (Viewpoint Feature Histogram) is derived from the FPFH (First Point Feature Histogram). The VFH consists of two parts: a viewpoint direction component and an extended FPFH component. To calculate the first one, the object's centroid is computed, which is the point that results from averaging the X, Y and Z coordinates of all points. After that, the vector between the viewpoint (the position of the sensor) and the centroid is computed, and normalized. Finally, for all points in the cluster, the angle between their normal and this vector is calculated, and the result is binned into an histogram. The vector is translated to each point when calculating the angle cause it makes the descriptor scale invariant. The viewpoint direction component calculation is shown graphically in figure below.

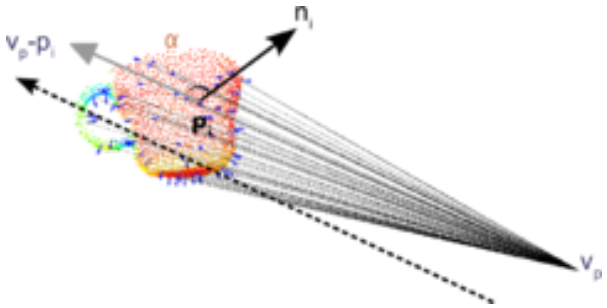

Figure 2.1.: a viewpoint direction component

The second component is calculated like as the FPFH that generates 3 histograms for the 3 angular features α, ϕ and θ as shown in fig below.

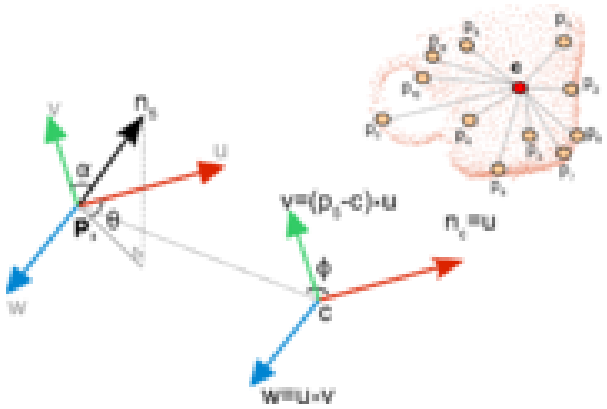

Figure 2.2.: a viewpoint direction component

It is measured between between the viewpoint direction at the centroid and each of the normals on the surface. The bottom left part of the above Figure presents the selection of the Darboux frame and a graphical representation of the three angular features. In detail, for a pair of 3D points (p_i, p_j) and their estimated surface normals (n_i, n_j) the set of normal angular deviations can be calculated as:

$$\begin{aligned} \alpha &= v \cdot n_j \\ \phi &= u \cdot \frac{(p_j - p_i)}{d} \\ \theta &= arctan(w \cdot n_j, u \cdot n_j) \end{aligned} \tag{2.1}$$

where u, v, w represent a Darboux frame coordinate system chosen at p_i. Then,

the Point Feature Histogram at a patch of points $P = p_i$ with $i = 1, ..., n$ captures all the sets of (α, ϕ, θ) between all pairs of (p_i , p_j) from P, and bins the results in a histogram. The resulting 4 histograms as 1 for the viewpoint component and other 3 for the extended FPFH component are concatenated to create the final VFH descriptor as shown in figure below.

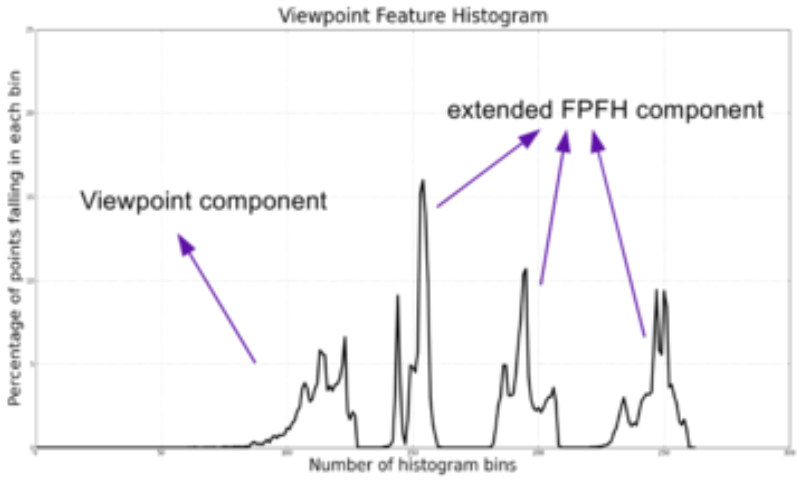

Figure 2.3.: VFH histogram

2.2. Theoretical Background

In this thesis work we are working with image data and find out the best classification algorithm, which classifies the data in the most precisely. As to work with the whole image is very costly, so we use algorithm to find the pixel points or features in the image, which can represent the corresponding image perfectly. Deep learning algorithms are very efficient and precise to describe the image features. Therefore, in our work we use convolution nural network (CNN) and visual geometric group (VGG16) as our features. For point cloud or 3D image, we use view point feature histogram and cluster viewpoint feature histogram algorithms. For classification there are a lots of algorithms available in Machine learning. Reviewing the literature, we are come up with the support vector machine (svm), random forest(RF), gradient boosting (GBT) and K-nearest neighbor algorithms. The descriptions about the algorithm and how these work are given bellow;

2.3. shape based object recognition

2.4. Feature

When we process images, we can not work with hole image due to high computational cost. Instead of this, we find out some points by applying some characteristics using different algorithm in the image. These characteristics of the point is called feature, which can help us to differentiate one point from another point in image. We can tell a feature is optimal, if it satisfied the following criteria:

1.It must be robust to transformations: rigid transformations (the ones that do not change the distance between points) like translations and rotations must not affect the feature. Even if we play with the cloud a bit beforehand, there should be no difference.

It must be robust to noise: measurement errors that cause noise should not change the feature estimation much.

It must be resolution invariant: if sampled with different density (like after performing downsampling), the result must be identical or similar.

2.4.1. Convolution Neural Network

2.4.2. Visual geometric group

3. Learning Algorithm

In general, any machine learning algorithm can be divided into one of the two major types called supervised and unsupervised learning.

Supervised learning In this learning, we are given a data set and we already know what our exact output should look like, considering the idea that there is a relationship between the input and the output. Supervised learning problems are again categorized into "regression" and "classification" problems. In a regression problem, we predict results within a continuous output, so we map input variables to some continuous function. In a classification problem, we predict results in a discrete output. In other words, we try to map input variables into discrete categories. Examples of Supervised Learning are Regression, Decision Tree, Random Forest, K-nearest neighbor, Logistic Regression etc.

Unsupervised learning This learning allows us to deal problems with very little or no idea what our results should look like. We try to derive structure from data where we don't necessarily know the effect of the variables. We can derive this structure by clustering the data based on the relationships among the variables in the data. In this algorithm, we do not have any target or outcome variable to predict or estimate. So there is no feedback based on the prediction results. Examples of unsupervised learning are K-means, Apriori algorithm etc.

In thesis thesis work we solely focus on supervised learning more precisely classification problem. There are a lot of supervised learning algorithms available to choose from. We pick most popular ones such as Support Vector Machine, Random forest, K-nearest Neighbor, Gradient boost tree for our data classification tasks. The details about how this algorithms work are described in the followings.

3.1. Support Vector machine

Support vector machine is a complex and advanced machine learning algorithm. It works on labeled train samples (supervised learning) and outputs an optimal

hyperplane which categorize novel samples. The algorithm tries to find the hyperplane that gives the largest minimum distance to the training samples. For better intuition of how the algorithm decide the optimal hyperplane, lets consider the following scenarios. Consider the image below which has two types of data, red and blue. Form the image it is clear that, there are many lines possible lines to separate the blue and read data.

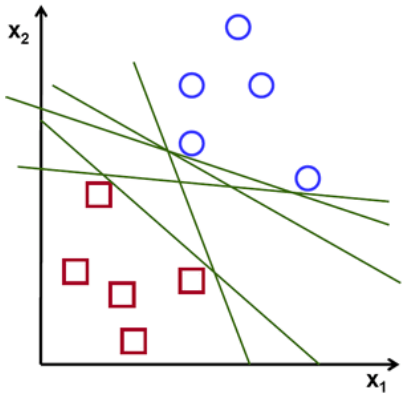

Figure 3.1.: support vector machine

Therefore, we need a way to find out the best line to separate the data. The algorithm finds out some special data points from both types (in below figure filled blue and read rectangle points) which are very close to each others. These data points are called support vectors and the lines pass through them are called support planes (doted green lines).

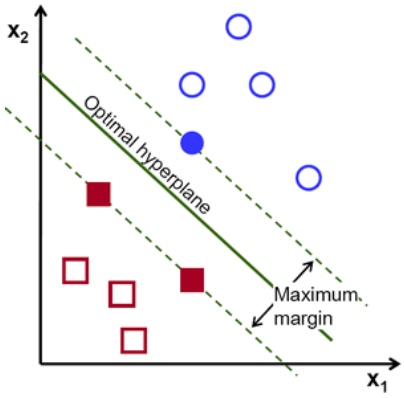

Figure 3.2.: support vector machine

Then the algorithm finds the optimal hyperplane (bold green line), which passes thorough the maximum margin distance from both of the support planes. To get the maximum margin is an optimization problem. So far we consider the data which are linearly separable.

Lets consider the following figure, where data can not be linearly separable. So svm uses a technique called kernel trick. The kernel trick is a special function, which maps the train samples from lower dimensional feature space to higher dimensional feature space.

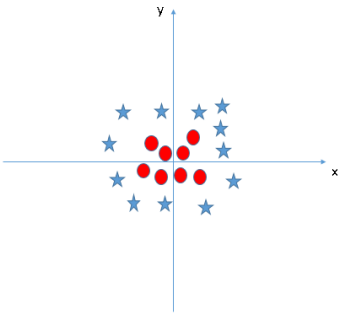

Figure 3.3.: support vector machine

To understand the concept let's map our train data (above figure) to a new feature space $z = x^2 + y^2$. Therefore, all values for z would be always positive because z is the squared sum of both x and y as shown by the figure below. So, it is seen from the figure the data can be easily. Therefore, the chance is more for a nonlinear separable data to become linear separable in higher dimensional space.

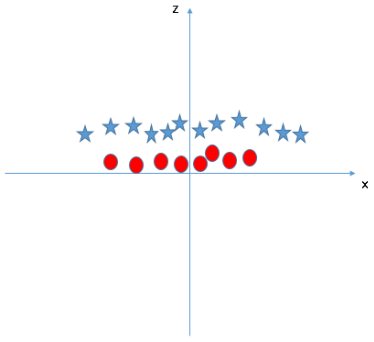

Figure 3.4.: support vector machine

In our original input space the hyperplane should looks like a circle as shown by the following figure.

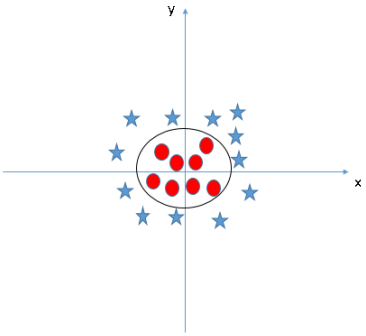

Figure 3.5.: support vector machine

Considering all the concepts above, there still exists the problem of misclassification data. So it is not sufficient just finding decision boundary with maximum margin. Therefore we have to consider the problem of misclassification errors also. So model should be be modified such that it finds the decision boundary with maximum margin but with less misclassification. The minimization criteria is modified like as:

$$\min ||\omega||^2 + C(distance\ of\ misclassified\ samples\ to\ their\ correct\ regions)$$

This concept is described by the figure below.

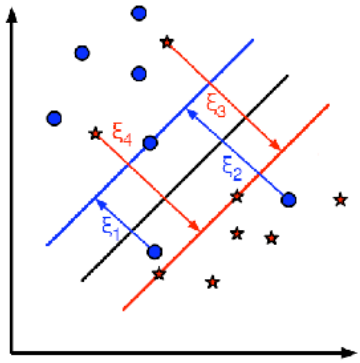

Figure 3.6.: support vector machine

For each sample of the training data a new parameter ξ_i is declared. ξ_i is the distance from its corresponding training sample to their correct decision region (shown by the read and blue arrows). The samples which are not misclassified and therefore, they fall on their corresponding support planes, their distance is considered as zero. So the updated optimization problem is:

$$\min_{\omega,b_0} L(\omega, b_0) = ||\omega||^2 + C\sum_i \xi_i \text{ subject to } y_i(\omega^T x_i + b_0) \geq 1 - \xi_i \text{ and } \xi_i \geq 0 \,\forall i$$

How should the parameter C be chosen, which depends on how the training data is distributed. Although there is no general answer but it is useful to consider the following rules:

- Large values of C provide solutions with less misclassification errors but a smaller margin. Consider the fact that it is expensive to make misclassification errors. Since the goal of the optimization is to minimize the argument, so few misclassifications errors are allowed.
- Small values of C provide solutions with bigger margin and more classification errors. In this case the minimization does not consider the summation term of ξ_i, instead focuses more on finding a hyperplane with big margin.

So the choice of the value of parameter C is a trade-off between the misclassifiaction and margin size.

3.2. Random Forest

Random forest is a supervised learning algorithm and also known as ensemble method. The goal of ensemble method is to build a strong classifier from the the ensemble of several weak classifiers. Decision tree algorithm is used as the weak classifiers. The whole algorithm approaches can be divided as model classifying phase.

Model Training: At first, we create different training datasets, where each dataset holds the same number of samples as the original dataset using Bootstrap method. Bootstrap is the process of drawing sample data as random with replacement. Therefore, some samples will draw more than once and some will be absent. We use these new training datasets for creating the trees. For N numbers of new training datasets, we get N numbers of trees in the forest. At each node of each trained tree, a random subset of the features instead of all features are used to find the best split. With each node a

new subset of features is generated and the size of the subset is fixed for all the nodes and all the trees. Generally, If we have F number of features in a sample, then the subset contains $\sqrt{F}$ features. In the forest, none of the built trees are pruned.

Classifying Phase: The classifier takes the input sample, classifies it with every tree in the forest and outputs the class label that received the majority of votes.

The classification phase of the algorithm is shown by the following figure. Where internal or decision node is shown by circle and leaf or terminal node is shown by rectangle.

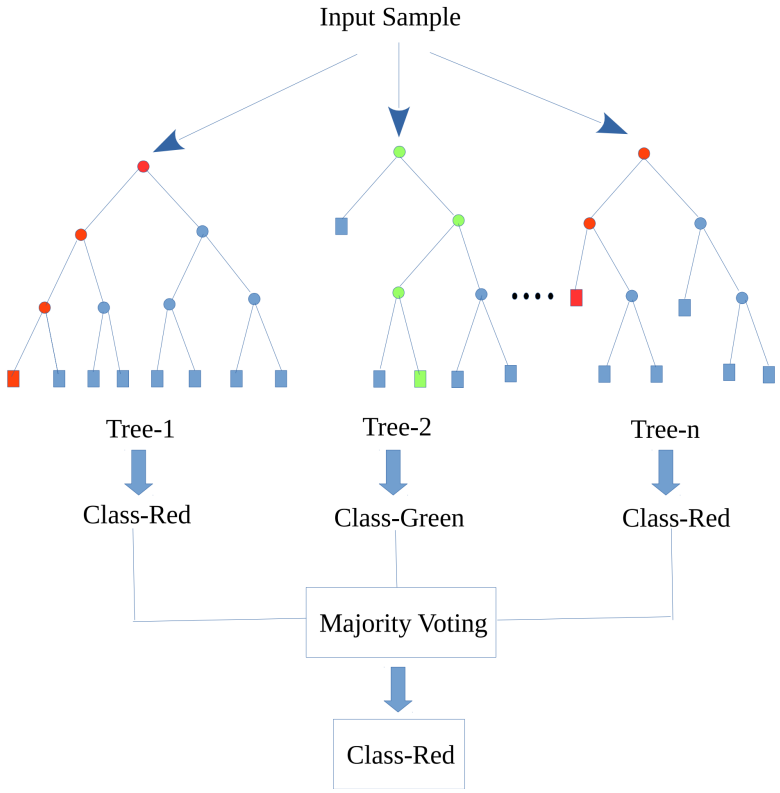

Figure 3.7.: Random forest Classifier

3.3. Gradient Boost Tree

Gradient boosting decision tree(GBDT) is a boosting ensemble algorithm which builds predictive model by the ensemble of weak models, usually decision trees. GBDT works in following steps:

1. Optimization of a loss function.
2. Predictions by a weak model.
3. Adding weak model to minimize loss function.

The pseudo code of boosting algorithm is given below: The input training set $(x_i, y_i)_{i=1}^{n}$ and for M number of iterations, the loss function is $L(y, f(x))$
1.Initialize weak model which can be written as:

$$F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma) \tag{3.1}$$

2. For $M = 1$ to m, compute the gradient as:

$$z_{im} = -[\frac{\delta L(y_i, f(x_i))}{\delta f(x_i)}]_{f=f_{m-1}} \tag{3.2}$$

where $i = 1, ..., n$
3. Fit base tree $g_m(x)$ to predict the targets Z_{im} for all training data.
4. Compute gradient descent step size as:

$$\gamma_m = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, f_{m-1}(x_i) + \gamma g_m(x_i)) \tag{3.3}$$

5. Update the model as :

$$f_m(x) = f_{m-1}(x) + \gamma g_m(x)) \tag{3.4}$$

6. Return the final model $f_M(x)$ ensemble.GradientBoostingClassifier and ensemble.GradientBoostingRegressor pack- ages are implemented GBDT for classification and regression algorithm, respectively.

3.4. K-Nearest Neighbor

K-nearest neighbor (KNN) is very simple but powerful supervised learning algorithm, which is used both for classification and regression problems. The

algorithm works in three steps.

- Firstly, for given a positive integer K and a test sample x_0, the KNN classifier recognizes the neighbors K points in the training data that are closest to x_0, denoted by N_0.
- Secondly, It calculates the conditional probability for class j as the fraction of points in N_0 whose response values are equal to j. The conditional probability is given by the following equation [5]:

$$P_r(Y = j|X = x_0) = \frac{1}{K} \sum_{i \in N_0} I(y_i = j) \tag{3.5}$$

  Where, $I(x_0)$ is the indicator function which evaluates to 1 when the argument x_0 is true and 0 otherwise.
- Finally, KNN classifies the test sample x_0 to the class with the largest probability.

Now we illustrate how KNN algorithm works with the following figure. We have plotted a small training data set consisting of five red stars and eight green circles observations. Let's say stars and circles are in red and blue classes respectively. Our aim is to make a prediction for the test data labeled by the blue rectangle.

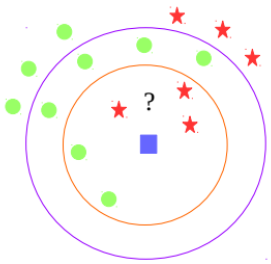

Figure 3.8.: Simple figure to explain KNN algorithm

In the first case, suppose that we choose $K = 5$. Then KNN will first recognize the five observations that are closest to the blue rectangle. This neighborhood is shown as a circle of orange color. It consists of two green points and three red points. So the resulting estimated probabilities for the red class is 3/5 and for the green class is 2/5. Hence KNN will predict that the blue rectangle belongs to the red class.

In the second case, assume that we choose $K = 9$. Then algorithm first identifies the nine observations that are closest to the blue rectangle. This time neighborhood is displayed as a circle of purple color. It has six green points and three 3 red points. Therefore, the resulting estimated probabilities for the red class is 3/5 and for the green class is 6/5. Hence KNN will predict that the blue rectangle belongs to the green class. To avoid the tie condition, the value of K should be chosen as odd integer number.

4. Dataset and Evaluation Method

4.1. Result Evaluation Terms

4.1.1. Classification evaluation method

Confusion matrix: A confusion matrix has information about actual and predicted class labels done by a model. The data in the matrix are commonly used to evaluate performance of such model. The following table illustrates the confusion matrix for binomial classification model.

Table 4.1.: Confusion Matrix

		Predicted	
		Positive	*Negative*
Actual	*Positive*	TP	FN
	Negative	FP	TN

The data in the confusion matrix has the following meaning:
TP is the number of true predictions that an instance is positive,
TN is the number of true predictions that an instance is negative,
FP is the number of false predictions that an instance is positive,and
FN is the number of false of predictions that an instance negative.

Accuracy Score: Accuracy score is the most common evaluation metric for classification models. It is the number of correct predictions as a ratio of the total number of predictions. The best accuracy is 1 and the worst is 0.0. It can be calculated as:

$$Accuracy = \frac{\sum TP + \sum TN}{\sum TP + \sum TN + \sum FN + \sum FP} \tag{4.1}$$

Recall: Recall is the number of correct predictions as a ratio of the total number of positives. It is also call the true positive rate. It can be written as:

$$Accuracy = \frac{\sum TP}{\sum TP + \sum FN} \tag{4.2}$$

Precision: Precision is the number of correct positive predictions as a ratio of the total number of positive predictions. It can be written as:

$$Accuracy = \frac{\sum TP}{\sum TP + \sum FP} \tag{4.3}$$

ROC Curve: ROC curve illustrates relative tradeoffs between true positives and false positives. It is atwo dimensional graph in which recall is plotted on the Y axis and specificity is plotted on the X axis.Specificity is the number of correct negative predictions as a ratio of the total number of negative predictions.A binomial model is one that outputs only a label.Figure shows binomial models. Some points are important in ROC space. The lower left point (0,0) depicts the strategy of never issuing a positive classification; such a model does have no false positive errors but also no true positives gain. The point (0, 1) means this is a perfect classification such as performance of D's.In contrast, the performance of E's is worst.

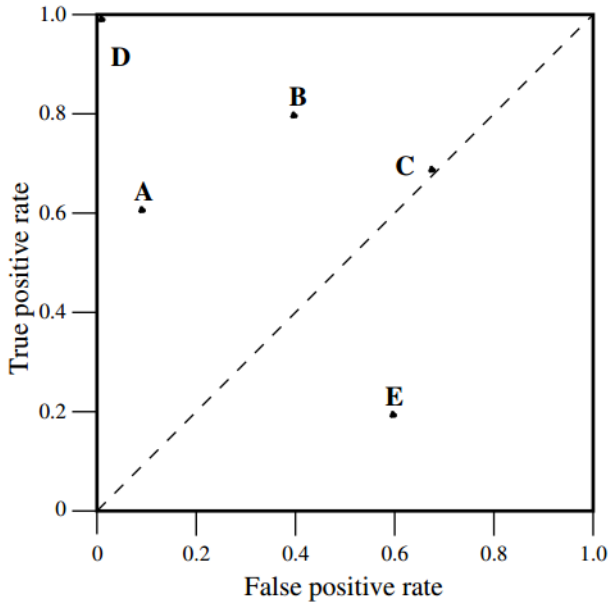

Figure 4.1.: A basic ROC graph showing five discrete classifiers.

4.1.2. Cross Validation

Cross validation is a method for evaluating predictive models. It is better than residuals. There is no indication about how well the model will predict for new unseen data in residuals.This problem can be overcome by not using entire data set when training a model. A portion of data is removed before training the model. This removed data will be used for testing the model performance on new data. This idea is cross validation. holdout method and k-fold cross validation methods are used to evaluate the model performance of this study. Holdout Method: It is the simplest type of cross validation. The data set is splitted into two sets as training sets and testing set. The model is trained with the training set and then testing data used for prediction. As the splitting training and testing sets are independent, the evaluation result may be different. K-fold cross validation: In K-fold cross validation, the data set is split into k subsets. Each time k-1 subsets are put together and sued to train the model, and rest one subset used to evaluate the model which was not used to train. This process is repeated for k times where each time different subset used for evaluation. Then the average error across all repetitions is computed. It can detect the overfitting. Each data point is tested. The computational

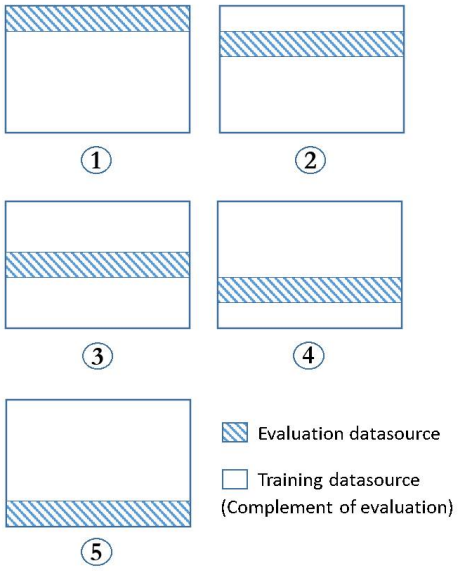

Figure 4.2.: Splitting dataset in 5-fold cross validation. Reproduced from

time is high compared to holdout method. It is possible to choose independently how large the each test set. The Figure 6.2 shows an example of k-fold cross validation where k = 5. First 20 percent of data used for evaluation and rest 80 percent for training in model one. Second 20 percent (21-40 percent) used for evaluation and rest subsets for training and so on. To evaluate predictive models of this study, k=10 is used for cross validation. 10- fold cross validation is used only for regression models where the dataset has 147473 observations.

5. System Overview

From the recent past till now a lots of researches has been going on in the field of robotics, specially how to equipped a robot with perceptual capabilities. Therefore, a new brunch of robotics has been immersed is known as cognitive robotics or robotic perception, where people research how to build an intelligent robot or machine. To get human like perception from a robot is still a dream. There is huge gap between the functionalities require for a robot performing human-scale manipulation tasks and the functionalities today's perception algorithms provide with. However there are some robots in the lab environment, which already equipped with some perceptual capabilities. It is easy task for a human to recognize a object from a table. But for the same task, a robot needs a lots of complicated approaches and perception algorithms.

The field of image processing and robotics are complementary with each other. In the history of image processing, there were huge numbers of researches have been carried out and a lots of algorithms has been developed. People preserved these research works as programming library and framework, so that the research work can be used later. There are quite significant numbers of libraries exist, which provide the perception algorithms.

In the aim to facilitate and organize the research work in the field of robotic perception, the institute for artificial intelligence developed a perception framework called **roboSherlock** [3]. The framework has been designed to work with perception algorithms that exist in different libraries and provides a common place to integrate these algorithms flexibly. Since, our developed perception system for this thesis work is based on **Robosherlock**, so at first we will give an overview of the components and functionalities of **roboSherlock** in the following section. After that we will illustrate about our implemented system and it's usages.

5.1. RoboSherlock

Robosherlock is based on the open source framework UIMA (Unstructure information management architechture). UIMA is a software framework for the

analysis of unstructured data such as text documents, video or audio data. It is a pluggable architecture allowing integration and modularization, and has been proven to be a powerful tool for unstructured information processing.Perhaps the most famous example of a UIM system is Watson [4], a question answering system that competed against the champions of the quiz show called jeopardy and and won the quiz show. In UIM, pieces of unstructured informations are processed by a collection of specialized information extraction algorithms (called annotators) and each algorithm contributes pieces of knowledge with respect to their expertise. Therefore, the output knowledge collected from different annotators are combined to get the consistent final decision. The infrastructures and components of the UIMA is shown by the following figure.

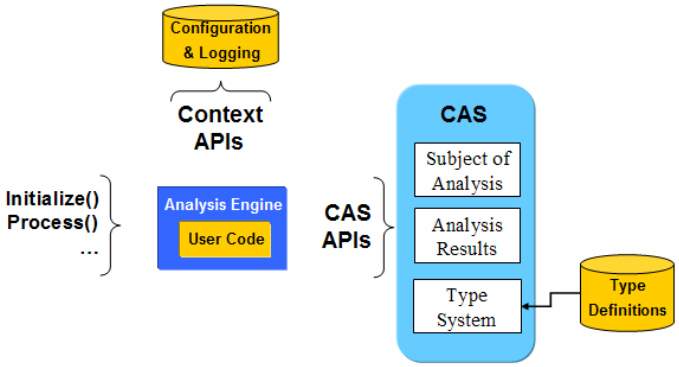

Figure 5.1.: UIMA Framework Core

RoboSherlock inherits the basic data structure and components of control mechanism from UIMA. As mentioned in [2] a user defined type system serves as a blue-print for the internal data structure and permits communication between the components of the framework through the common analysis structure (CAS). The CAS acts as a white-boards architecture, allowing components to post and retrieve data solely based on the type system. At any given times, components only need to know only the data types they are supposed to post or send to CAS. The main components of the framework is Analysis engine. there are two types of analysis engines depending on functionality. The first one is primitive analysis engine (Annotator), which is used to provide the functionality of a single method or algorithm. The second one is aggregate analysis engine (ensembles of annotators), which provides the complete functionalities of a perception system. It is also known as the pipeline. The annotators are mainly perception algorithms from libraries OpenCV (open source computer vision) and PCL (Point Cloud Library). The complete work flow of a pipe line

are describe using the following figure.

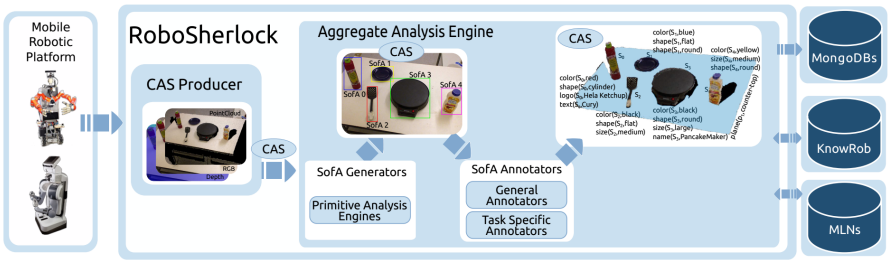

Figure 5.2.: Example of pipeline execution in Robosherlock

The sensor data used by the pipeline is generated by a Kinect camera mounted on the head of the robot. The data is typically standard RGB color image and point cloud (depth-RGB image), which is a set of 3D point in space represents the external surface of an object. Robosherlock considers row sensor data (image scene) as unstructured information in which it tries to find pieces of data that represent object or object groups. These pieces of data is called hypothesis or Sofa (subject of analysis), which are produced by annotators, consist of image segmentation and clustering algorithms. After that the hypotheses are annotated with symbolic (e.g. class label) or sub-symbolic (e.g. color histogram) information. Depending on the task at hand, the annotations are used to test and rank the possible answer. In the pipeline the output of one annotator can be the input of another annotator. If someone wants to store the data precessed by pipeline for further analysis, there is a storage writer annotator, which writes data in database.The database is consider as the memory of robosherlock. The storage writer annotator should placed at the end of the pipeline. When the pipeline processing is finished, the CAS gets reset and new processing loop starts again.

5.1.1. Memory For Robosherlock

In cognitive psychology, memories are categorized into short-term and long-term memory, where the short-term is a small capacity storage that provides the context for executing the current task. The long-term memory is a high volume memory that gives comprehensive information for all sorts of tasks and [12] can be further categorized as procedural or declarative memories. Declarative memory is often considered as consciously accessible and the procedural memory consists of compiled or subconscious information. Episodic memories store experienced event information that is temporally and spatially organized

and combined with context information. Episodic memories has great importance in robotic perception [], because they enable robots to "relive" past events and learn from them.

The framework provides MongoDB database based state of the art memory system for robotic agents performing manipulation tasks in kitchen environment [2]. The reason of choosing MongoDB as the storage location because it is a schema-less database means it doesn't have any fixed data structure. This property nicely matches the paradigms of unstructured information processing, as nobody can not ensure that ever hypothesis will be annotated in the same way or that every execution loop produces the same views. MongoDB organizes data in collection and documents. The top level structure of the collections of the database and their relations are shown by the following figure.

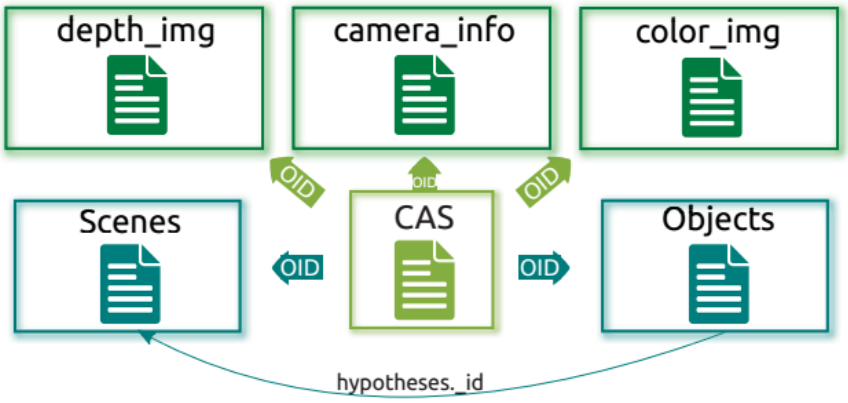

Figure 5.3.: Top level structure of MongoDB and relation bewteen collections taken from [**?**].

The collections are connected with each other by unique Id (OID). They are indexed on the timestamps of the incoming data and previously mentioned unique ids are automatically generated by the MongoDB client. Inside collections documents are arranged as key pair value structure. As the names of the collections in the top row infer that they contain information about the point cloud (Depth image), camera and color image receptively.

The CAS collections contains the OID (ObjectId) of the documents all other collections and acts as a parents. One can programmatically manage the "ObjectId" of a documents from CAS and can access documents correspond with

this ”ObjectId” from all other collections. The CAS collection for a single document is shown as follows.


```
{
  "_id" : ObjectId ( "59141fc10c8aad1ad83ca302" ) ,
  "_timestamp" : NumberLong ( 1494491073116242246 ) ,
  "scene" : ObjectId ( "59141fc10c8aad1ad83ca301" ) ,
  "color_image_hd" : ObjectId ( "59141fc10c8aad1ad83ca304" ) ,
  "camera_info_hd" : ObjectId ( "59141fc10c8aad1ad83ca305" ) ,
  "depth_image" : ObjectId ( "59141fc10c8aad1ad83ca308" ) ,
  "camera_info" : ObjectId ( "59141fc10c8aad1ad83ca309" ) ,
  "color_image" : ObjectId ( "59141fc10c8aad1ad83ca30c" ) ,
  "depth_image_hd" : ObjectId ( "59141fc10c8aad1ad83ca30d" )
}
```


Figure 5.4.: The cas collection

Every perception system that runs during long-term task, needs to build a belief state about the world. The ”objects” collection contains the objects that are believe to be exist in the environment after entity resolution and are connected to the individual instances through the IDs of the object hypotheses that are located in scenes collections as mentioned in [2].

```
{
  "_id" : ObjectId ( "59141fc10c8aad1ad83ca301" ) ,
  "_parent" : ObjectId ( "59141fc10c8aad1ad83ca302" ) ,
  "_type" : "rs.core.Scene" ,
  "timestamp" : NumberLong ( 1494491073116242246 ) ,
  "viewPoint" : {
    "_id" : ObjectId ( "59141fc10c8aad1ad83ca303" ) ,
    "_parent" : ObjectId ( "59141fc10c8aad1ad83ca301" ) ,
    "_type" : "rs.tf.StampedTransform" ,
    "rotation" : [ . . . ] ,
    "translation" : [ . . . ] ,
    "frame" : "map" ,
    "timestamp" : NumberLong ( 1494491073017362373 ) ,
    "childFrame" : "head_mount_kinect_rgb_optical_frame"
  } ,
  "identifiables" : [ ] ,
  "annotations" : [ ] ,
  "thermalViewPoint" : {}
}
```


Figure 5.5.: The scene collection

```
"identifiables" : [
{
   "_id" : ObjectId ( "59d0f4143bdc9f927efa5bb4" ) ,
   "_parent" : ObjectId ( "5914206c0c8aad1ad83ca99b" ) ,
   "_type" : "rs.scene.Cluster" ,
   "annotations" : [
   {
      "_id" : ObjectId ( "59d0f4183bdc9f927efa5bdc" ) ,
      "_parent" : ObjectId ( "59d0f4143bdc9f927efa5bb4" ) ,
      "_type" : "rs.annotation.Classification" ,
      "source" : " " ,
      "classification_type" : " " ,
      "classname" : "blue_spotted_plate" ,
      "featurename" : "CNN" ,
      "classifier" : "Random Forest" ,
      "parameters" : " " ,
      "model" : "IAI" ,
      "confidences" : {}
   } ,
   { . . . } ,
   ] ,
}]
```

Figure 5.6.: Structure of identifiables in a scene document

```
"identifiables" : [
{
   "_id" : ObjectId ( "59d0f4143bdc9f927efa5bb4" ) ,
   "_parent" : ObjectId ( "5914206c0c8aad1ad83ca99b" ) ,
   "_type" : "rs.scene.Cluster" ,
   "annotations" : [
   {
      "_id" : ObjectId ( "59d0f4183bdc9f927efa5bdc" ) ,
      "_parent" : ObjectId ( "59d0f4143bdc9f927efa5bb4" ) ,
      "_type" : "rs.annotation.Classification" ,
      "source" : " " ,
      "classification_type" : " " ,
      "classname" : "blue_spotted_plate" ,
      "featurename" : "CNN" ,
      "classifier" : "Random Forest" ,
      "parameters" : " " ,
      "model" : "IAI" ,
      "confidences" : {}
   } ,
   { . . . } ,
   ] ,
}]
```


Figure 5.7.: Structure of identifiables in a scene document

5.1.2. Web based Visualization tools

As the database contains the meta information, so it is not understandable which annotations are wrong. The frame work also provides a javascript based visualization tools [**?**]. It helps researcher to understand when the robot makes mistakes during the manipulation tasks. So he takes right measures from the errors.

Figure 5.8.: Structure of identifiables in a scene document

5.2. Developed System's Architecture

To implement this thesis work, we had to exploit quite significant number of tools or software code. This tools are mostly wrapped with different perception and machine learning algorithms from different perception frameworks or libraries. We consider **Robosherlock** is our main target framework to implement our work. Some of the tools are already existed in the framework and we also developed some tools on top **Robosherlock** using other libraries. As the **Robosherlock** is build by **C++** programming language, so we use the same language for developing tools, which are utilized to work together with all ready existed one. We use **python** programing language to work with the episodic memory and to evaluate different experiments, which we have done through the entire thesis work. We use web based python library called **flask** and the markup language **HTML** to add the ground truth annotation tool on top of already existed web based visualization tool and also add some extra functionalities to visualize the ground truth information of the object hypotheses. The usages of the implemented tools are given in index. The overview of the developed tools and their functionalities are described by the consecutive flow diagrams are given below. These flow diagrams also resemble the works flows of this thesis.

5.2.1. Work Flows from extracting feature to predictive model evaluation:

As we already mentioned, in this thesis work we focus on supervised learning algorithms for the classification tasks. For the supervised learning problem we need data with ground truth, which is actually the object name or class label. The **FeaturExtractor** tool takes input as the dataset, extracts features from every image and set a label for the corresponding object in the image. After that, it splits the dataset as training and test data, where every forth image in the dataset considers as test data and rest of the data is used to train the classifier. Therefore, we get approximately 75% as training data and 25% as the test date, which is used for the cross-validation of the predictive model. The **FeaturExtractor** tool is wrapped with feature extraction algorithms from **PCL** and deep learning framework called **Caffe** through **openCV** library. The algorithms are VFH, CVFH and CNN which is trained on two different networks called BVLC_CFC7 and VGG16. The algorithms have been already illustrated in chapter 2. The usages of the tool are given in index A.1.

In the first step, we train the specific classifier using **ModelTrainer** tool.

The tool takes training data as input and generate the predictive model of the specific classifier. The tool is wrapped with popular machine learning algorithms like random forest, support vector machine, K-nearest neighbor and gradient boost trees from **openCV** machine learning module. The usages of the tool is given in index.A.2.

After that, we evaluate the performance of the trained or predictive model using **ModelEvaluator** tool. The tool takes inputs as the predictive model and the test data generated during the feature extraction step. As output it generates the confusion matrix, which describe the classification results of the test data and the overall accuracy of the classifier. The usages of the tool are illustrated in index.A.3. The whole processes are shown by the following flow diagram.

Flow Diagram

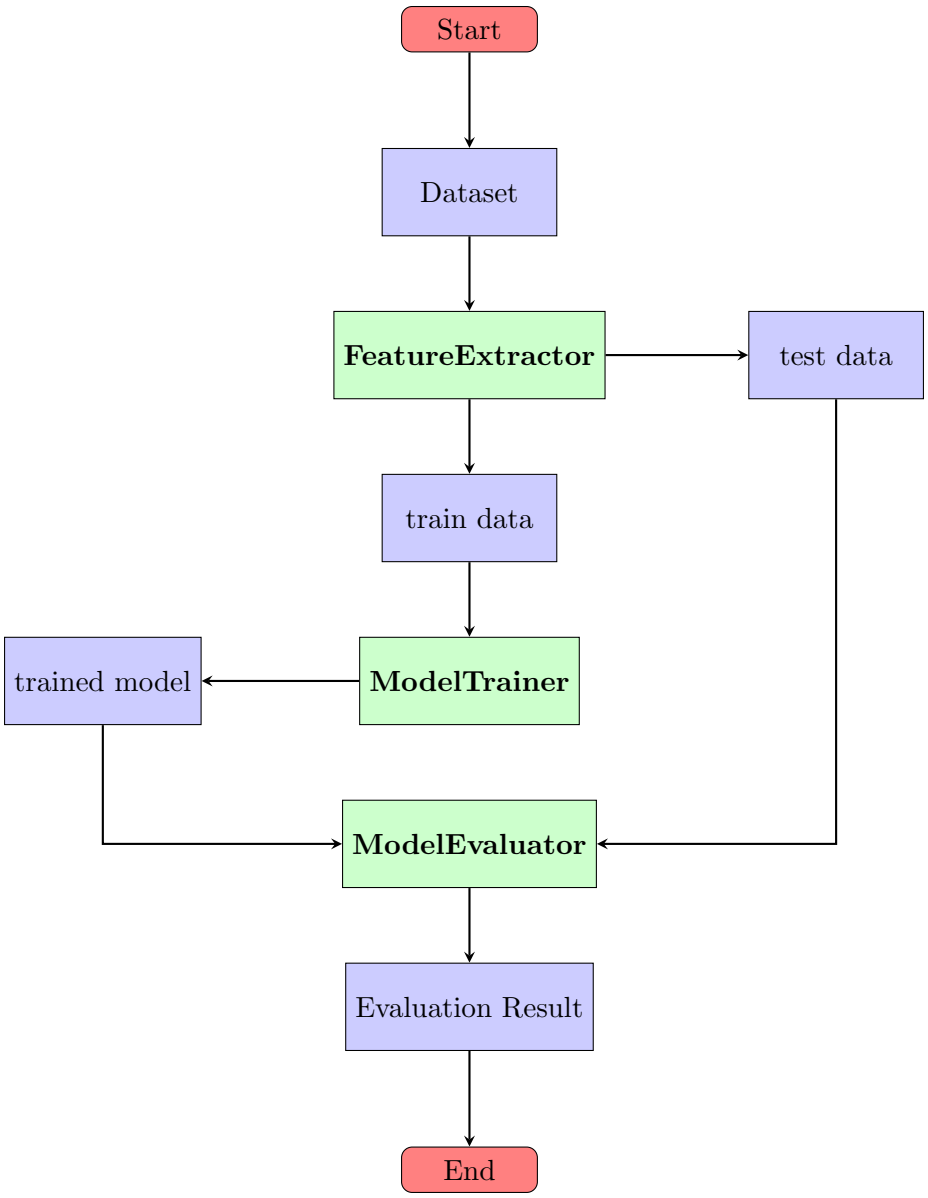

Figure 5.9.: Overview of the process from input to a predictive model evaluation

5.2.2. Classifier Annotator

Since all the perception algorithms in **Robosherlock** are build as annotators, we also build each classifier in the same way in order to work with **Robosherlock** pipe line. These classifier annotators are placed in the pipeline in a specific order and classify the object hypotheses, which are generated by the other annotators in the pipeline. The classifier annotator takes inputs as specific predictive model and annotate each hypotheses with a class label more precisely the object name or shape of the object in case of shape classification. The usages of the classifier annotator are given in index A.3 and the flow diagram is show below.

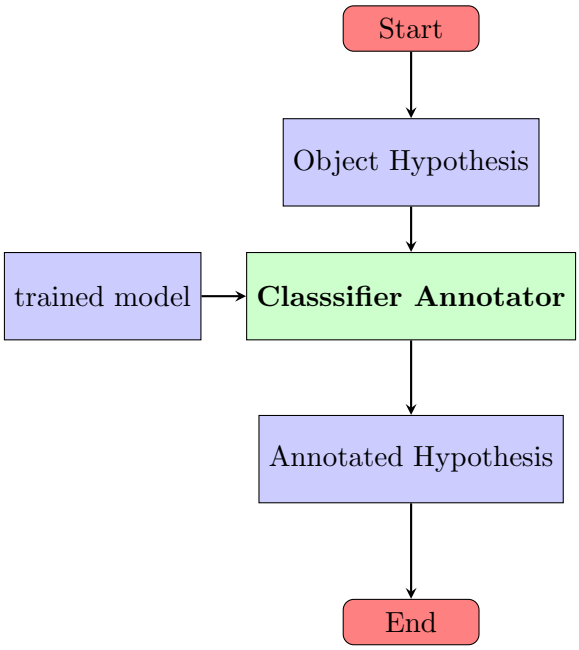

Figure 5.10.: Overview of object hypotheses classification

5.2.3. Flow chart to describe robosherlock pipeline and query to the database

When data is coming from a robot manipulation task or from a database, these data are initially considered as test data and there are no label with the data. There for it is not possible to evaluate the classification result. One way to add the object label with the data is to manually add them by human interaction but the process is quite time consuming and monotonous as described in cite ditehobe. Because a quite significant number of data are coming from a robot manipulation task. We solve this problem by trained a specific classifier on a small turn table dataset. As we know which objects are contains in the dataset coming from the pipe line. So we trained the classifier on same number of objects on turn table data and generated a predictive model. Then we classify the image coming from the pipeline using this small model which add the class label to the data. The reason for using small predictive model is that the less variation of the data in the predictive model provides more classification accuracy. Furthermore, there might be wrong label added to the data because there is no guarantee that the classifier add the correct label to the data. Using the web visualization tool we can see which hypotheses has which class label, so false label annotation easily traceable. We developed a small tool called **GorundTruth Annotation** is used to correct the false label annotation. The figure of the annotation tool is shown below. Where the sceneId is the id of the scene in which the hypothesis is belong and object number is the number of hypothesis in the scene. The **CL pipeline** is used to classify the object hypotheses on the large predictive model which contains all the possible objects of the environment where robot agent supposed to work. In our case we consider 54 objects which names are mentioned in section Data Acquisition. Finally we get the database where the hypothesis contains and annotated with all the necessary information to do our next experiment. As the pipeline contains many other annotators which also annotate the hypotheses according their expertise. This database is considered as the episodic memory of the robot. To query this memory we do different experiments and evaluate the performance of process. The details about the different experiments are illustrated in the next chapter. The overview are shown by the following flow diagram.

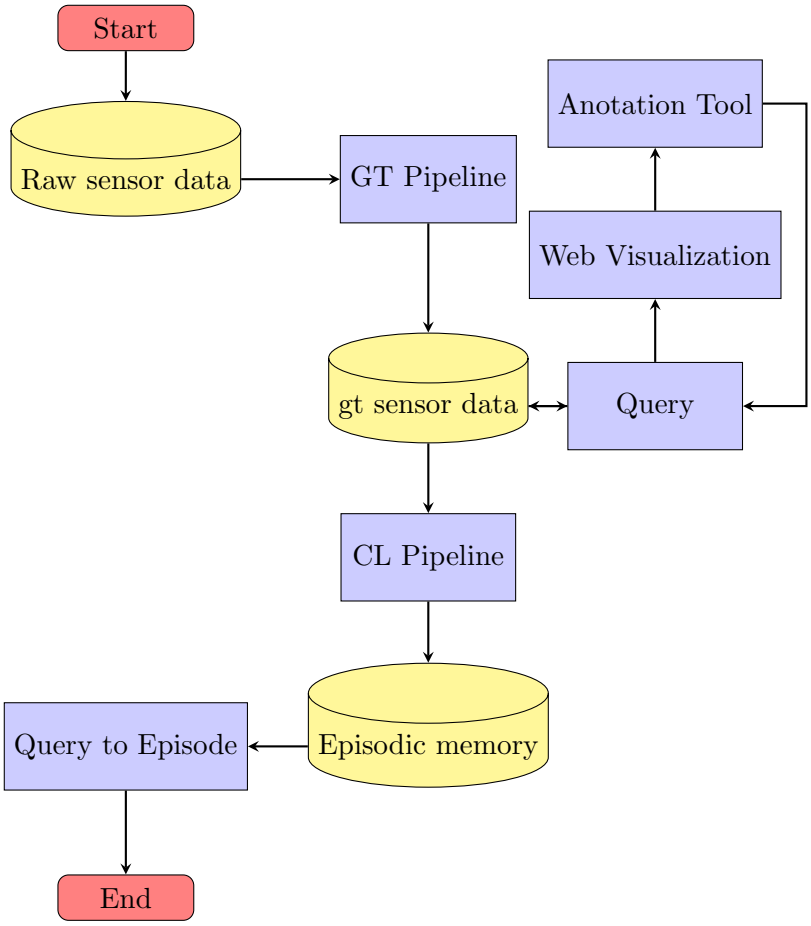

Figure 5.11.: Architecture of the developed perception system

5.2.4. Libraries for Algorithms Implementations

5.2.5. UIMA

It stands for unstructured information management architecture. The library provides tools and functions to analyze large volumes of unstructured information in order to find structure data from them. For an example, UIM application absorb information from a plain text and identifies entities such as persons, places, organizations or relations that a end user wants. The infrastructure and components of the application system is shown by the following figure and their functionalities will be illustrate during the discussion of robosherlock components. The framework provides perception capabilities for a

robotic agent performing everyday human manipulation tasks in the kitchen environment. It provides Robosherlock is mainly wrapped with other three libraries of frame work name UIMA, OpenCV, PCL. A short introductions of these libraries are given below.

5.2.6. ROS

It stands for robot operating System and is a flexible framework for writing robot software. The framework provides tools, libraries, and conventions that goal to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms. ROS is developed and maintain by the company called willow garage. With the contributions of a lot of researchers the framework become a standard within the robotic research community.

5.2.7. OpenCV

The open source computer vision (OpenCV) library provides all the necessary classes and function to process the 2D image. The library contains the algorithms to simple filter the image to advanced object detection algorithms. It also provides algorithms and approaches for machine learning problems.

5.2.8. PCL

The point cloud library (PCL) is a framework which provide functionality to work with depth image (point cloud) taken from senor like kinect. It contains algorithms to simple task like filter out unwanted portion of the cloud to extract single objects from the cloud.

5.2.9. Caffe

6. Experiments and Evaluations

In this chapter we will illustrate how the implemented learning system was tested and performance was evaluated.The subsequent sections will first focus on the composition of the test data and how it was gathered. Next, there will be a brief description of how the tests were conducted and how the results were extracted from the system. Finally there will be an assessment of the test results and the quality of the learning data produced for the single test sets with the different algorithms.

6.1. Data Acquisition

In this thesis work we use two datasets, kitchen environment dataset (ODU) from Institute for Artificial Intelligence and large scale hierarchical multi-View RGB-D object dataset [**?**] from University of Washington. For our experiments, we use subset of ODU dataset, where 54 classes of objects are considered. The names of the objects instances are given below.

- albi_himbeer_juice
- black_bowl
- blue_camping_cup
- blue_cup
- blue_plastic_bowl
- blue_spotted_plate
- cappuccino
- coffee_el_bryg
- cup_eco_orange
- fork_blue_plastic
- fork_red_plastic
- fork_ycb
- frying_pan
- frying_spatula
- hela_curry_ketchup
- ja_milch
- jod_salz
- kelloggs_corn_flakes
- kelloggs_toppas_mini
- knife_blue_plastic
- knife_red_plastic
- knife_ycb
- knusper_schoko_keks
- large_grey_spoon
- linux_cup
- lion_cereal
- marken_salz
- meer_salz
- mondamin
- nesquik_cereal
- pancake_maker
- pitcher_blue
- pitcher_white
- pitcher_yellow
- pot
- pfanner_pfirsch_icetea
- pfanner_grune_icetea
- pringles_paprika
- pringles_salt
- pringles_vinegar
- red_spotted_cup
- red_spotted_bowl
- red_spotted_plate
- reine_butter_milch
- sigg_bottle
- slotted_spatula

- soja_milch
- spitzen_reis
- toaster
- tomato_al_gusto_basilikum
- tomato_sauce_oro_di_parma
- voll_milch
- white_bowl
- yellow_plate

Besides the object instance, we are curious about object shape classifications because shapes are important source of informations [7]. The shape categories are as follows.

- box
- cylindrical
- disk
- flat
- sphere
- other

On the other hand, The RGB-D Object Dataset contains visual and depth images of 300 physically distinct objects arranged in 51 categories. For example, apple is an category, which contains 5 different objects or instances means they are also apple but different ones. The objects in the dataset are commonly found in home and office environments, where personal robots are considered to operate. In the dataset the objects are organized into a hierarchy taken from WordNet hypernym/hyponym relations and is a subset of the categories in ImageNet [6]. The hierarchy as shown by the following figure, where fruit and vegetable are top-level subtrees in the hierarchy. Each of the 300 objects in the dataset belong to one of the 51 leaf nodes in this hierarchy, with between three to fourteen instances in each category. The leaf nodes are shaded blue and the number of object instances in each category is given in parentheses.

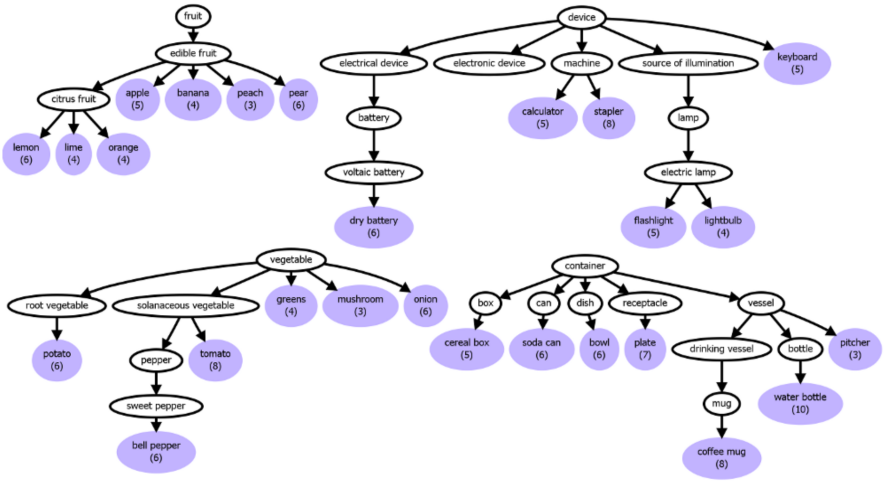

Figure 6.1.: Object hierarchy in the dataset as a subset of the ImageNet category. Taken from [?]

The following figure shows objects from the dataset, where each object belong to a different category.

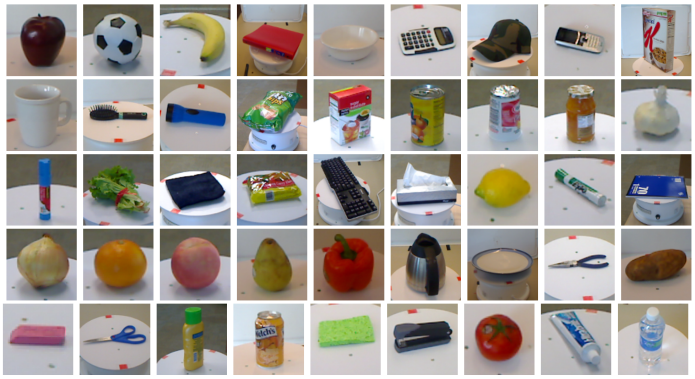

Figure 6.2.: Objects from the RGB-D object dataset. Taken from [6]

Both of the dataset are created using similar procedure. The video sequence of each object is recorded as the object is spun around a turntable, where cameras mounted at three different heights relative to the turntable, at approximately 30°, 45° and 60° above the horizon. Farther and most importantly, we generate new data using our developed model and use these data to train the model and evaluate the model.

6.2. Test procedure

In this experiment we will reveal, how the accuracy of the classifier increases by a trained model, which contains the data generated from episodic memory.

For this task, at first we train classifiers on ODU dataset, which contains 8000 samples of 54 objects.The data is produced using the turn table. At this point we test the classifier accuracy with a small dataset contains 480 samples of 25 objects and get the over all classifier accuracy 63%. Each time we test the accuracy of the classifier using different small dataset and in the next step we add this data with the previous trained dataset. The dataset is generated from robot manipulation task. where each there the new test dataset is the subset of previous test dataset and contains the data of robots manipulation tasks. At first we test the classifier accuracy using date taken from robot manipulation task in kitchen environment. The test data has 240 samples of 21 objects. Here we get the classifier accuracy is 63

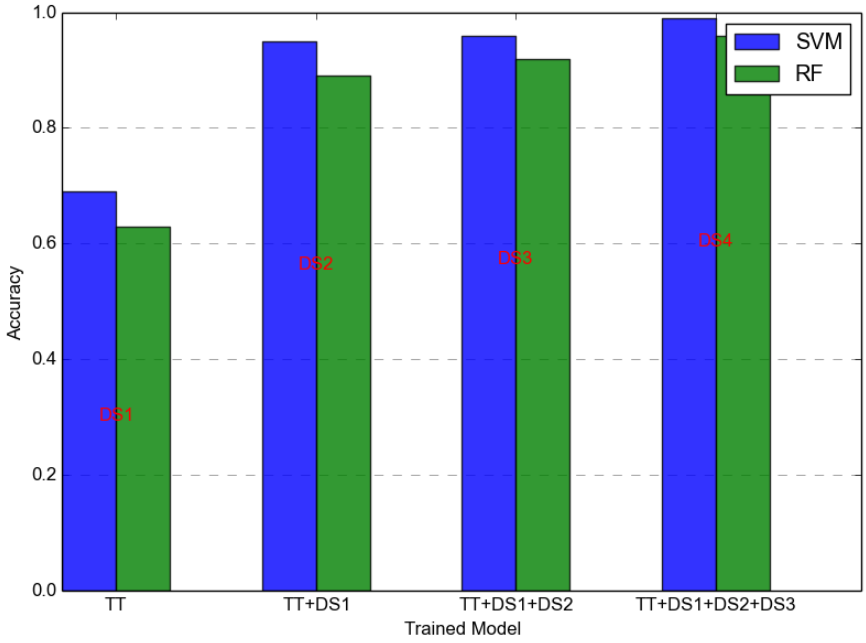

Figure 6.3.: Categorization of WRGBD dataset

6.3. Test

In the first experiment, we will reveal how much train data is needed to get high accuracy of the different classifiers. For this task, we train classifiers on ODU dataset, which contains 8000 samples of 54 objects. Each time we test the accuracy of the classifier using different small dataset and in the next step we add this data with the previous trained dataset. The dataset is generated from robot manipulation task. where each there the new test dataset is the subset of previous test dataset and contains the data of robots manipulation tasks. At first we test the classifier accuracy using date taken from robot manipulation task in kitchen environment. The test data has 240 samples of 21 objects. Here we get the classifier accuracy is 63

6.3.1. Supervised Results

In this thesis work we consider that the robot is working in the kitchen environment and the task of the robot is like pick and place of the object. We use Washington university dataset [**?**] and kitchen environment dataset from

institute for artificial intelligent. Both datasets are created by taken image of different objects using a turn table with and angle difference 30 degree. We use popular supervised machine learning algorithms like support vector machine, random forest, k-nearest neighbor, gradient boost tree to create the trained model in order to classify the data which is coming from real world robot manipulation task. Before classify the real world data we cross validate our trained model using the test data and evaluate the performance of the classifiers. The real world data is recorded during the robot manipulation task and put it into a database. The real data is row sensor date and consider as unstructured data.

6.3.2. Semi-supervised Results

Table 6.1.: Algorithm's result comparison

Algorithm / **Feature+Dataset**	**SVM**	**RF**	**RF**	**GBT**
BVLC_7CF + RGBD	foo	bar	foo	bar
BVLC_7CF + iai_ODU	foo	bar	foo	bar
VFH + RGBD	foo	bar	foo	bar
VFH + iai_ODU	foo	bar	foo	bar
CVFH + RGBD	foo	bar	foo	bar
CVFH + iai_ODU	foo	bar	foo	bar
VGG16 + WURGBD	foo	bar	foo	bar
VGG16 + iai_ODU	foo	bar	foo	bar

7. Conclusion

A. Appendix

The usages of all the developed softwares are illustrated below.

Prerequisite: Robosherlock, Caffe, PCL, openCV, rs_resources

The **rs_learning** package consists of three modules.

- Module for extracting features from image.
- Module for training the classifier.
- Module for classify the images.

A.1. Extracting Feature Module

To get help:

$ rosrun rs_learning featureExtractor -h

To extract feature

$ rosrun rs_addons featureExtractor -s splitName -i storage -d datasetName -f feat

Where:

splitName It is a .ymal file, contains informations about objects and object's class label. The file should be in catkin workspace rs_resources/objects_dataset/splits folder.

storage It is the name of input image storage folder.The folder should be in rs_resources/objects_dataset. For this project we use two datasets, one is kitchen environment dataset from Institute for Artificial Intelligence and the other one is from Washington University dataset. Parameter's (storage) value should be iaiImageFolder/wuImageFolder, if someone wants to use both datasets at once else just iaiImageFolder or wuImageFolder folder name.

datasetName It's value should be should be IAI (to use dataset from Institute for artificial intelligence) and WU (to use dataset from Washington University). If someone wants to use both datasets at once, he should select the parameter's value as BOTH.

feat It should be CNN or VGG16 (RGB data) and VFH or CVFH (for RGB-D data).

The above command should generate following files in rs_resources/objects_dataset/extractedFeat folder. So check the folder called extractedFeat is there or not, if not create one and name it as extractedFeat.

1. datasetName_feat_ClassLabel_splitName.txt
2. datasetName_feat_MatTrain_splitName.yaml
3. datasetName_feat_MatTrainLabel_splitName.yaml
4. datasetName_feat_MatTest_splitName.yaml
5. datasetName_feat_MatTestLabel_splitName.yaml

The first file contains the object to class label map in data type double, second and third file contain the train data and it's label respectively, fourth and fifth file contain test data and it's label accordingly.

Example: If someone type the following command:

$ rosrun rs_learning featureExtractor -s ObjectOur -i partial_views -d IAI -f CNN

Output should be following:

1. IAI_CNN_ClassLabel_ObjectOur.txt
2. IAI_CNN_MatTrain_ObjectOur.yaml
3. IAI_CNN_MatTrainLabel_ObjectOur.yaml
4. IAI_CNN_MatTest_ObjectOur.yaml
5. IAI_CNN_MatTestLabel_ObjectOur.yaml

A.2. Classifier Trainer Module

In rs_learning package each annotator has one .xml file in Descriptors/annotators folder and the ensemble of annotators is called analysis engine. If someone wants to create the TrainedModel for the specific classifier,should first provide the following parameter's values in trainerAnnotator.xml file.

classifierType It's value should be rssvm (for support vector mechine) or rsrf (for random forest) or rsgbt (for gradient boost tree) or rsknn (for k-Nearest neighbour).

trainDataName The name of the train data file (datasetName_feat_MatTrain_ObjectOur) in path rs_resources/objects_dataset/extracetedFeat

train_label_name The name of the data trainLabel file (datsetName_feat_MatTrainLabel_splitName) in path rs_resources/objects_dataset/extracetedFeat/

It will generate a TrainedModel file as datasetName_feat_classifierTypeModel_ObjectOur.yaml in rs_learning/trainedData folder.

Example: If someone choose parameters classifierType as rssvm, trainDataName as IAI_CNN_MatTrain_ObjectOur and train_label_name name as IAI_CNN_MatTrainLabel_ObjectOur in trainerAnnotator.xml file and type the following command on Ubuntu terminal.

$ rosrun robosherlock run model_trainer

Then as output IAI_CNN_rssvmModel_ObjectOur should be generated in rs_learning/trainedData folder.

A.3. Image Classifier Module

This module is divided into two parts classify offline and online. If someone has test data on hand, he can use ClassifyOffline annotator and classifies the images. The command for that:

$ rosrun robosherlock run ClassifyOffline

Before enter the command please tune the following parameter in ClassifyOfflineAnnotator.xml file.

classifierType It should be rssvm or rsrf or rsgbt or rsknn

trained_model_name The name of the trainedModel file (Ex. if someone selects classifier_type(= rssvm),then traindModel should look like IAI_CNN_rssvmModel_ObjectOur).

test_data_name It should be the test data file name (Ex.IAI_CNN_MatTest_ObjectOur.yaml)

test_label_name The name of the testLabel data file (Ex.IAI_CNN_MatTestLabel_ObjectOur)

actual_class_label The name of classLabel file (Ex.IAI_CNN_ClassLabel_ObjectOur)

If the classifier_type (=rsknn), instead of trained_model_name selects the following two files.

trainDatamatrix The name of the train matrix file (Ex.IAI_CNN_MatTrain_ObjectOur)

trainlabel_matrix The name of the trainLabel matrix file (IAI_CNN_MatTrainLabel_ObjectOur)

If test data is coming from a .bag file or any database or from real time robot manipulation task, the process is called online. Then someone has to use the following command.

$ rosrun robosherlock run my_demo

my_demo is an analysis engine with many Annotators(specially classifiers). Each classifier has two options. It can classify or set the groundTruth for the images. So before runing the above command please tune the parameters in the respective annotator's .xml file. The parameters name are same for classifiers (rssvm, rsrf, rsgbt) and they are:

set_mode It should be CL (to classify) and GT (to set groundtruth)

trained_model_name Name of the trainedModel (Ex.IAI_CNN_rssvmModel_ObjectOur).

actual_class_label Name of classLabel file
(Ex.IAI_CNN_ClassLabel_ObjectOur)

And for classifier (rsknn), please selects set_mode (=rsknn) and instead of parameter (trained_model_name) tune the the following parameters.

trainKNN_matrix The name of the train matrix file
(Ex.IAI_CNN_MatTrain_ObjectOur)

trainKNNLabel_matrix The name of the trainLabel matrix file
(IAI_CNN_MatTrainLabel_ObjectOur)

Attention: When classify images online please make sure that the image features coming from Robosherlock annotators pipeline (Ex. PCLfeatureExtractor or caffe) must be the same as the respective trainedModel's features.

B. List of Figures

C. List of Tables

D. Bibliography

[1] Aldoma, Aitor, Markus Vincze, Nico Blodow, David Gossow, Suat Gedikli, Radu Bogdan Rusu und Gary Bradski: *CAD-model recognition and 6DOF pose estimation using 3D cues*. In: *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, Seiten 585–592. IEEE, 2011.

[2] Balint-Benczédi, Ferenc, Zoltán Csaba Márton, Maximilian Burner und Michael Beetz: *Storing and retrieving perceptual episodic memories for long-term manipulation tasks*. In: *Advanced Robotics (ICAR), 2017 18th International Conference on*, Seiten 25–31. IEEE, 2017.

[3] Beetz, Michael, Ferenc Bálint-Benczédi, Nico Blodow, Christian Kerl, Zoltán Csaba Márton, Daniel Nyga, Florian Seidel, Thiemo Wiedemeyer und Jan Hendrik Worch: *RoboSherlock: Unstructured Information Processing Framework for Robotic Perception*. In: *Handling Uncertainty and Networked Structure in Robot Control*, Seiten 181–208. Springer, 2015.

[4] Ferrucci, David, Eric Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya A Kalyanpur, Adam Lally, J William Murdock, Eric Nyberg, John Prager *et al.*: *Building Watson: An overview of the DeepQA project*. AI magazine, 31(3):59–79, 2010.

[5] James, Gareth, Daniela Witten, Trevor Hastie und Robert Tibshirani: *An introduction to statistical learning*, Band 112. Springer, 2013.

[6] Lai, Kevin, Liefeng Bo, Xiaofeng Ren und Dieter Fox: *A large-scale hierarchical multi-view rgb-d object dataset*. In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, Seiten 1817–1824. IEEE, 2011.

[7] Marton, Zoltan Csaba, Ferenc Balint-Benczedi, Florian Seidel, Lucian Goron und Michael Beetz: *Object categorization in clutter using additive features and hashing of part-graph descriptors*. Spatial Cognition VIII, Seiten 17–33, 2012.

[8] Niemueller, Tim, Nichola Abdo, Andreas Hertle, Gerhard Lakemeyer, Wolfram Burgard und Bernhard Nebel: *Towards deliberative active perception using persistent memory.* In: *Proc. IROS 2013 Workshop on AI-based Robotics*, 2013.

[9] Niemueller, Tim, Stefan Schiffer, Gerhard Lakemeyer und Safoura Rezapour-Lakani: *Life-long learning perception using cloud database technology.* In: *Proc. IROS Workshop on Cloud Robotics*, 2013.

[10] Oliveira, Miguel, Gi Hyun Lim, Luís Seabra Lopes, S Hamidreza Kasaei, Ana Maria Tomé und Aneesh Chauhan: *A perceptual memory system for grounding semantic representations in intelligent service robots.* In: *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, Seiten 2216–2223. IEEE, 2014.

[11] Rusu, Radu Bogdan, Gary Bradski, Romain Thibaux und John Hsu: *Fast 3d recognition and pose using the viewpoint feature histogram.* In: *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, Seiten 2155–2162. IEEE, 2010.

[12] Wood, Rachel, Paul Baxter und Tony Belpaeme: *A review of long-term memory in natural and synthetic systems.* Adaptive Behavior, 20(2):81–103, 2012.