FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

# Optimisations and Improvements of Topographic Robotic Grasping

## Using Machine Learning Approaches

DIPLOMARBEIT

zur Erlangung des akademischen Grades

**Dipl.-Ing**

im Rahmen des Studiums

**Computational Intelligence**

eingereicht von

**Soroosh Mortezapoor**
Matrikelnummer 1225049

an
der Fakultät für Informatik der Technischen Universität Wien
und der Fakultät für Elektrotechnik und Informationstechnik

Betreuung: Vincze, Markus; Ao.Univ.Prof. Dipl.-Ing. Dr.techn.
Mitwirkung: Fischinger, David; Projektass. Dipl.-Ing. Dr.techn.

Wien, TT.MM.JJJJ

(Unterschrift Verfasser) (Unterschrift Betreuung)

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

# Optimisations and Improvements of Topographic Robotic Grasping

## Using Machine Learning Approaches

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

**Dipl.-Ing**

in

**Computational Intelligence**

by

**Soroosh Mortezapoor**
Registration Number 1225049

to the Faculty of Informatics,
and the Faculty of Electrical Engineering and IT
at the Vienna University of Technology

Advisor: Vincze, Markus; Ao.Univ.Prof. Dipl.-Ing. Dr.techn.
Assistance: Fischinger, David; Projektass. Dipl.-Ing. Dr.techn.

Vienna, TT.MM.JJJJ

(Signature of Author) (Signature of Advisor)

# Acknowledgements

I would like to express my gratitude to my supervisor, Prof. Markus Vincze, for his outstandingly useful comments, remarks, and engagement through the learning process of this project. Furthermore, I would like to express my sincere appreciation to my co-adviser, Dr. David Fischinger, whose engagement through the entire process and guidelines, made the project possible.

Special thanks to Prof. Horst Eidenberger, for teaching me how human vision works and helping me to find a better understanding of media processing.

Additionally, I would like to thank my loved ones, who have supported me from the first moment thought the last, both by keeping me harmonious and helping me putting pieces together. I will be grateful forever for your love.

# Abstract

Even though grasping has been an interesting and crucial topic in robotics for a long time, robots still have great difficulty in picking up arbitrary objects when they face open, unknown environments or under uncontrolled conditions. Solving these issues would make using robots fit for many repetitive scenarios such as checking and restock supermarket aisles, tidying up households, dispatching mail orders at distribution centres, collecting ripe fruits, or doing ecological pest control by selectively removing bugs. To become reality, robots have to learn to grasp as reliable as humans. Many people have tried to discover the tricks people unconsciously employ when they cannot rely on their perception, and transfer these insights to robots.

As a result, a new research and development conducted with a new vision to this problem introducing a novel approach for training a robot how to grasp, using topographic features of the objects, especially developed for grasping without requiring a prior knowledge of the objects, called Height Accumulated Features (HAF) and Symmetry Height Accumulated Features (SHAF). This method abstracts topographic information from perceived surfaces of objects hence enables to learn how to grasp them, even if they are unknown or on a heap of other objects.

An important and inseparable part of the core of such an approach is the decision part, where the perceived scenes are interpreted into abstract models and need to be classified, to help the rest of the core to make a conclusion and find the best suitable points to grasp an object. This thesis proposes improvements and optimisations on this part of the core, in three different stages, extending machine learning training instances, introducing new feature set definitions, to be used as either substitutions or complements to HAF and SHAF, and selecting the best classification algorithm, to be employed for the grasping using topographic features.

Three new feature set definitions, Circular Feature Definition (CF), Differential Intra-Circular Feature Definition (DICF) and Bell-Circular Feature Definition (BCF) try to summarize the scene in an abstract form with enormously less data dimensions, compared the combination of HAF-SHAF, not only to maintain and even improve the accuracy of the outcome, but also to increase efficiency by reducing the computation complexity.

Furthermore, the extended data set is used in order to put the new features, as well as HAF and SHAF into test, with different classification algorithms including Support Vector Machines, Decision Trees, $k-$Nearest Neighbors and Random Forest, in order to find the best combination of (feature set definition, classification algorithm) for the problem of grasping using topographic features. Finally, the results of the experiments are presented and compared to provide more insight into the performance of each setting.

# Kurzfassung

Obwohl Greifen seit langer Zeit ein interessantes und elementares Thema in der Robotik ist, haben Roboter immer noch große Schwierigkeiten beliebige Objekte aufzunehmen, wenn diese sich in unbekannten Umgebungen befinden, oder unkontrollierte, offene Bedingungen herrschen. Durch die Lösung dieser Probleme, würden sich Roboter dazu eignen repetitve Aufgaben zu erfüllen, wie zum Beispiel die Kontrolle und das Wiederauffüllen von Supermarktregalen, das Aufräumen von Haushalten, die Ablieferung von Postsendungen bei Verteilerzentren, das Sammeln reifer Früchte oder in ökologischen Szenarien durch selektives entfernen von Schädlingen. Um dies Wirklichkeit werden zu lassen, ist es essenziell, dass Roboter lernen Objekte mit der selben Zuverlässigkeit zu greifen wie der Mensch. Viele haben versucht die unbewussten Tricks, welche Menschen anwenden wenn sie sich nicht auf ihre Wahrnehmung verlassen können, zu entschlüsseln und diese Methoden auch auf Roboter anzuwenden.

Als Ergebnis, wurde ein neuer Ansatz in Forschung und Entwicklung eingeführt um den neuen Visionen gerecht zu werden. Dieser nutzt topographische Merkmale der zu greifenden Objekte, welche keine vorherigen Kenntnisse der Begebenheiten der Gegenstände erforden. Die dazu eingesetzten Methoden heißen Height Accumulated Features (HAF) und Symmetry Height Accumulated Features (SHAF). Diese Methode abstrahiert topographische Informationen aus wahrgenommen Oberflächen von Gegenständen und ermöglicht dadurch das Greifen dieser. Selbst wenn die Gegenstände unbekannt sind oder mehrere verschiedene auf einander liegen, funktioniert diese Methode.

Ein extrem wichtiger Teil eines solchen Ansatzes ist die Entscheidungsfindung in einem Teil des Kerns. In diesem werden die wahrgenommenen Szenen in abstrakte Modelle interpretiert und klassifiziert um den Rest des Kerns beim finden von den besten Angriffspunkten zu unterstützen.

Diese Arbeit schlägt Verbesserungen und Optimierungen in diesem Teil des Kerns, in drei verschiedenen Stufen vor. Diese enthalten erweitertes maschinelles lernen, mit neu eingeführten Feature-Set Definitionen, entweder zur Substitution oder Unterstützung von HAF und SHAF um den besten Klassifikationsalgorithmus auszuwählen, welcher Anwendung beim Greifen mit topographischen Merkmalen findet.

Drei neue Feature-Set Definitionen, Circular Feature Definition (CF), Differential Intra-Circular Feature Definition (DICF) und Bell-Circular Feature Definition (BCF), versuchen die Szene mit einer sehr geringen Datendimensionen in abstrakter Form zusammenzufassen. Verglichen mit der Kombination von HAF-SHAF wird hier nicht nur die Genauigkeit der Ergebnisse verbessert, sondern auch die Effizienz verbunden mit einer Verringerung der Rechenkomplexität.

Darüber hinaus wird der erweiterte Datensatz dazu verwendet, um die neuen Funktionen sowie HAF und SHAF zu testen. Dafür werden verschiedene Klassifikationen und Algorithmen einschließlich Support Vector Machines, Decision Trees, kNearest Neighbors und Random Forest verwendet um das Problem, greifen mit topographischen Merkmale, zu lösen. Schlussendlich wurden die Ergebnisse der Experimente dargestellt und verglichen, um einen besseren Einblick in die Performance der einzelnen Einstellungen zu bekommen.

# Contents

# List of Tables

# List of Figures

CHAPTER 1

# Introduction

There are plenty of objects with which a normal human being interacts every day. These objects are designed with a particular way of being grasped and manipulated that is determined and characterized by their shape and the contact constraints. Consequently, if robots are supposed to replace humans in a wide variety of tasks, ranging from a dangerous situation in search and rescue to repetitive everyday activities, the capability of grasping different types of objects is a must to be taken into account. Furthermore, there are so many challenges in the way of making a robotic hand capable of grasping differently shaped objects in an uncertain, semi-structured and in many cases unstructured real environment. For example, a noteworthy challenge that could be mentioned is to find a proper pose for starting the procedure of grasping.

There have been so many research projects conducted to tackle such problems and they proposed significantly different approaches, each presuming a lot of different criteria. From these approaches, some tried to rely on 3D object recognition algorithms [26] [2] while some others tried to use shape primitives such as boxes, spheres, etc. to approximate the new object [21] [17] [29] [10].

Among all approaches which tried to address the problem of grasping, a state-of-the-art approach for grasping using a robotic hand proposes to use topographic features [8] [9] [7]. In topographic feature robotic grasping approach, called *TFRGA* hereafter in this essay, two feature types based on the topography of objects or scenes are introduced and motivated in order to extract topographic features of the scene for further process.

Moreover, almost every scientific approach deals with many parameters and variables to which making changes will result in a different outcome in terms of quality [34]. Thus picking the best setting for the parameters of these approaches is significantly important. In other words, an approach can have different sets of results depending on their parameter settings. In this project, we try to design and implement a tool which is capable of evaluating the outcome of TFRGA in order to reach a better setting for the parameters that are used in this approach. In addition, some new extensions are proposed, introduced and evaluated for TFRGA.

In this chapter, first it is tried to describe the optimisation problem of the TFRGA. Next, the contribution of this project is briefly introduced, followed by the chapter on related work.

Subsequently, the theoretical contribution of this thesis is discussed in more depth. Finally, the practical result of this project are presented along with a discussion on the obtained results and a conclusion is made. When necessary, figures and tables are utilized to demonstrate some parts of the contents of the chapters.

## 1.1 Problem Statement

TFRGA tries to address the problem of grasping objects. This problem could informally be defined as a problem of finding a proper pose, for a robotic hand, in a multi-dimensional grasp space, incorporating position, orientation and respectively, gripper opening width, that ensures a stable grasp. Additionally, the solution should include the trajectory by which the robotic hand can reach the desired pose. Besides, a stable grasp is defined as a type of grasp by which a robotic hand is able to pick an object with a reasonably low probability of dropping it due to a wrong grasping pose. Definition 1 shows a formal definition of this problem. As stated in Definition 1, the solution to this optimisation problem looks like a $7-$dimensional tuple $(x, y, z, R, P, Y, w)$ which determines the position $(x, y, z)$, the orientation $(R, P, Y)$, and the opening width of the gripper $(w)$.

DEFINITION 1: PROBLEM STATEMENT

**INSTANCE:**
A point cloud $C$ of a scene $S$, A stack of objects $T$ with `top(T)` as the top element of the stack, a robotic manipulator $M$.

**PROBLEM:**
Given the point cloud $C$ of the scene $S$, find a tuple $(x, y, z, R, P, Y, w)$ that maximizes $P$, the probability of grasping the object `top(T)` as a stable grasp for the manipulator $M$.

In most of the autonomous robotic interactions with semi-structured or unstructured probabilistic uncertain environments, an abstract Perception-Decision-Action cycle [14] can be considered that ensures the proper actions by the robots in different circumstances and helps the robots to stay tuned with the latest changes in the environment. We regard the grasping procedure in TFRGA as a similar type of activity. Figure 1.1 depicted this abstract system of interacting with surrounding environment. As represented in this figure, in the first stage, Perception, the robot perceives the environment through its diverse sensors. Then as soon as the information is gathered, it is transmitted to a processing unit that we call the Planner, in the second stage, Decision. When we talk about Decision unit, in fact, we talk about a unit in which all activities, since the environment is perceived by the corresponding sensors till a complete plan is ready for a physical execution are planned. In the Planner, a plan containing all information about the reaction to the perception is generated and the answer of the system in the form of some consec-

utive commands is transmitted to the actuators for the third stage, Action. TFGRA follows the same principle. An important computation step in the Decision stage of TFGRA, can be seen as a function, whose input is a perceived scene and its output is a three-value tuple, $(x, y, z)$, reference to Definition 1, that repeatedly calls a decision function for binary answers with altered selected frames from a big scene, indicating whether two grasp-suitable spots, as shown on the bottom left corner of Figure 1.1 as two red spots, can be found, where it's improvement and optimization is the main contribution of this project.

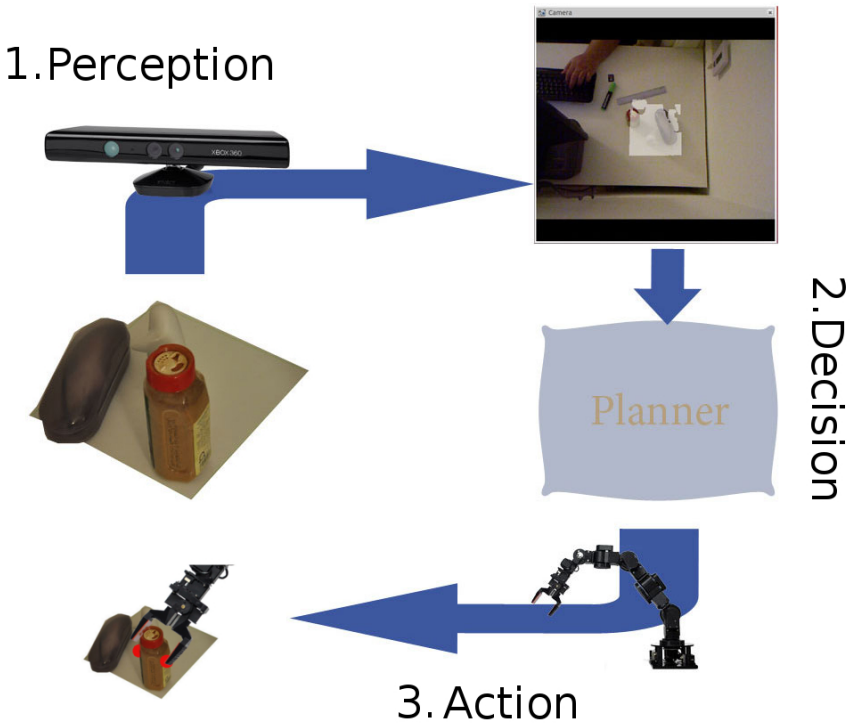



*Figure 1.1: Abstract stages of interaction of a robot with its surrounding environment. In the Perception stage, the robot perceives the environment. Then the information is transmitted to the Planner, for a Decision phase, where it is decided what the response to the environment should look like, and the reaction commands are transmitted to the actuators for execution as the final stage, Action. These three stages are done in a loop which means that after each action, the robot goes into the Perception stage to have a new understanding of the surrounding environment. The main contribution of this project will impact the Decision stage. The red spots are the gripper's fingers locations for grasping.*

The Planner is responsible for processing the input data and generating some output commands based on this input. Figure 1.2 illustrates the stages which should be done in order to turn some perceived data, input, to the corresponding output commands. As it is shown in this figure, first the input data in the form of a point cloud is passed to the Planner. From this point cloud, the Planner creates a $14 \times 14$ grid called a *Height Grid* which is illustrated in figure 1.3 holding the highest measured point in a 1 $cm^2$ cell of a 196 $cm^2$ scene. Next, from the Height Grid, some feature vectors are extracted that form a feature space, helping to summarise the

scene. Immediately after extracting each feature vector, a classifier, trained by a binomial set of instances of feature vectors for grasping scenes is utilized to categorise the perceived scene. As a result, depending on the outcome of classification, the Planner tries to find a proper trajectory for grasping.

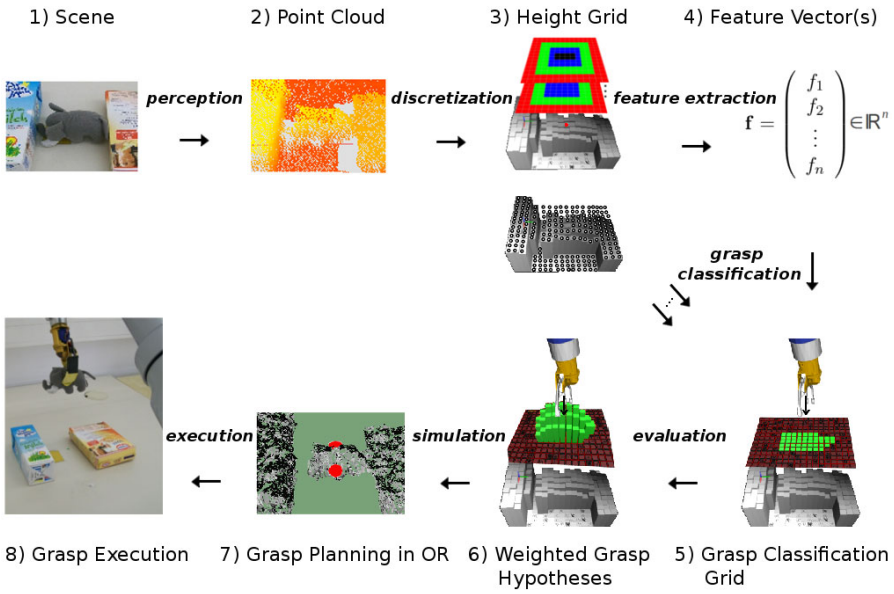



*Figure 1.2: This figure shows a complete interaction cycle of TFRGA which is grabbed from [9]. The stages 2 through 7 are done in the Planner. This project tries to optimize the stages 4 to 6.*

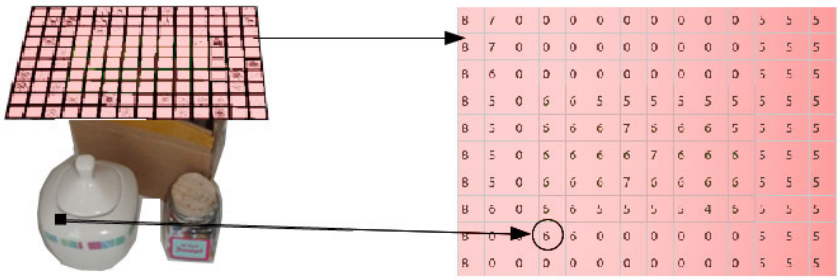


*Figure 1.3: $14 \times 14$ Height Grid. Each cell corresponds to $1\ cm^2$ and holds the height of the highest measured point in that cell in the perceived scene.*

TFRGA, defines two feature types based on the topography of objects or scenes, Height Accumulated Features, HAF for short, which are developed specifically for abstracting relevant information of a grasp, and Symmetry Height Accumulated Features, SHAF for short, as an additional feature type, to deal with the cases that a classification model, which is trained with HAF instances seems to be unable to classify correctly. The basic concept of HAF was first

published in Fischinger et al. [8] whilst the concept of SHAF proposed later as a complement in Fischinger et al. [9]. These two concepts are explained more in details in following chapters of this essay.

In this project, it is tried to optimise the process by which the Planner of TFRGA decides what position shall be used for grasping. Reference to Definition 1, we are mainly interested in first three elements of an answer to the optimization problem which mean $(x, y, z)$. Furthermore, we consider no correlation between these tree elements and the remaining elements in an answer tuple. In other words, the elements correspond to position, orientation, and the opening width are considered independent of each other. Hence they can be optimized independently and the problem of ensuring consistency is left out of this project. In following chapters we discuss how the evaluation on the gained results of the Planner is performed based on HAF and SHAF. Additionally, we propose some new feature definitions with particular properties to be used in combination or as a substitute for HAF and/or SHAF. Besides, we try to utilize different classification methods and compare them to find the best matching classifier for the discussed domain.

## 1.2 Contribution of Project

There is a wide variety of aspects for optimising a process. Besides, here in our case, the optimisation process could be applied in different parts of the Decision Stage, which was depicted in Figure 1.1, in Planner's decision procedure. Defining *decision quality* in the current context as the accuracy of binomial decision making of the Planner in the form of (YES|NO) answer to the question of "Is the point $(x, y, z)$ a good position for grasping" in an observed scene $S$, the goal of this project can be formulated as identifying a set of candidate positions for grasping to maximize the decision quality. To be more specific, a larger set of points in a $3-$dimensional space is selected by the Planner to find a proper candidate subset, by giving to the trained classifier to be examined once at a time.

The very first problem to be addressed is how to measure the degree of success achieved by applying any type of change in the parameters and processing stages of our optimisation problem. In order to measure the quality of decisions by selecting different settings of parameters for the Planner, a new software is developed in the frame of this project in order to facilitate the process of evaluation.

Having such an evaluation software developed and ready, the process of making decisions can be inspected, evaluated and consequently optimized in four different levels. Initial instances used for training the classifier, extracted features, classification algorithm, and the parameters to customize the selected classifier algorithm. Following sections show the operations which are performed in each of these level.

### Training Instances

The first step of the optimisation is to gather more instances of both successful and failed cases of grasping. The more the diversity of instances of scenes as examples of success and failure, the better and more precise the result of a classifier. This phase corresponds to the stages 1 and 2 of

Figure 1.2. In order to capture more grasping scenes and turning them into classification training instances, some prerequisites should be met which are explained in more details in Chapter 4.

### Features

As mentioned before, TFRGA introduces two different types of features in order to cover the different scenarios of detecting the suitable poses of grasping. In this project, more features are defined, having different properties, to deal with the aspects at which HAF and SHAF can be optimized. These features try to help the classifier to deal with various new scene instances to increase the quality of classification. These features are introduced and explained later in this report and the results obtained by using these features are presented and compared to other cases.

### Classifier

One of the question that always arises in classification tasks is to find the most appropriate classification algorithm for the problem [33] [1] [3]. There are so many approaches to evaluate the outcome of different classifiers on the same problem [40] [39] [18]. Nonetheless, the first step toward using any of these approaches is to have an application capable of applying each classification algorithm, on demand, on any desired dataset. This is the reason why it is decided to take the flexibility of plugging different classification algorithms into consideration in designing and implementing the evaluation software.

In evaluation application, there are different options to use many famous and widely used classifiers including k-NN, Decision Trees, Random Forest, Stochastic Gradient Descent (SGD) and Support Vector Machines (SVM). This way it would be possible to apply different algorithms on the same dataset and observe the outcome and consequently pick the best algorithm matching the problem.

### Classifier Parameters

Another important aspect in optimization of a classification algorithm is to find a set of parameters that help the algorithm to output more accurate results. These parameters are capable of varying the behavior of a classification algorithm significantly. In order to be able to manipulate the behavior of the algorithms mentioned before, the evaluation application is implemented in a way that allows altering parameters of the selected working algorithm easily from the user interface. Changing the working underlying algorithm in each run cause changes in the number of parameters and their characteristics depending on the chosen algorithm.

## 1.3 Outline

Following the current chapter, in Chapter 2, some related work around the current problem, mainly the idea behind the approach for grasping using Topographic Features is discussed. After that, the theoretical contribution of this thesis project is described in Chapter 3, comprising introducing new features in order to help classifying the observed scenes more precisely, preprocessing steps on the training dataset, and a detailed plan for experiments on selected classifi-

cation algorithms with different configuration settings. Subsequently, in Chapter 4 practical parts of the project are explained, the architecture and properties of the test application are described, steps which are taken in order to achieve the ultimate goal of the project are demonstrated and the obtained results are presented. Furthermore, Chapter 4 is finished by a summary over all gained experimental results. Finally a conclusion, in Chapter 5, is made, out of what is done in this project, to show the impact of the project on gaining better results in TFGRA, followed by sketching the future plan.

CHAPTER 2

# Related Work

Turning light into concepts and connecting them to visually understanding features and objects, that usually involves distilling foreground from background, recognizing objects, which can be presented in a wide range of positions and orientations, and interpreting spatial cues accurately, is a complex and considerably complicated task that a human vision system is capable of. As mentioned in Chapter 1, so many people tried to solve the vision problem for robots with various approaches.

## 2.1 Topographic Features

It is already mentioned that, TFRGA introduces two different features, HAF and SHAF. HAF tries to detect scenes that share similar characteristics by means of comparing average heights of the different regions in a $14cm \times 14cm$ discretized point cloud data as a $14 \times 14$ grid. In other words, a HAF can be seen as a linear function over two, three or four variables as one HAF is defined as a two, three or four-region $14 \times 14$ mask over the Height Grid, having an independent weight for each region. Considering each cell $c_{ij}$ of the Height Grid as the highest measured point of the point cloud data in that cell, $R_k$ as the $k-$th defined region, $\downarrow i_{R_k}$ and $\downarrow j_{R_k}$ as the starting indices and $\uparrow i_{R_k}$ and $\uparrow j_{R_k}$ as the ending indices of $i$ and $j$ in the $k-$th region, and finally $w_k$ as the weight of the corresponding region, the HAF value of a Height Grid, $HAF(C_s, R)$ for the point cloud of the scene, $C_s$, and the regions definition $R$ is calculated as shown in ( 2.1).

$$HAF(C_s, R) = \sum_{k=1}^{\#R} \sum_{i=\downarrow i_{R_k}}^{\uparrow i_{R_k}} \sum_{j=\downarrow j_{R_k}}^{\uparrow j_{R_k}} c_{ij}.w_k \tag{2.1}$$

Figure 2.1 depicts a HAF with three different regions as a mask over a Height Grid. However in Fischinger et al. [7] it is mentioned that HAF may misclassify some certain type of scenes. Typically these scenes are those in which one tall object located at center but from one side it is expanded too long to be fit in a gripper for grasping. Thus TFRGA introduces another type of feature called Symmetry Height Accumulated Features, SHAF. In SHAF, very similar to HAF, there are some regions but they are always disjunctive and sized equally. A simple SHAF as a mask on a Height Grid is shown in Figure 2.2. This SHAF comprises three different regions where each region has its own weighting factor. As it was mentioned before, unlike HAF, SHAF is diminishing which means that the blue and the red areas are subtracted from the green area.

Furthermore, there is another important difference between HAF and SHAF which is the difference in the calculations. While HAF is a cumulative approach, SHAF is a diminishing one. Consequently SHAF is calculated as can be seen in ( 2.2), ( 2.3) and ( 2.4) while similar to (2.1), $C_s$ is the scene, in the form of a point cloud, $R$ is the definition of the regions, $\uparrow i_{R_{xx}}$, $\downarrow i_{R_{xx}}$, $\uparrow j_{R_{xx}}$ and $\downarrow j_{R_{xx}}$ are the starting and ending indices of $i$ and $j$ in the $xx$ region, where $xx$ can have a value from $\{rr, gg, bb\}$, that stand for the red region, green region and blue region respectively, $c_{ij}$ shows the highest value of the cell $R_{ij}$ and $w_{xx}$ is the weight of the corresponding region. Furthermore, $GDiff(C_s, R)$ represents the difference between the region value $RV_{gg}$ and the maximum of the $RV$s of the two other regions, $rr$ and $bb$ .

$$\begin{cases} SHAF(C_s, R) = \begin{cases} GDiff(C_s, R) & GDiff(C_s, R) > 0 \\ -1 & \text{otherwise} \end{cases} & (2.2) \\ GDiff(C_s, R) = RV_{gg} - max(RV_{rr}, RV_{bb}) & (2.3) \\ RV_{xx, x\in\{rr,gg,bb\}} = \sum_{i=\downarrow i_{R_{xx}}}^{\uparrow i_{R_{xx}}} \sum_{j=\downarrow j_{R_{xx}}}^{\uparrow j_{R_{xx}}} c_{ij}.w_{xx} & (2.4) \end{cases}$$

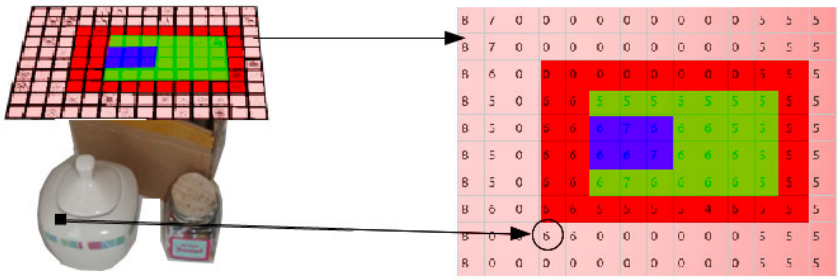


*Figure 2.1: Demonstration of a HAF as a mask on a Height Grid. This HAF comprises three different regions. Each region has its own weighting factor.*

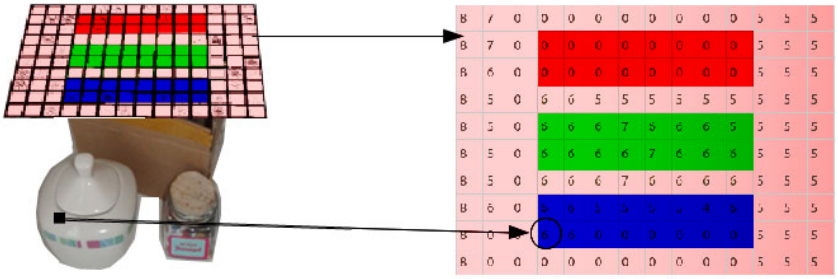


*Figure 2.2: Demonstration of a SHAF as a mask on a Height Grid, comprising three different regions. Each region has its own weighting factor.*

It should also be stated that, introducing SHAF in Fischinger et al. [9] as a complementary feature type to the HAF which was introduced and used earlier in Fischinger et al. [8] can be seen as a step toward optimizing TFRGA. In addition, not only it was tried to optimize the outcome of the system by defining the new feature type, but also Fischinger et al. [7] directly states that overall around $71,000$ different HAF features were created, most of which automatically using a brute force algorithm, and then subsets of size $300$ to $325$ were selected, measured by their f-score in order to increase the performance, whilst maintaining the quality which means having minimum impact on the result in the performance-quality trade off.

## 2.2 Previous Obtained Results

In Fischinger et al. [7] it is claimed that HAF brings significant additional information value. Conducting a test a comparison between a classifier trained with HAF data versus a classifier trained with simple Height Grids in the mentioned resource supports the result this claim. Table 2.1 shows the result of this test on a $3928-$record dataset. It can be seen that according to this result, HAF increased the success rate by over $21\%$. In a world in which people are struggling to increase the Accuracy of models by a tenth of a percent, a $21\%$ increase in Accuracy sounds like an outstanding improvement.

Nonetheless, the same statement could not be made about SHAF, according to the same article. Another test was performed measuring the improvement by adding SHAF features on the same HAF feature set. The outcome on the same dataset as the previous test seemed drastically changed again but this time by decreasing the Accuracy to $74.31\%$. Then the author of Fischinger et al. [7] in a try to explain the shocking results, gathered more instances for classifier data training and included more specific cases for which SHAF was proposed, into the training set. The final result after including such instances reached the point of $85.50\%$, still less than the result obtained from HAF only. However, it should be mentioned that according to further investigations by the authors of TFRGA, it turned out that the Accuracy of SHAF is not bal-

| Data Type | Succeeded cases | Success Rate (%) |
|---|---|---|
| Height Grid | 2516/3928 | 64.05 |
| HAF | 3368/3928 | 85.74 |

*Table 2.1: Comparison between HAF and Height Grid on the same dataset using Support Vector Machine classification algorithm. This result is extracted from Fischinger et al. [7] directly. HAF increased the success rate notably.*

anced on different classes. Having two classes of *Positive* and *Negative*, SHAF hit an Accuracy of $90\%$ on Negative class. Therefore it can be concluded that including SHAF, increased *recall* of Negative class and decreased *precision* of Positive class, again a trade-off between precision and recall.

According to Fischinger et al. [7], "Clearly, missing training data cannot be considered as a sufficient explanation why the HAF-classifier achieved a higher Accuracy rate than the classifier trained with HAF and SHAF"[1]. Thus, in following chapters where the outcome and the experimental results of this project are presented, probable causes of this incident when one deals with Support Vector Machine classifiers is also discussed.

[1] Quote from Fischinger et al. [7] Chapter 5 Section 3

CHAPTER 3

# Theoretical Contribution

In the previous section, is was presented that HAF and SHAF bring additional information value. In this chapter, three new feature set definitions, Circular Feature Definition (in short, CF), Differential Intra-Circular Feature Definition (in short, DICF) and Bell-Circular Feature Definition (in short, BCF) are introduced and the motivations behind using them are explained. Furthermore, a plan for the experimental phase is presented that includes pre-processing steps on the dataset and an overview of the selection process of the classification algorithms.

## 3.1 New Features

In order to expand the experiment to other possible types of features, here in this project, we introduce three new features definitions. These three features definitions, Circular Feature (CF), Differential Intra-Circular Feature (DICF) and Bell-Circular Feature (BCF) follow a very simple principle. They try to find a horizontal circular cylinder which is suitable to grasp. In comparison to HAF and SHAF, these three features are considerably lighter in terms of computational complexity, to be extracted and processed, due to the characteristics of Height Grids. However, more detailed experiments are planned to evaluate the adequacy of these new features. Later in this report, experimental results of the new features, which are extracted using these new feature definitions, are presented, along with a comparison between the results which are obtained with, and without adding these features to the system, that mainly work with HAF and SHAF.

### Circular Feature Definition - CF

Circular Feature, CF for short, is a type of feature that splits a height grid to three disjunctive regions using a simple circle, Inside (I), Outside(O) and the Border (B) as shown in Figure 3.1. Then based on a linear mathematical combination of these regions, it is tried to estimate whether a new scene has a suitable grasping point or not. CF circles are always centered at the center of the Height Grid, which is a the point where $c_{7,7}$, $c_{7,8}$, $c_{8,7}$ and $c_{8,8}$ meet while $c_{i,j}$ indicates the cell at the position $(i, j)$ of a $14 \times 14$ Height Grid. Consequently, only there are 6 circles

which could be defined in a Height Grid of size $14 \times 14$, and this is the reason why circle based features, which are defined in this project, can be considered lightweight in terms of computations. Finally, the Figure 3.2 shows the application of a CF on a Height Grid, whilst (3.1) depicts how the circular feature value of the Height Grid is calculated.

$$V_{cf}(hg, cf) = Avg(hg, cf, B) - Avg(hg, cf, I) \tag{3.1}$$

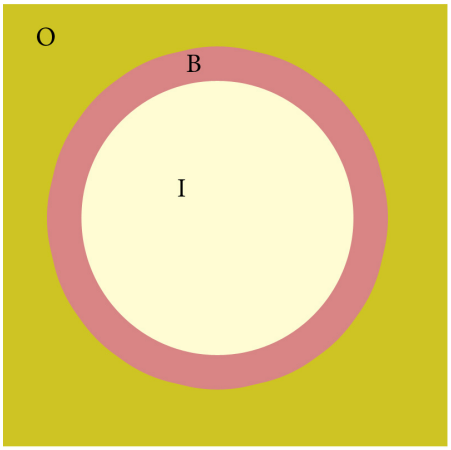



*Figure 3.1: Three regions of a Height Grid, separated by a circle. These three regions are called Inside (I), Outside (O) and the Border (B). Depending on the radius of the circle, the width of I and O regions can vary but the border is always 1 cell long.*

In (3.1), $V_{cf}(hg, cf)$ is the feature value of the Height Grid $hg$, having the feature definition $cf$. This value is calculated by subtracting the average height in the region $I$ from the average height in the region $B$. In other words, the supporting idea is that if in a cylinder, the border area is averagely lower than the inside area, then this cylinder can be a good candidate for grasping. However, the feature values from all 6 cylinders should be seen together, for a more accurate prediction.

### Differential Intra-Circular Feature Definition - DICF

Although Differential Intra-Circular Feature, DICF, follows the same principle as CF in using circles, there is a difference between the way in which the DICF is computed and the computation of CF. While CF concentrates on the average height of I and B regions, DICF tries to find the difference between the highest point and the lowest point in I region. As a result, the computation of DICF is as simple as computation of CF, and is shown in (3.2).

$$V_{dicf}(hg, dicf) = Max(hg, dicf, I) - Min(hg, dicf, I) \tag{3.2}$$

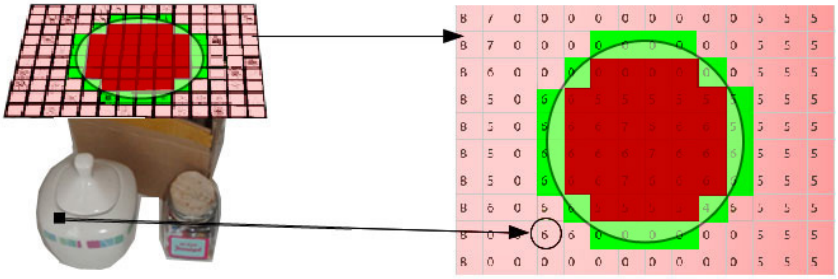


*Figure 3.2: A CF definition applied to a Height Grid. In this illustration, the green cells are border (B) cells and all other cells surrounded by these green cells are inside (I) cells. Subsequently, all the rest are outside (O) cells.*

In (3.2), $V_{dicf}(hg, dicf)$ is the DCIF feature value of the Height Grid $hg$ using the feature definition $dicf$. It was mentioned before that the value is the difference between the highest point in I, $Max(hg, dicf, I)$, and the lowest point in I, $Min(hg, dicf, I)$. The strength of CF and DICF is in their simplicity in terms of both definition and necessary computation steps. Figure 3.3 shows how a DICF is applied to a Height Grid.

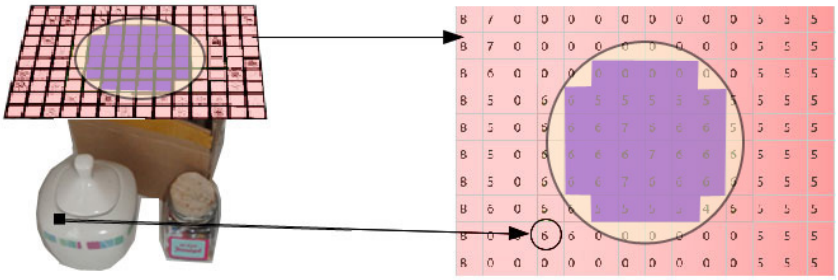


*Figure 3.3: A DICF definition applied to a Height Grid. In this figure, only violet region, I, is involved in the computations.*

### Bell-Circular Feature Definition - BCF

Assuming that, rather than a simple circle, a Height Grid is masked by a weighted circle, it will be possible to prioritize some cells and involve them in the result by their importance. It is important to mention that, for a gripper, it is essential to find two suitable points to place its fingers. That's why BCF concentrates on two sides of the circle which are the locations of the fingers. BCF forms an imaginary crown as illustrated in Figure 3.4 and tries to find the average

height of the cells which are underneath the circle of the crown according to their importance defined by the relative position of the cell to the center of the Height Grid. From a different point of view, a BCF calculation can be seen as measuring the similarity of a scene to a crown.

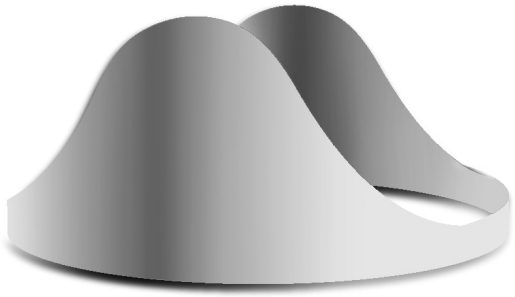


*Figure 3.4: BCF crown of cell importance. A visualization of the crown of the Bell-Circular Feature with two bells which specify the importance of each cell on the circle.*

Having the crown and the region B of the circle at the bottom of the crown, (3.3) and (3.4) are utilized to obtain the Bell-Circular Feature value. B in (3.3) refers to the B section of the Figure 3.1. Furthermore, $(x', y')$ indicates the location of the center of the cell $c_{i'j'}$ where $i'$ and $j'$ are the indices of the cells after transforming the Height Grid's coordinate system's base, to the center of the Height Grid, with y-axis in reverse order. Besides, Figure 3.5 visualizes how the BCF is applied to a Height Grid and defines a weight for each cell underneath its circle. In this figure, the lighter the green color goes, the more important that cell becomes, whilst $\alpha$ specifies the location of the cell and its importance.

$$V_{bcf}(hg, bcf) = \frac{\sum\limits_{c_{i'j'} \in B} |Cos(\alpha_{x'y'})| \times c_{i'j'}}{\sum\limits_{c_{i'j'} \in B} |Cos(\alpha_{x'y'})|} \tag{3.3}$$

$$\begin{cases} \alpha_{x'y'} = \frac{y'}{x'} \\ (x', y') = (i' + 0.5, j' - 0.5) \\ (i', j') = (i - 7, -j + 7) \end{cases} \tag{3.4}$$

**BCF as Convolution**

The process of calculating BCF can be seen as convolving the Height Grid with six different kernels. Each kernel, $k_i; (1 \leq i \leq 6)$, is a matrix with value 0 for all cells but the cells on a circle with $r = i$. Furthermore, in this case, the kernel is the same size as the original matrix. Thus no sliding is required to get the convolution result.

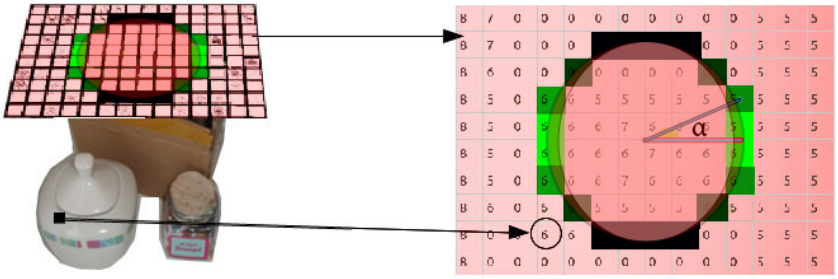



*Figure 3.5: A BCF definition applied on a Height Grid. The more lighter the green color, the more important the cell. $\alpha$ specifies the location of the cell and its importance.*

## 3.2 Learning Process

### Chunked Data sets

An important pre-processing step which was done in Fischinger et al. [7] has been producing new data, based on the binomial gathered samples of successful and failing grasping scenes, by manipulating the height of the objects in the scene. As there is no sufficient argument in the article to reject the hypothesis of impacting the final result by this pre-processing step, the newly generated data is suspected to bias the classifier's result model. In order to prevent such ambiguity in the newly gathered samples, it is decided to split the raw samples to samples with no artificial reproduction, in 5 disjoint subsets. Afterwards, it will be possible to generate new data which is believed to help the classifier to classify more precisely. However, this time, the data-generating phase is not intended to manipulate the height of the object in the scene but to flip the Height Grid, once horizontally and once vertically. Therefore as it can be seen in Figure 3.6, three new samples can be generated from one original sample which are believed to have no harm to the classifier. In addition, these newly generated data stay in the subset to which the original scene belongs.

Generally, the effectiveness of what has been done in Fischinger et al. [7] on the height of the samples, should be tested as there might be no difference for certain classifiers, particularly those that use a separator linear line to separate the data into binary classes. Thus, in Chapter 4 of this document, where experimental results are presented, this question will be answered.

### Pre-processing

The machine learning cases in which the data range of the attributes vary significantly, attribute values may need to be standardized, scaled or normalized prior to use for training a model, particularly as utilizing a machine learning algorithm whose objective function uses distances between different attribute values. Having a vast range of values, one attribute may dominate the others due to the larger difference between its values [13]. Thus, the range of all features

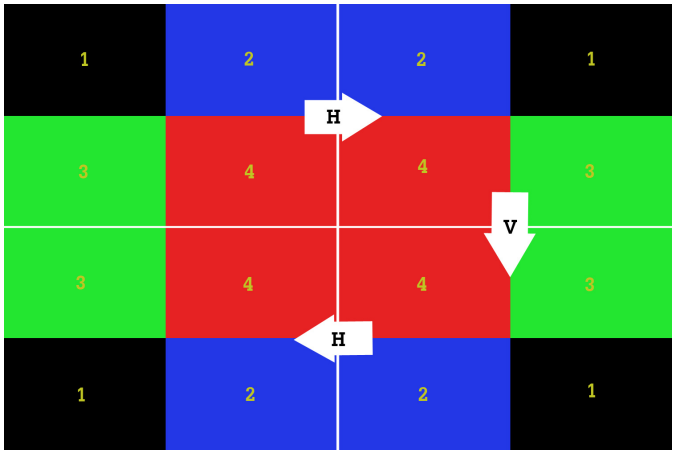



*Figure 3.6: The figure illustrates a sample assumed scene with four regions and its corresponding generated scenes. The one with 4 regions in the top left corner is the original scene,* H *and* V *characters on the arrows show that the scene is flipped horizontally or vertically respectively relative to its predecessor. These four colors as four regions are chosen in order to facilitate understanding of the process.*

should be normalized such that each attribute contributes approximately proportionately to the final decision.

In this project, normalization/scaling is so important since, on one hand, the value of different features can considerably vary and on the other hand, Support Vector Machines are sensitive to attributes' ranges [16]. Furthermore, it is very important to use the same normalization/scaling method on both training and test sets. In TFRGA, the authors used a linear method for scaling the attribute data which is commonly known as *Min-Max scaling* or simply *Rescaling*. In this method, as shown in (3.5), each attribute is scaled linearly to the range of $[0, 1]$ and although the scaling function is notably simple, it can result in outstandingly better outcomes especially when dealing with support vector machines [16]. What's more, $S(a)$ in (3.5) indicates the scaled value of the attribute value $a$.

$$S(a) = \frac{a - \min(a)}{\max(a) - \min(a)} \tag{3.5}$$

Since rescaling is used in TFRGA and it performed pretty well, therefore it is decided to use the same approach. The rescaling functionality which is used in this project is provided by a `MinMaxScaler` class object of SciKit Learn.

### Classifier Selection

In order to have more information about what classification method can perform better on such a problem, some well-known and widely used classification algorithms are employed. These algorithms, then, will be assessed, based on the quality of the results, using some measurements

such as Precision, Recall, F1 score, Accuracy, and the area under their Receiver Operating Characteristic curves.

### Parameters

For the parameters, in each experiment part of this project on different classifier algorithm, the amounts which are commonly believed to be more effective on general problems are set. In each section of the Experimental Result's section, the parameters which are used by the classifier algorithm to build models are specified.

CHAPTER 4

# Experiments and Results

In Section 1.2, it was mentioned that, in order to be able to conduct different experiments, a testing software is required to be developed. In this chapter, first of all, the testing application which has been developed for the experiments of this project is introduced, and more information on different aspects and features of the application, along with prerequisites to run it are provided. After that, it is described that what steps should be taken in order to prepare a test rig, extend sample dataset, and prepare requirements for performing various tests.

## 4.1 Set-up

In the path to optimize the solution, and in order to put the theories in use and test them, a test rig, along with a set of applications are required to be set up and implemented respectively. Here in this chapter, all of the requirements for a test environment are discussed more in depth, followed by the experimental results of the project, having different settings of datasets, features, algorithms and parameters. At the end of this chapter, the obtained results from different algorithms are analysed in order to give a deeper understanding of the characteristics of our problem.

### Test Application

The first step in an optimisation problem is to have a measurement tool. As it is mentioned before, an application is developed in order to help see the differences in the outcome of any individual change that is made to each part of the process. This application should be flexible enough to enable putting any affecting parameter in test. Thus as designing, it is tried to make the application architecture as flexible as possible to the changes which may come in mind later.

What's more, there are some important points which should be taken into consideration about the Test Application. The first point to consider is that unlike the operator application which is installed on the robotic hand, that we call Production Application, Test Application does not need to be real time. In contrast, the Test Application should be more flexible to possible changes than the Production Application. Therefore, some features of the production

application that are introduced in Fischinger et al. [7] which have no impact on the result and just implemented to improve the performance such as "height accumulated rectangular regions" are skipped in the Test Application. Besides, while the Production Application is implemented in `C++`, the Test Application is implemented in `Python`, that is known to be significantly slower. Another point is that, in contrast to the Production Application, the Test Application is highly customizable in terms of underlying algorithms and parameters. Nonetheless, it should be the case about every Test Application to provide testers with a wider range of capabilities to customize the behavior and observe the differences.

### Architecture

As it has been stated before, the Test Application must keep the flexibility meaning that testing new theories by changing any part of the application should be fairly easy. On the other hand, the application should be capable of working with some similarly-structured components with different behaviors, such as different classifier algorithms. In order to maintain these two important points, the core of the Test Application is designed like a state machine. As it is obvious there are many advantages in this form of design. The first benefit is that the flexibility requirement is satisfied by a very simple and loosely-coupled architecture of a state machine. Figure 4.1 represents the architecture of the core of the application as a state machine while Table 4.1 describes each state as well as the task which should be done before leaving the state. The second important port for which we are looking, is the possibility of using similarly-structured components as substitutions. The state machine architecture provides a satisfying level of abstraction and isolation. Therefore it will be very easy to unplug a part of the state machine and replace it by another similarly-structured part.

As it is demonstrated in the Figure 4.1, there are different zones in the Test Application which are distinguished by different colors. For instance, green and red zones show the stages that positive and respectively negative samples of grasping are loaded by the application and the Height Grids are extracted from them. The reason why there are 4 states instead of one for each of these zones is definitely the memory problem. For large datasets, it is not simply possible to read all files in one move and keep them in memory. However, Height Grids are much lighter in terms of size on memory. Another reason is to isolate the transformation of the point could to Height Grid from other stages of the procedure.

Another interesting fact about the Test Application is that the output of each zone is saved on the disk. Consequently, it is easily possible to start off from a saved stage in future runs. This helps a lot to save time in testings, since usually reading, parsing and transforming point clouds to Height Grids take plenty of time. This way, if the method for turning point clouds to Height Grids is not subjected to change, the tester could simply start from generated Height Grids by previous runs. The same story applies to Height Grids to feature values and feature values to trained classifier. As it can be seen in the Figure 4.1, there are four different application entry points, distinguished by *PCD*, *HG*, *F* and *T* that stand for *Point cloud data*, *Height Grids*, *Feature values* and *Trained classifier* respectively.

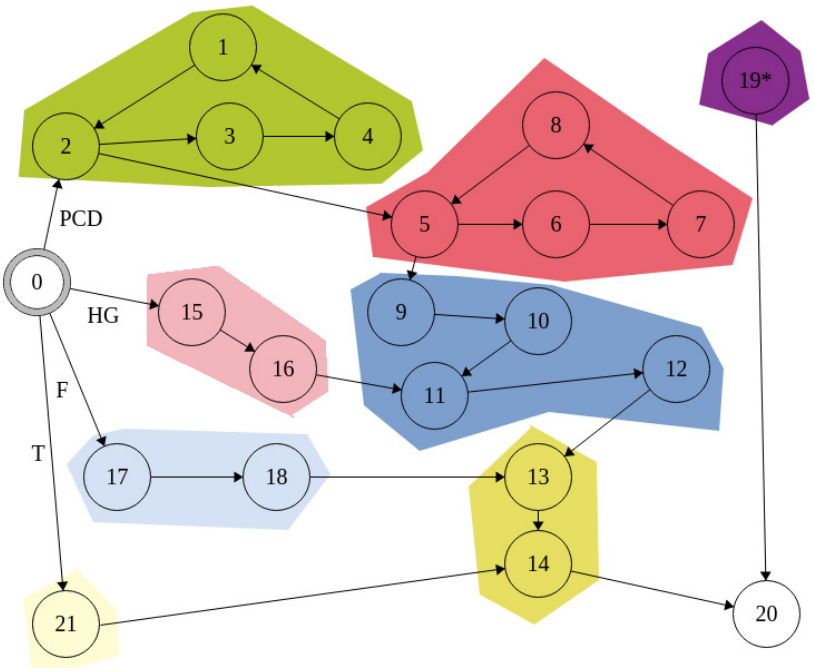



*Figure 4.1: Architecture of the Test Application as a state machine. The* green *zone shows the states in which PCD files for positive instances are processed. The* red *zone is where PCD files for negative instances are read and processed. Next, the* blue *zone shows the states in which feature definitions are read from files and Height Grids from green and red zones become feature values with respect to the feature definitions. After that, a classifier is trained with the feature values and tested in the* yellow *zone before going to the final state where the results are shown. 19* state is the error state where there is an error in any state. Therefore there should be an arrow from each state to 19*. However, in order to maintain simplicity, these arrows are not depicted. Three regions with lighter colors, light red, light blue and light yellow, are some states that do exactly what zones with their corresponding intense colors do having only one difference. They read the requirements from files which were saved before, in each former zone.*

### User Interface

In addition to the core, a user-friendly user interface is developed for the Test Application to make it as easy as possible to deal with different settings and arrangements of data, algorithms, and parameters. Figure 4.2 shows the main panel of the Test Application. As it can be seen in the figure, there are different modes to select for testing. Mode $1..4$ show different application entries which were discussed in the previous section. By selecting each mode of testing, the tester should provide the application with different input parameters. Furthermore, for modes $1$ to $3$, it is possible to customize the classifier's parameters. Figure 4.3 shows how the panel for customizing the parameters of an SVM classifier looks like. It should be noted that these parameters are different than the parameters shown for other classifiers such as $k-NN$ or *Random*

| Current State No. | Current State Job | Next State No. |
|---|---|---|
| 0 ((s)) | StateMachine Start | 1 |
| 1 | Positive PCD Read | 2,5 |
| 2 | Positive PCD Parsed | 3 |
| 3 | Positive Height Grid Created | 4 |
| 4 | Positive Height Grid Saved | 1 |
| 5 | Negative PCD Read | 6,9 |
| 6 | Negative PCD Parsed | 7 |
| 7 | Negative Height Grid Created | 8 |
| 8 | Negative Height Grid Saved | 5 |
| 9 | Feature Definition Read | 10 |
| 10 | Feature Definition Parsed | 11 |
| 11 | Feature Values Created (ALL) | 12 |
| 12 | Feature Values Saved (Disk) | 13 |
| 13 | Learner Trained | 14 |
| 14 | Learner Tested+Saved (Disk) | 20 |
| 15 | Saved Height Grid Read (ALL) | 16 |
| 16 | Height Grid Parsed | 11 |
| 17 | Saved Feature instances Read (ALL) | 18 |
| 18 | Feature instances Parsed | 13 |
| 19 | Error | 20 |
| 20 ((t)) | FINISH | (0) |
| 21 | Saved classifier Read | 14 |

*Table 4.1: The table represents the states in the state machine depicted in Figure 4.1.* Current State Job *shows the job which is done before checking the condition of transition.* All *means both positive and negative instances and* Disk *means while leaving that state, the data is saved.*

*Forest*.

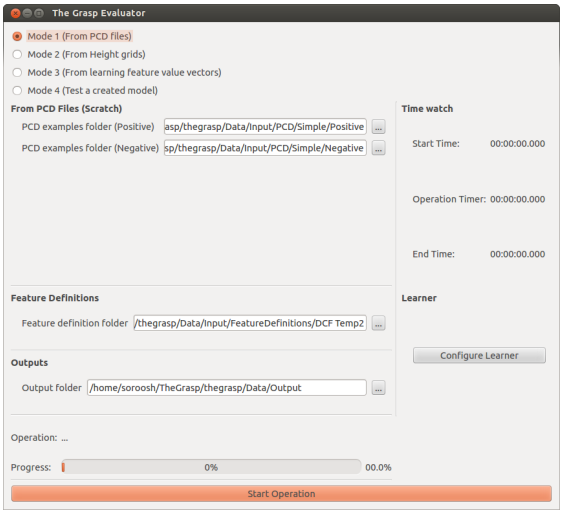



*Figure 4.2: The main panel of the Test Application. There are different parameters and inputs to set, such as the path to the input data and the desired path to write the output date including intermediary result of each zone.*

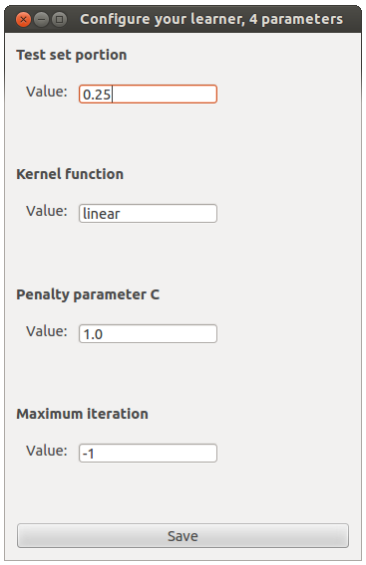



*Figure 4.3: The panel to customize the parameters for the classifier. Here in this figure, the parameters for an SVM classifier are shown since the selected classifier for the application is an SVM.*

However, for the mode 4 it is not possible to customize the parameters anymore since the mode 4 simply loads a classifier which is trained before and a data set as a test set. These saved classifier models are pickled classifier objects on disk from previous runs. Figure 4.4 illustrates the panel for loading a saved classifier and a test dataset.

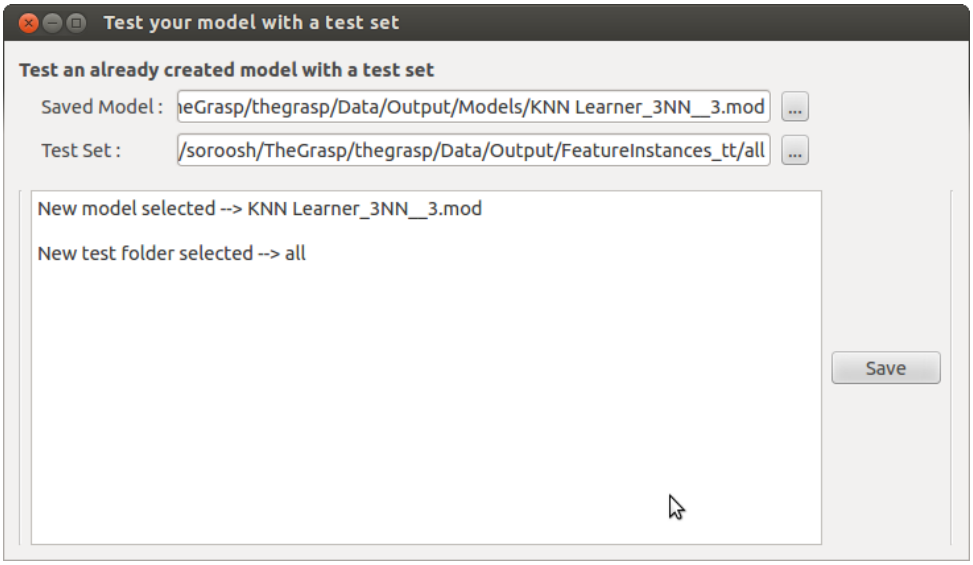



*Figure 4.4: The panel to Load a pickled classifier and a test set from disk. Immediately after selecting these two requirements, it would be possible to test the classifier on the dataset.*

Finally, after the *Start Operation* button is pushed and the procedure is done, the result is shown in a proper informative way. Figure 4.5 shows how the output is represented. There are different types of information projected on this board. Starting from the top left corner, it is stated that what classifier is used for this experiment. Then, the size of the dataset which is loaded, the proportion of the data used to split the dataset and the final number of instances which are used for training and testing are shown. Furthermore, on the right-hand side of the panel, the detailed information about the classifier is shown, starting with the final result of the experiment in terms of classification precision, recall and f-score. Then a log-like data is presented containing the information mentioned above in addition to a confusion matrix of the classifier and an overview of the parameters used to train the classifier. Nevertheless, when Cross-Validation is used for testing the classifier, the confusion matrix is not shown anymore and instead, Receiver Operating Characteristic, ROC is generated to reflect the characteristics of each classifier on the domain.

### How To Run

There are some prerequisites which should be met before one can simply run the application. These prerequisites are mainly some `Python` packages and dependencies. Meeting these prerequisites makes it possible to run the application regardless of what operating system is the host since the `Python` and all dependent packages can work cross-platform, and so the application. The prerequisites are listed below:

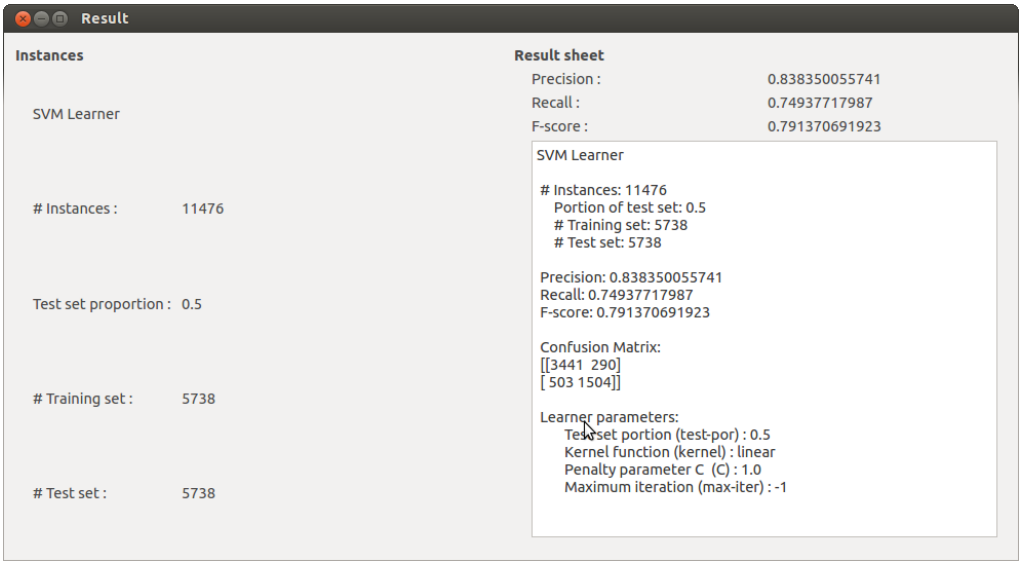



*Figure 4.5: The board to show the result of the procedure. There are different types of information shown on this board such as the information about the input, the classifier type, the parameters of the classifier and the output.*

- ⋆ **Python:** Python [38] version 2.7.3, an "easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming [37]".
- ⋆ **Numpy:** Numpy [24] version 1.6.1, a "fundamental package for scientific computing with Python. NumPy can also be used as an efficient multi-dimensional container of generic data [23]".
- ⋆ **Qt:** Qt [22] version 4, a "cross-platform application framework that is widely used for developing application software that can be run on various software and hardware platforms with little or no change in the underlying codebase, while having the power and speed of native applications [31]".
- ⋆ **PyQt4:** PyQt4 [4] version 4.9.1, a "set of Python $v2$ and $v3$ bindings for The Qt Company's Qt application framework and runs on all platforms supported by Qt including Windows, MacOS/X and Linux [30]".
- ⋆ **Scikit-learn** Scikit-learn [28] version 0.16.1 a "Simple and efficient tools for data mining and data analysis [36]" which is developed for Python. Most of machine learning tools such as classifier algorithms and pre-processing tools which are used in this project are provided by Scikit-learn package.

Having the prerequisites fulfilled, it is fairly easy to run the application. However, before running the application, the tester should decide which classifier algorithm needs to be plugged into the application. In order to test different types of the algorithms on the datasets of the prob-

lem domain, some well-known algorithms are selected to be implement and utilized. However, one can add a new classifier to the application with a little amount of effort. The classifier, as mentioned before, should be specified through a configuration file, that is placed inside the project. The configuration file which is named *learners.txt* can be found in the path "*src/Starter/UI/*" relative to the root of the project. Appendix A.1 shows how to select the desired classifier by putting a * next to its name in the file. Adding a new classifier to the list of the classifier is quite easy. There are only two different things to declare. At first, parameters should be defined, if any is required to be configured through the user interface. Appendix A.2 demonstrates how to define the parameters that are required to be possible to change in the application. Finally, the classifier algorithm should be specified and the defined parameters can be used to declare the new classifier as shown in Appendix A.3. It can be seen that the parameters which are defined in Appendix A.2 are the same parameters as the shown in Figure 4.3 with one addition, the proportion of the test set, which is in common by all classifier algorithms. In the cases where Cross-Validation is used, the number of folds is equal to $\lfloor 1/\texttt{proportion} \rfloor$.

## Extending Sample Database

In almost every case in machine learning in which a model should be created, it is commonly known that the more data gathered for training, the more accurate the model will become. However, there are some considerations in extending data samples. In this section, it is explained that how some new samples are gathered for this project.

The newly gathered data as before and in order to maintain consistency to the data that existed before and gathered in Fischinger et al. [7], is collected in a binomial form of either successful or failure cases of grasping. However, it should be mentioned that during this process, no robotic manipulator is used and only the possible grasping dimensions of an object, located in the center of the scene are checked. In more details, provided that in a $14cm \times 14cm$ scene, an object located in the center of the scene, having maximum width of $10cm$ and sufficient depth for grasping, meaning at least $4cm$ in our case, then the captured scene is categorized as *successful* grasping case and all other cases are considered as *failure* grasping cases.

### Prerequisites

In order to gather new data samples, a test rig is built and some other requirements needed to be met. In following, the most important parts of this process are described consisting of the test rig, the process of gathering, transforming and reproducing the data.

### Test Rig

The first step toward having more samples is to have a proper test rig and a way of collecting the information of a particular scene in this test rig in the form of a point cloud. For this purpose, a test rig is created and a Microsoft Kinect [41] is utilized to capture the necessary information of the scene. Figure 4.6 shows this created test rig from two different views, one from outside and the other, closer view. As it is illustrated in Figure 4.6a, a Microsoft Kinect is mounted on the wall, monitoring the area that is in front of it, meaning the part of the table with some object in

*(a) An outer look of the test rig* *(b) A close view of the test rig*

*Figure 4.6: Two different views of the test rig. The left side figure illustrates the test rig and the Microsoft Kinect device, mounted to collect the information from the scene. The right side figure shows the* $14cm \times 14cm$ *square of the scene.*

different shapes. Figure 4.6b magnifies the important part of what Microsoft Kinect observes, which is of our interest. As it can be seen in this figure, a square of size $14cm \times 14cm$ is drawn in which three different objects with diverse dimensions are placed. This square is considered as our scene for which point cloud files will be created in later stages.

### Gathering

In order to obtain desired data from Microsoft Kinect, ROS[1] [32], a robotic platform along with some packages and programs are employed and some extensions are implemented. In following, a list of what is required in order to obtain the target point cloud files is stated:

- ⋆ **ROS** Hydro Medusa, a robot operating system that "provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more. [6]".
- ⋆ **Rviz**, a 3D visualization tool for ROS [35].
- ⋆ **Openni_camera**, "A ROS driver for OpenNI depth (+ RGB) cameras. These include: Microsoft Kinect, PrimeSense PSDK, ASUS Xtion Pro and Pro Live The driver publishes raw depth, RGB, and IR image streams. [25]".
- ⋆ **Freenect_camera**, "A libfreenect-based ROS driver for the Microsoft Kinect. [11]"
- ⋆ **PCL**, Point Cloud Library, a ROS package "for point cloud processing - development. The PCL framework contains numerous state-of-the art algorithms including filtering, feature estimation, surface reconstruction, registration, model fitting and segmentation. [27]".

[1] Robot Operating System

*(a) Test rig, RGB + depth view* *(b) Test rig, depth view*

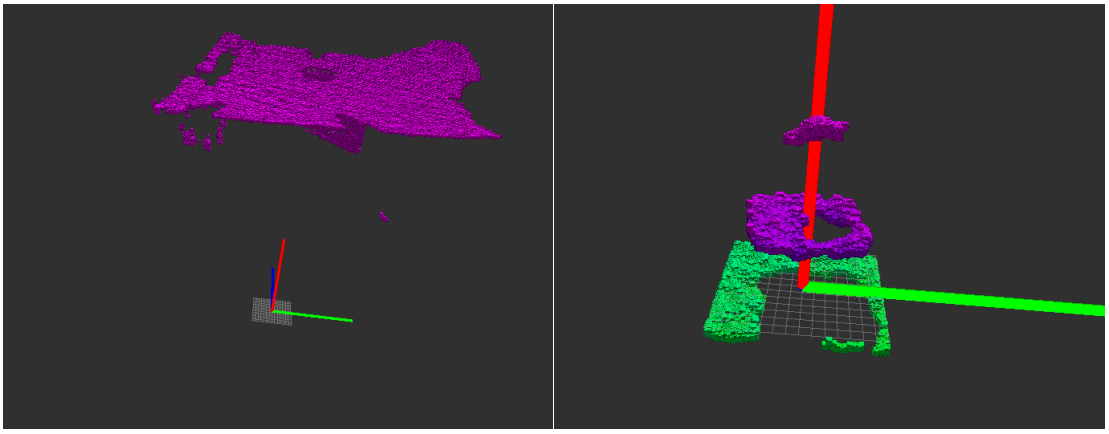
*Figure 4.7: Two different depth views of the test rig. The left side figure illustrates the test rig in RGB form and a depth view of the square test rig. The right side figure shows the test rig in depth view and the square test rig of the scene in a different depth view layer.*

After having these application installed, it would be possible to obtain depth information in the form of point clouds from ROS and PCL library. Rviz, as mentioned, is a visualization tool by which it is possible to visualize the point cloud information from Microsoft Kinect. Figure 4.7 shows the view of the Microsoft Kinect, visualized in Rviz. The figure shows two different windows. The first window on the left side shows the RGB output of the camera along with a depth view layer over this RGB output, only for the $14cm \times 14cm$ square test rig. The one on the right-hand side, which is demonstrated in Figure 4.7b, shows two layers of the depth view of the Microsoft Kinect camera, one the whole camera view and the other one the square test rig.

Subsequently, the depth output information of the camera can be used as input for PCL package, turning the information to point clouds. Fortunately, the resulting point cloud can also be visualized in Rviz.

### Transformation

A transformation should be done in order to transfer the center of the coordination to the center of the point cloud as the depth view which is obtained from Kinect is relative to the position of the Kinect's depth camera. In other words, the center of the coordination of the obtained point cloud is the Kinect's depth camera and all points should be transformed to a new Cartesian coordination system where the X$-$Y plain matches the surface of the test rig. In order to transform the point cloud, the transformation matrix (4.1) is used. The transformation matrix (4.1) transforms a point $(x, y, z)$ to a corresponding point in the new coordination using $d_x$, $d_y$ and $d_z$ which stand for the distance of the center of the Kinect's depth camera from the center of the



*(a) Raw depth input* *(b) Transformed and hedged*

*Figure 4.8: Two different views from the same scene with and without coordination transformation. In addition to coordination transformation, (b) is a $14cm \times 14cm$ cut of the scene.*

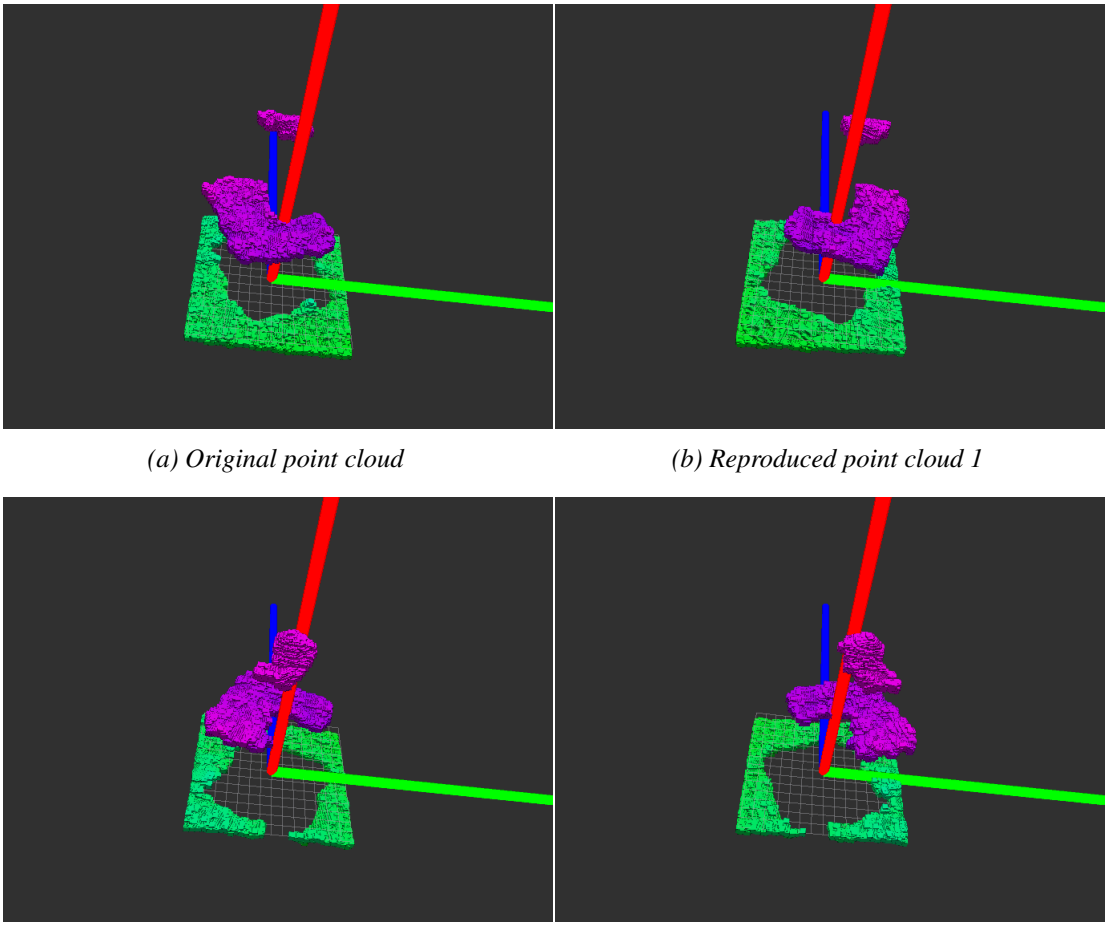
test rig along with $p$, $r$ and $y$ which stand for pitch, roll and yaw respectively. Furthermore, $S_a$ and $C_a$ in (4.1) show $Sin(a)$ and $Cos(a)$.

$$\begin{pmatrix} C_yC_p & C_yS_pS_r - S_yC_r & C_yS_pC_r + S_yS_r \\ S_yC_p & S_yS_pS_r + C_yC_r & S_yS_pC_r - C_yS_r \\ -S_p & C_pS_r & C_pC_r \end{pmatrix} \begin{pmatrix} x - d_x \\ y - d_y \\ z - d_z \end{pmatrix} \tag{4.1}$$

Figure 4.8 illustrates how the coordination transformation algorithm works. Figure 4.8a shows the raw input, that comes from the depth camera of Kinect whereas Figure 4.8b represents a transformed $14cm \times 14cm$ cut of the same scene. The transformation algorithm works as a ROS service, that receives the depth information from Kinect and publishes the transformed scene for further use.

### Reproduction

In Section 3.2, it was mentioned that in order to have more point cloud instances to be used in the process of training a model, three more point clouds are produced, out of each point cloud file that was obtained directly from Kinect and transformed, resulting in four times point cloud instances. Figure 4.9 demonstrates a point cloud and its three additional reproduced versions according to the description in Section 3.2.



*(a) Original point cloud* *(b) Reproduced point cloud 1*

*(c) Reproduced point cloud 2* *(d) Reproduced point cloud 3*

*Figure 4.9: An original point cloud, that comes from a real scene along with three different reproduced point clouds based on the original one.*

### Hardware Specifications

For all parts of the experiments that are conducted in this project, a single computer with the specifications mentioned in Table 4.2 is utilized. Thus, all represented running times of different algorithms or different runs are measured on the stated hardware system and may differ if tested on different specifications.

| Component | Details |
|---|---|
| Processor | Intel Core2 Duo CPU E8400, 3.00GHz $\times 2$ |
| Memory | 6GB DDRII |
| VGA | AMD ATI Radeon HD 4350 |
| Operating System | Ubuntu 12.04LTS x64 |

*Table 4.2: Hardware specifications on which the experiments of this project are conducted.*

## 4.2 Experiments Overview

In following sections, the experimental results of the project are presented which are obtained using different algorithms and parameters. In each section, it is mentioned that which part of the code/algorithm is different than the others along with the outcome. At the end, a comparison is made to magnify the differences between the approaches and a conclusion is made out of the result of the comparison.

For the experimental part of the project, an independent set of instances is reserved as the test set. This set is collected for this project and is reproduced using the method mentioned in Section 3.2. The set contains 2240 instances in total and all experimental results, presented in this chapter will be the result of the trained models on the mentioned test set.

## 4.3 Parameters

The set of the parameters, that are used to configure the classification algorithms that are utilized in this part of the experiments are shown in Table 4.3. Where necessary, the parameters are explained and the reasons behind selecting them are discussed in more details in their corresponding sections. It is noteworthy that for each classification algorithm, that is utilized in the experimental part of the project, there are more parameters to adjust. However, the most important ones, with the most impact on the results of the algorithms are picked to be shown in Table 4.3, and the rest of the parameters are set according to the default values of SciKit libraries.

| Algorithm | Parameter | Value |
|---|---|---|
| SVM−Lin | Kernel | Linear |
| | Penalty (C) | 1.0 |
| | Probability Estimation | True |
| SVM−RBF | Kernel | Radial Basis Function (RBF) |
| | Penalty (C) | 1.0 |
| | $\gamma$ | $\frac{1}{\#\text{Instances}}$ |
| | Probability Estimation | True |
| SVM−SGD | Penalty Parameter | l2 |
| | Loss function | modified_huber |
| | $\alpha$ | 0.0001 |
| | $\epsilon$ | 0.1 |
| 3−NN | # Neighbors | 3 |
| | Weights | Uniform |
| | Distance Algorithm | Auto (ball_tree, kd_tree, brute) |
| Random Forest | # Trees | 10 |
| | Criterion | Impurity (Gini) |
| | Max Features | Auto |
| | Min Leaf Size | 1 |

*Table 4.3: Selected parameters for the different classification algorithms that are utilized in the experiments.*

## 4.4 SVM – Linear Kernel

For the fist part of the experiments, it is decided to use SVM, as it has been also used in Fischinger et al. [7]. In this part of the experiments, an SVM is trained over the set of instances that was gathered in Fischinger et al. [7], along with the newly collected instances of this project, to show the result of the SVM classifier on the data.

In order to keep the experiment as close as possible to the experiments which were conducted in Fischinger et al. [7], the same set of parameters is selected for the algorithm, that has been

mentioned in Table 4.3 of Section 4.3 under *SVM−Lin*.

### Including and Excluding Reproduced Instances

The experiment is done in two different stages. In the first stage, the results of the trained models over the original dataset and reproduced dataset of Fischinger et al. [7] are presented, in order to test the effectiveness of the generated instances by altering the height of the objects in the scenes, as an answer to the question which was raised in Section 3.2. Table 4.4 shows the outcome of the experiment on the original set of instances along with the results which are obtained from the dataset having the reproduced data. In all tables that appear in this section, hereafter, the columns DS, FD, #At, CLS, P, R, F1, A, BT, and CM stand for Data Set, Feature Definitions, Number of Attributes, Classifier, Precision, Recall, F1 Score, Accuracy, Build Time and Confusion Matrix respectively. Furthermore, O and R under the column DS indicate the Original dataset and Reproduced dataset. Similar to Fischinger et al. [7], the same $302$ features for HAF and $21$ features for SHAF are used for the experiments.

According to the information which is provided in Table 4.4, in the case of HAF features, there is an improvement on the Precision, unlike the Recall which decreased from $0.848$ to $0.780$. However, the F1 score, which introduces a way of measurement based on both Precision and Recall, has a significant improvement which has jumped above $0.72$ from below $0.65$. Besides, the Accuracy improved remarkably, from $0.664$ to $0.791$. Nevertheless, such an improvement can not be seen in HAF−SHAF cases. According to the table, apart from the Precision, all factors dropped slightly.

In addition, Figure 4.10 illustrates the Receiver Operating Characteristic curve, ROC curve, of all four trained models. The area under the ROC curve, AUROC, measures discrimination [15] and it is used as another mean of measuring how good a classifier is, compared to other classifiers. The more the AUROC is, the better the binary classifier operates. As it can be seen in Figure 4.10, despite all differences, AUROC of the trained models with original dataset and the reproduced dataset in both cases of HAF and HAF−SHAF, are not notably different.

As a result, it can be concluded that, altering the heights of the objects in the scenes do not have any considerable impact on the results of the SVM classification models although it increased the required time for building the model drastically. However, ignoring the time overhead in building the models, the generated data seems to be harmless to the trained models and in both cases of HAF and HAF−SHAF slightly improved the results, according to the AUROC in Figure 4.10. Therefore, it is decided to use the reproduced dataset in addition to the newly collected instances for the rest of the experiments, in this project.

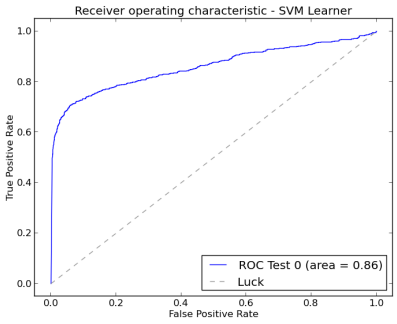
Picking the reproduced dataset as the base and adding the newly collected samples along with their mirrors, which were explained in former chapters of this document, the same experiment is conducted as the second stage of the experiments, on the new dataset, which will be the dataset for all later experiments. The results of the training over this dataset, using HAF and HAF−SHAF feature definitions are shown in Table 4.5. As it is shown in the table, and Figure 4.11, which depicts the ROC curves of the trained models, the results are slightly better than the best results from the previous stage.

| DS | FD | #At | CLS | P | R | F1 | A | BT | CM |
|---|---|---|---|---|---|---|---|---|---|
| O | HAF | 302 | SVM | 0.521 | 0.848 | 0.646 | 0.668 | 0:16 | $\begin{bmatrix} 818 & 662 \\ 121 & 679 \end{bmatrix}$ |
| O | HAF-SHAF | 323 | SVM | 0.748 | 0.725 | 0.736 | 0.814 | 0:17 | $\begin{bmatrix} 1245 & 195 \\ 220 & 580 \end{bmatrix}$ |
| R | HAF | 302 | SVM | 0.681 | 0.780 | 0.727 | 0.791 | 2:17 | $\begin{bmatrix} 1149 & 291 \\ 176 & 624 \end{bmatrix}$ |
| R | HAF-SHAF | 323 | SVM | 0.753 | 0.693 | 0.722 | 0.809 | 2:26 | $\begin{bmatrix} 1258 & 182 \\ 245 & 555 \end{bmatrix}$ |

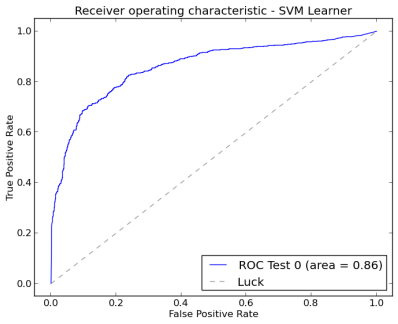
*Table 4.4: The results of the SVM algorithm on the datasets including and excluding the reproduced samples with different feature definitions. O shows the original dataset whilst R shows the instances including the reproduced data.*

| FD | #At | CLS | P | R | F1 | A | BT | CM |
|---|---|---|---|---|---|---|---|---|
| HAF | 302 | SVM | 0.796 | 0.721 | 0.757 | 0.834 | 15:14 | $\begin{bmatrix} 1293 & 147 \\ 223 & 577 \end{bmatrix}$ |
| HAF-SHAF | 323 | SVM | 0.759 | 0.717 | 0.737 | 0.817 | 16:07 | $\begin{bmatrix} 1258 & 182 \\ 226 & 574 \end{bmatrix}$ |

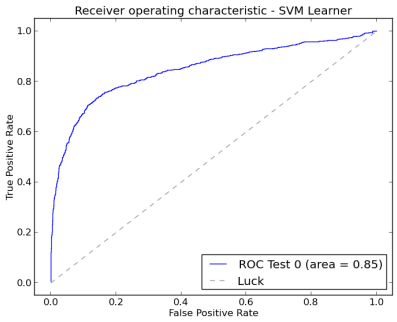
*Table 4.5: The results of the SVM algorithm on the final dataset with different feature definitions.*



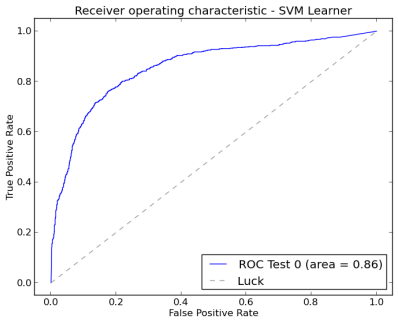

*(a) HAF (Original)*




*(b) HAF (Reproduced)*

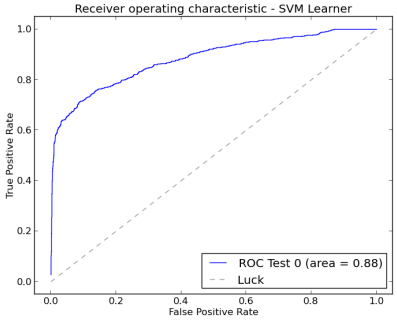



*(c) HAF-SHAF (Original)*

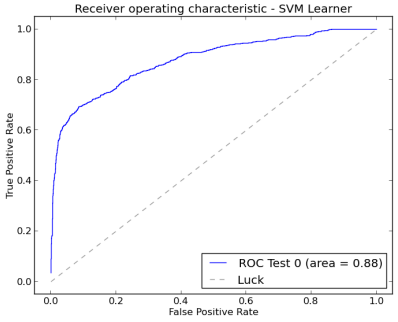



*(d) HAF-SHAF (Reproduced)*

*Figure 4.10: ROC curves of the trained models over original and reproduced data sets with two different feature definitions, HAF and HAF−SHAF*




*(a) HAF*




*(b) HAF-SHAF*

*Figure 4.11: ROC curves of the trained models over the final data set, comprising the reproduced data set from Fischinger et al. [7] and the newly collected instances, with two different feature definitions, HAF and HAF−SHAF*

### Newly Defined Features

According to what has been explained in Section 3.1 of this document, three new feature sets are proposed in order to improve the results of classification, CF, DICF, and BCF. The results of the trained models with each set of the features are demonstrated in Table 4.6. Based on the data that can be seen in the table, comparing to HAF and HAF−SHAF, these three newly defined features show acceptable outcomes with a better Accuracy from DICF as an example, that is above $0.85$, in comparison with $0.834$ from HAF. In terms of F1 score and Accuracy, the other two, CF and BCF obtained weaker results. However, it should be noted that these two features contain only 7 attributes each, which is only about $2\%$ of the size of the HAF and HAF-SHAF. Such a reduction in the size of attributes can have a dramatic impact on the required amount of time to build a model for the algorithms which are sensitive to the size or dimensions of the data, such as SVM. As a wittiness, the required time for building a model with HAF−SHAF is recorded at $16:07$ minutes, compared to $3:27$ minutes for a model with CF and $3:00$ minutes for a model with BCF.

| **FD** | **#At** | **CLS** | **P** | **R** | **F1** | **A** | **BT** | **CM** |
|---|---|---|---|---|---|---|---|---|
| HAF | 302 | SVM | 0.796 | 0.721 | 0.757 | 0.834 | 15:14 | $\begin{bmatrix} 1293 & 147 \\ 223 & 577 \end{bmatrix}$ |
| HAF-SHAF | 323 | SVM | 0.759 | 0.717 | 0.737 | 0.817 | 16:07 | $\begin{bmatrix} 1258 & 182 \\ 226 & 574 \end{bmatrix}$ |
| CF | 7 | SVM | 0.753 | 0.720 | 0.736 | 0.816 | 3:27 | $\begin{bmatrix} 1252 & 188 \\ 224 & 576 \end{bmatrix}$ |
| DICF | 6 | SVM | 0.939 | 0.625 | 0.750 | 0.851 | 4:29 | $\begin{bmatrix} 1408 & 32 \\ 300 & 500 \end{bmatrix}$ |
| BCF | 7 | SVM | 0.710 | 0.625 | 0.664 | 0.775 | <u>3:00</u> | $\begin{bmatrix} 1236 & 204 \\ 300 & 500 \end{bmatrix}$ |
| All | 343 | SVM | <u>0.978</u> | <u>0.753</u> | <u>0.851</u> | <u>0.906</u> | 18:38 | $\begin{bmatrix} 1427 & 13 \\ 197 & 603 \end{bmatrix}$ |

*Table 4.6: The results of the SVM algorithm on the final dataset with different feature definitions. The best scores are underlined.*

Interestingly, the combination of all five sets of feature definitions, which is called "All" in Table 4.6, hits a record of $0.851$ for F1 score and $0.906$ for Accuracy. To be more precise, "All"

feature definitions improved the results, obtained from HAF−SHAF by around $12\%$ in F1 score and $9\%$ in Accuracy. Nonetheless, it should be considered that the time for building a model is the highest among all in this case, with 18:36 minutes.

Furthermore, Appendix A.1 represents the ROC curves of the models of the Table 4.6. The inference can be drawn from the figure in the appendix that the model with "All" feature definition has the best AUROC among all others. Based on AUROC, HAF and HAF−SHAF follow "All", both with $0.88$.

## 4.5 More Classifiers

In order to find the best classification algorithm for the problem, more algorithms are utilized and tested on the final dataset, with all $6$ sets of feature definitions, that were used in the previous section. In this section, the experimental results of applying the selected algorithms, Radial Basis Function Kernel SVM, SVM with Stochastic Gradient Descent Training, $k-$Nearest Neighbors, Decision Trees and Random Forest are presented and compared to the results which were obtained from an SVM having a Linear Kernel.

### SVM − Radial Basis Function Kernel

Support vector machines accept different kernels, one of which "Radial Basis Function", RBF, which follows a simple principle, meaning that the influence of a training example depends on its distance from the hedging line. Here in this experiment, two important RBF parameters, $C$, the penalty parameter that trades off misclassification of training examples against simplicity of the decision surface, and $\gamma$, that in simple words, defines how far the influence of a single training example reaches [12] are adjusted according to the Table 4.3 of Section 4.3.

Despite spending more time for building the models with RBF Kernel, the data of Table 4.7 shows that the achievements are not very promising. Compared to the SVM with Linear Kernel, the best result with RBF Kernel still stands below the best scores from SVM with Linear Kernel by almost $4\%$ difference in Accuracy and $6\%$ in F1 score. However, it should be noted that in some cases such as HAF and DICF, the results are almost the same as SVM with Linear Kernel and in the case of HAF−SHAF, the results are slightly better, but still far behind the best score of the SVM with Linear Kernel. Moreover, Appendix A.3f shows how DICF outperforms the rest of the trained models based on AUROC, equal to $0.90$.

| FD | #At | CLS | P | R | F1 | A | BT | CM |
|---|---|---|---|---|---|---|---|---|
| HAF | 302 | SVM−RBF | 0.694 | 0.732 | 0.712 | 0.789 | 23:26 | $\begin{bmatrix} 1182 & 258 \\ 214 & 586 \end{bmatrix}$ |
| HAF-SHAF | 323 | SVM−RBF | 0.771 | 0.715 | 0.742 | 0.822 | 26:25 | $\begin{bmatrix} 1271 & 169 \\ 228 & 572 \end{bmatrix}$ |
| CF | 7 | SVM−RBF | 0.723 | <u>0.760</u> | 0.741 | 0.810 | 6:12 | $\begin{bmatrix} 1208 & 232 \\ 192 & 608 \end{bmatrix}$ |
| DICF | 6 | SVM−RBF | <u>0.932</u> | 0.625 | 0.748 | 0.850 | 8:35 | $\begin{bmatrix} 1404 & 36 \\ 300 & 500 \end{bmatrix}$ |
| BCF | 7 | SVM−RBF | 0.708 | 0.655 | 0.680 | 0.780 | 5:50 | $\begin{bmatrix} 1224 & 216 \\ 276 & 524 \end{bmatrix}$ |
| All | 343 | SVM−RBF | 0.845 | 0.743 | <u>0.791</u> | <u>0.859</u> | 31:35 | $\begin{bmatrix} 1331 & 109 \\ 205 & 595 \end{bmatrix}$ |

*Table 4.7: The results of the SVM algorithm with RBF Kernel on the final data set with different feature definitions. The best scores are underlined.*

### SVM − Stochastic Gradient Descent Training

Stochastic Gradient Descent, SGD for short, is a simple and very efficient approach to discriminative learning of linear classifiers under convex loss functions such as (linear) Support Vector Machines, which has been used for the current part of the experiments. This estimator implements regularized linear models with SGD learning which means the gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing strength schedule (aka learning rate). The list of the important parameters which are used to configure the algorithms can be found in Table 4.3 of Section 4.3 under *SVM−SGD*. As mentioned in the table, the penalty parameter (aka regularization term) is set to *l2*, which is the standard regularizer for linear SVM models. Furthermore, $\epsilon$, which determines the threshold at which it becomes less important to get the prediction exactly right, is set to $0.1$, which is considered as the best amount, generally. For epsilon-insensitive loss functions, *huber* for instance, which is utilized in this experiment, any differences between the current prediction and the correct label are ignored if they are less than this threshold.

The results of the experiment are presented in Table 4.8. According to the results, the best F1 score and Accuracy is obtained by using the DICF features, with $0.765$ for F1 score and $0.864$ for Accuracy. It is noteworthy that the Precision in the case of DICF is measured at $1.000$, and according to the confusion matrix, all $1440$ positive instances are identified correctly, in addition to $304$ negative instances which are guessed to be positive instances. Adequacy of such a classifier depends on the application, and in our case, it interprets to the question, *Is it acceptable if the robot tries to grasp something that might not exist?*. If the answer is yes, then it could be said that the DICF trained model achieved the best result so far with $100\%$ success rate in identifying the positive cases and $62\%$ success rate in identifying the negative cases. Another interesting fact, is that the DICF achieved such results with only $6$ feature values, which corresponds to the attributes of the data set and as a result, the training time, BT, is the lowest with only $1 : 10$ minutes, compared to "All" case with the highest amount of building time, $6 : 04$.

| **FD** | **#At** | **CLS** | **P** | **R** | **F1** | **A** | **BT** | **CM** |
|---|---|---|---|---|---|---|---|---|
| HAF | 302 | SVM−SGD | 0.520 | 0.860 | 0.648 | 0.667 | 2:27 | $\begin{bmatrix} 807 & 663 \\ 112 & 688 \end{bmatrix}$ |
| HAF-SHAF | 323 | SVM−SGD | 0.434 | <u>0.961</u> | 0.598 | 0.538 | 3:02 | $\begin{bmatrix} 438 & 1002 \\ 31 & 769 \end{bmatrix}$ |
| CF | 7 | SVM−SGD | 0.958 | 0.460 | 0.621 | 0.800 | 1:10 | $\begin{bmatrix} 1424 & 16 \\ 432 & 368 \end{bmatrix}$ |
| DICF | 6 | SVM−SGD | <u>1.000</u> | 0.620 | <u>0.765</u> | <u>0.864</u> | 1:07 | $\begin{bmatrix} 1440 & 0 \\ 304 & 496 \end{bmatrix}$ |
| BCF | 7 | SVM−SGD | 0.862 | 0.560 | 0.678 | 0.810 | 1:20 | $\begin{bmatrix} 1368 & 72 \\ 352 & 448 \end{bmatrix}$ |
| All | 343 | SVM−SGD | 0.617 | 0.921 | 0.739 | 0.768 | 6:04 | $\begin{bmatrix} 984 & 456 \\ 63 & 737 \end{bmatrix}$ |

*Table 4.8: The results of the SVM algorithm with SGD training on the final data set with different feature definitions. The best scores are underlined.*

Moreover, Appendix A.3f illustrates the ROC curves of the classifiers of this section. The results obtained from SVM−SGD classifier still do not hit the top scores of the best classifier so far.

### $3-$**Nearest Neighbors**

Moving on, to the next part of the experiments, the data is used with a $3-$NN classifier. It should be noted that in a $k-$NN classifier, there is no building time associated with the results as $k-$NN classification is a lazy-learning approach [19]. Instead, the prediction time is important for such algorithms, especially, in the case of a problem with many calls to the model for prediction.

Table 4.3 of Section 4.3 shows the parameters that are used to build a model under $3-$*NN*. Besides, the final results from the $3-$NN classifier are presented in Table 4.9. Interestingly, BCF outperforms all others, not only in this section of the test but also all other experiment parts that have been conducted so far, with a score $0.926$ in F1 score and $0.945$ in Accuracy. It is notable that the results are obtained using only $7$ features, that is a great achievement. However, as mentioned before, the one drawback of the $3-$NN algorithm is the amount of time it needs to predict. In Table 4.9, rather than Building Time (BT), Prediction Time (PT) is stated in seconds. It can be seen that the amount of time, BCF needed for predicting $2240$ instances, is considerably less than the time for predicting the same sample set using HAF, HAF$-$SHAF and "All" feature definitions. Another important aspect that should be noted, is that the balance between the Precision and the Recall, especially in the case of BCF, unlike what we observed for SVM$-$SGD.

| **FD** | **#At** | **CLS** | **P** | **R** | **F1** | **A** | **PT(s)** | **CM** |
|---|---|---|---|---|---|---|---|---|
| HAF | 302 | $3-$NN | 0.769 | 0.871 | 0.817 | 0.860 | 30.17 | $\begin{bmatrix} 1182 & 258 \\ 214 & 586 \end{bmatrix}$ |
| HAF-SHAF | 323 | $3-$NN | 0.788 | 0.868 | 0.826 | 0.870 | 34.15 | $\begin{bmatrix} 1271 & 169 \\ 228 & 572 \end{bmatrix}$ |
| CF | 7 | $3-$NN | 0.737 | 0.955 | 0.832 | 0.862 | 1.34 | $\begin{bmatrix} 1168 & 272 \\ 36 & 764 \end{bmatrix}$ |
| DICF | 6 | $3-$NN | 0.446 | 0.920 | 0.601 | 0.564 | 1.40 | $\begin{bmatrix} 528 & 912 \\ 64 & 736 \end{bmatrix}$ |
| BCF | 7 | $3-$NN | <u>0.891</u> | <u>0.965</u> | <u>0.926</u> | <u>0.945</u> | 1.45 | $\begin{bmatrix} 1346 & 94 \\ 28 & 772 \end{bmatrix}$ |
| All | 343 | $3-$NN | 0.767 | 0.875 | 0.817 | 0.860 | 40.02 | $\begin{bmatrix} 1228 & 212 \\ 100 & 700 \end{bmatrix}$ |

*Table 4.9: The results of the $3-$NN training on the final data set with different feature definitions. The best scores are underlined.*

In addition to the results, that are presented in this section, it is interesting to observe that the AUROC of BCF is outstandingly high with $0.95$, among all, which are depicted in Appendix A.4f.

### Decision Tree

Based on the results, that were obtained by a $3-$NN classifier, we have guessed that a Decision Tree Classifier must also have good results on our data sets, with an important difference that Decision Trees are eager-learners and they don not have the prediction time problem, due to its importance in this project. The only important configuration setting while training a Decision Tree in this section, is the method for measuring the suitability of a split, which has been decided to be done by *Impurity (Gini)*. However according to Table 4.10, the results are not what we expected. The best results are obtained using DICF, having F1 score and Accuracy under $0.8$ which is far below what we achieved using $3-$NN or SVM. Also the AUROC of these trained models, which are illustrated in Appendix A.5, show that the performance of the Decision Trees on our data set is not quite satisfying.

| **FD** | **#At** | **CLS** | **P** | **R** | **F1** | **A** | **BT** | **CM** |
|---|---|---|---|---|---|---|---|---|
| HAF | 302 | DT | 0.613 | 0.741 | 0.671 | 0.741 | 2:49 | $\begin{bmatrix} 1067 & 373 \\ 207 & 593 \end{bmatrix}$ |
| HAF-SHAF | 323 | DT | 0.622 | 0.705 | 0.661 | 0.741 | 3:37 | $\begin{bmatrix} 1098 & 342 \\ 236 & 564 \end{bmatrix}$ |
| CF | 7 | DT | 0.560 | 0.757 | 0.643 | 0.700 | 1:17 | $\begin{bmatrix} 964 & 476 \\ 194 & 606 \end{bmatrix}$ |
| DICF | 6 | DT | <u>0.650</u> | <u>0.920</u> | <u>0.761</u> | <u>0.794</u> | 1:12 | $\begin{bmatrix} 1044 & 396 \\ 64 & 736 \end{bmatrix}$ |
| BCF | 7 | DT | 0.638 | 0.855 | 0.730 | 0.775 | 1:10 | $\begin{bmatrix} 1052 & 388 \\ 116 & 684 \end{bmatrix}$ |
| All | 343 | DT | 0.645 | 0.760 | 0.698 | 0.765 | 2:40 | $\begin{bmatrix} 1106 & 334 \\ 192 & 608 \end{bmatrix}$ |

*Table 4.10: The results of the Decision Tree on the final data set with different feature definitions. The best scores are underlined.*

### Random Forest

According to the weak outcome of Decision Trees on our data set, it is decided to perform another test on a similar structure, with a slight difference, that is Random Forest classifiers. Random Forest classifiers comprise a certain number of Decision Trees, but they are different in how the trees are created, and the prediction is done using a polling mechanism [20]. Similar to the previous section, the suitability measurement of a split is done by *Impurity (Gini)*, as shown in Table 4.3 of Section 4.3 under *Random Forest*. Furthermore, the number of Decision Trees in the Random Forest is set to $10$, whilst the maximum number of features in each tree is left to the algorithm to decide.

In contrast to the results from the Decision Trees, the results by Random Forests are more promising, with the best F1 score and Accuracy equal to $0.837$ and $0.880$ respectively, in the case of "All" feature definitions, followed by HAF−SHAF and HAF on the second and third places. Nevertheless, the results are not as good as the best results which are achieved so far by other classifiers. ROC curves of the Random Forest classifiers are demonstrated in Appendix A.6 where it can be seen that the best results belong to HAF, HAF−SHAF and "All" with AUROC equal to $0.94$.

| FD | #At | CLS | P | R | F1 | A | BT | CM |
|---|---|---|---|---|---|---|---|---|
| HAF | 302 | RF | 0.761 | <u>0.888</u> | 0.820 | 0.860 | 1:59 | $\begin{bmatrix} 1217 & 223 \\ 89 & 711 \end{bmatrix}$ |
| HAF-SHAF | 323 | RF | <u>0.827</u> | 0.826 | 0.826 | 0.876 | 2:50 | $\begin{bmatrix} 1302 & 138 \\ 139 & 661 \end{bmatrix}$ |
| CF | 7 | RF | 0.658 | 0.840 | 0.738 | 0.787 | 1:06 | $\begin{bmatrix} 1092 & 348 \\ 128 & 672 \end{bmatrix}$ |
| DICF | 6 | RF | 0.503 | 0.560 | 0.530 | 0.645 | 1:05 | $\begin{bmatrix} 998 & 442 \\ 352 & 448 \end{bmatrix}$ |
| BCF | 7 | RF | 0.771 | 0.875 | 0.784 | 0.828 | 1:16 | $\begin{bmatrix} 1156 & 284 \\ 100 & 700 \end{bmatrix}$ |
| All | 343 | RF | 0.812 | 0.865 | <u>0.837</u> | <u>0.880</u> | 3:37 | $\begin{bmatrix} 1280 & 160 \\ 108 & 692 \end{bmatrix}$ |

*Table 4.11: The results of the Random Forest on the final data set with different feature definitions. The best scores are underlined.*

### Summary of Results

Based on the results, which were obtained through different experiments and presented in the previous sections, Figure 4.12 and Figure 4.12 compare the results of the models employing different feature definitions, in terms of Accuracy and F1 score, respectively. As it was discussed before and according to the figures, by far, the best results are achieved using BCF feature definitions on the $3-$NN classifier, with around $0.95$ in Accuracy and $0.93$ in F1 score, compared to the best results, claimed by Fischinger et al. [7], which is $0.857$ using HAF features and $0.743$ using HAF-SHAF. Even the HAF and HAF-SHAF models achieved $0.834$ and $0.817$ after extending the dataset, on the SVM classifiers, which are generally weaker results, comparing to the best scores of the experiments, that are obtained in this project.

The second best score, according to the figures, belongs to "All" feature definitions, on SVM with almost $0.91$ in Accuracy and $0.85$ in F1 score. The comparison between the results, which are obtained from a $3-$NN classifier and the results which are obtained using an SVM classifier can be tricky in the case of our problem due to the fact that grasping process, as defined by the problem statement in Section 1.1, needs a lot of calls to the classification model, in order to find a suitable position for grasping, and as the $3-$NN classifier is a lazy-learner, it follows that the amount of time for predictions are so much higher than an eager-learner classifier such as SVM, and unlike SVM, this amount of time can vary according to the size of the training dataset. On the other hand, it should be considered that the amount of time, a machine needs in total for the case of BCF on $3-$NN might be less, based on the number of operations, that should be executed before calling the trained model, as the computational complexity of a BCF with only 7 feature definitions, can be drastically less, than the computational complexity of the 343 feature definitions of "All".

Another interesting fact is that the Random Forest classifier, performed almost acceptable with all three feature definition sets, HAF, HAF-SHAF and "All". Apart from BCF on $3-$NN and "All" on SVM, Random Forest classifier using HAF, HAF-SHAF and "All" can be considered the best results of all others.

Despite mentioning in previous sections, Figure 4.14 illustrates four ROC curves, HAF and HAF-SHAF on SVM, that come from Fischinger et al. [7], along with BCF on $3-$NN and "All" with SVM, to emphasize how the project successfully improved the results.

Overall, given the fact that, BCF on a $3-$NN reached the Accuracy of $0.945$ and the F1 score of $0.926$, it can be claimed that the outcome of the project is satisfying and promising as the optimisations, on the project by Fischinger et al. [7], has been performed successfully, and the best result of the TFRGA project is improved by nearly $9\%$, in terms of Accuracy.

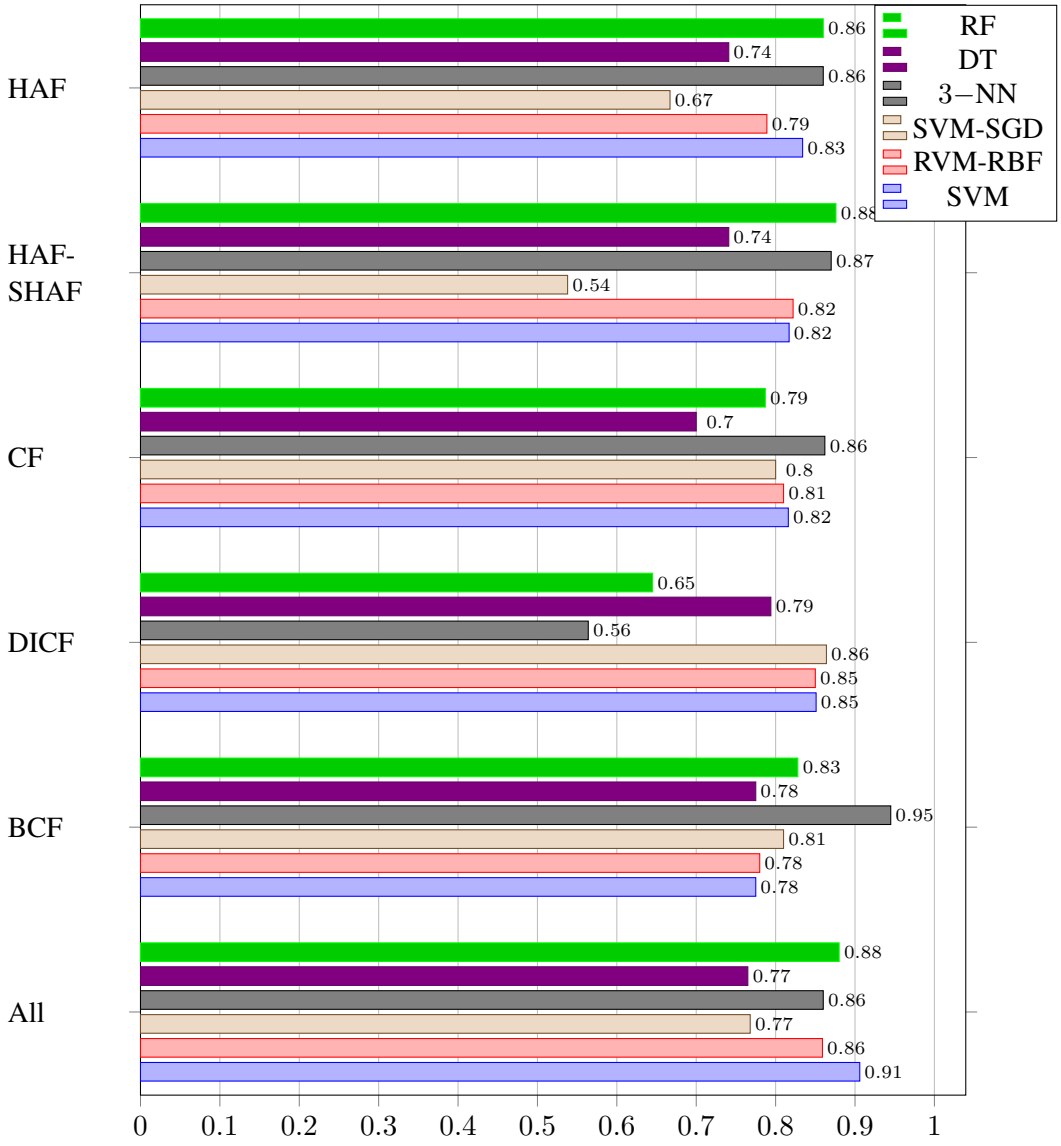



*Figure 4.12: A comparison between the performance of classification algorithms, on the feature values, extracted using different feature definitions. The results are the Accuracy of the trained models on the same test dataset.*

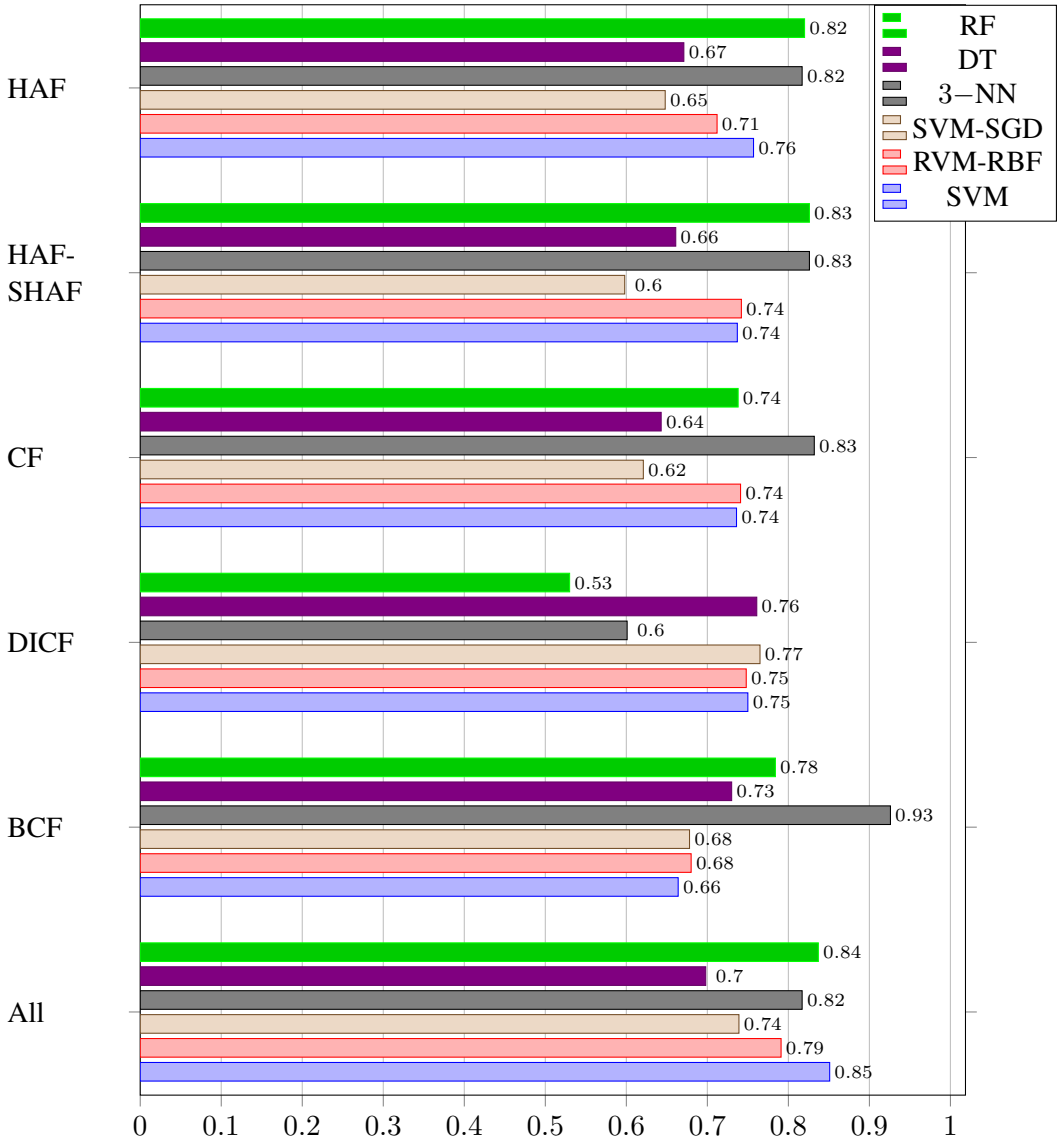
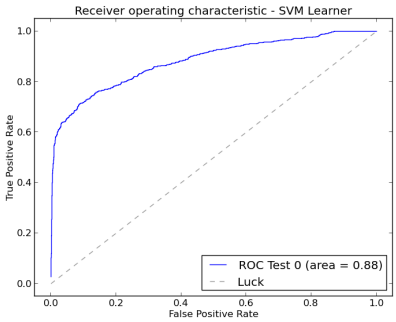
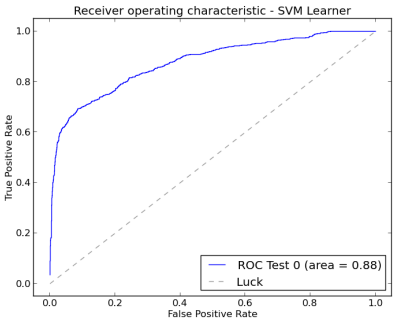


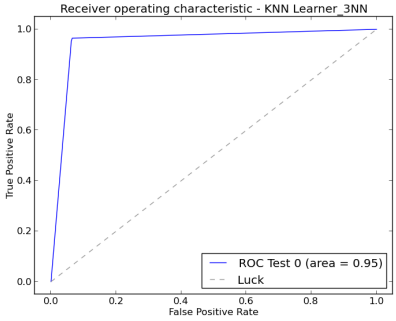

*Figure 4.13: A comparison between the performance of classification algorithms, on the feature values, extracted using different feature definitions. The results are the F1 scores of the trained models on the same test dataset.*



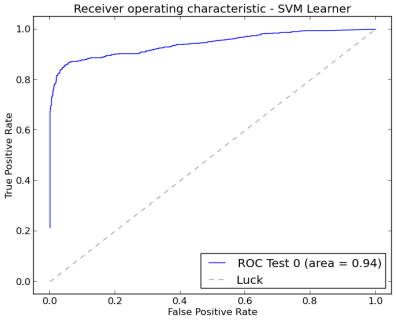

*(a) HAF with SVM*




*(b) HAF-SHAF with SVM*




*(c) BCF with 3−NN*




*(d) "All" with SVM*

*Figure 4.14: ROC curves of the best obtained results in this project, compared with the initial results, from TFGRA project.*

CHAPTER 5

# Conclusion and Future Work

In this chapter, a conclusion is made over the entire project, its goals, steps which have been taken to achieve these goals, and the outcomes, in Section 5.1. Furthermore, Section 5.2 sketches some possible ways to continue and expand this project toward further optimizations and improvements of the solution to the problem which was stated in Section 1.1.

## 5.1 Conclusion

In Definition 1 of Section 1.1, we formally defined the main problem of grasping, as an optimisation function, where within, the problem can be divided into different pieces, one of which a decision problem, that specifies, given a frame of a perception of the surrounding environment, as a scene, either two spots can be found in the center of the scene, to be used for grasping. The main goal of this project has been to optimize and improve the decision function, that is the solution for the mentioned decision problem. By the definitions of the TFRGA project in Fischinger et al. [7], the stated decision function is called so many times by other functions, in each Decision round of the iterations that were shown in Figure 1.1.

In order to achieve any improvement by changing or replacing any part of a solution, it should be possible to measure how the changes impact the success rate of the solution. Thus, as the first step in this project, a testing application has been developed, that allows making changes in different sections of the process, from input scene's features to classification algorithms and their parameters, and informs about the outcome of the changes. In Chapter 4 the testing application, its architecture, and abilities have been discussed.

As the main part of the core of the decision algorithm contains a classification model, another step toward making the experiments' results more robust and rigid, is to extend the training instances database. This step has been done in this project and the details on how to collect, transform, replicate, and prepare the new instances have been provided in Chapter 4, Section 4.1.

Moving on, it should be tested that, whether it is possible to achieve as good results as what was shown in TFRGA, with lighter and smaller feature set definitions. Therefore, three new

feature definitions have been proposed and discussed in Chapter 3, which are called Circular Feature Definition (CF), Differential Intra-Circular Feature Definition (DICF), and Bell-Circular Feature Definition (BCF), in order to be tested as both substitutes and complements for HAF and SHAF from TFRGA.

In addition to the new features, more classification algorithms should be tested, in order to make sure that the most effective classification algorithm is used to making binomial decisions based on the feature values of the scenes. As a result, a list of some popular and well-known has been made, consisting of Support Vector Machines with both Linear and RBF kernels, $k-$NN, Decision Tree, and Random Forest, in order to conduct more experiments, utilizing the mentioned classification algorithms, instead of the Linear kernel SVM, that was employed in TFRGA.

Finally, some different experiments have been performed on the final dataset, which was made out of the training instances from TFRGA, along with the collected and processed instances of this project, utilizing different classification algorithm, and using HAF and SHAF from TFRGA, and the three newly defined feature set definitions, CF, DICF, and BCF in different combinations. The results of the experiments were presented and discussed in Chapter 4 and the chapter has been concluded with a summary over all results, in Section 4.5. Reference to the summary, the project has made it possible to optimize the decision problem with an Accuracy of $0.95$, with a $3-$NN classification trained model, compared to the best results, claimed by Fischinger et al. [7], which was $0.857$ using HAF features and $0.743$ using the combination of HAF and SHAF.

## 5.2 Future Work

There are many steps, that can be made, toward further optimizations and improvements of the project, such as performing more experiments with different classification algorithms which have not been used in this project, and altering the parameters of the algorithms to achieve better results.

Another improvement on the developed testing application could be to have a possibility to run more than one classifier algorithm in parallel on the same dataset, and having some measures over the performance difference such as Statistical Significance Test [5]. Furthermore, the ability to ensemble classifiers could be a new approach for optimizing such a project as it has been observed in the results that different classifiers are capable of predicting with different patterns. Some predict positive instances better than the others whilst some are better with negative instances.

# Bibliography


[1] Shawkat Ali and Kate A Smith. On learning algorithm selection for classification. *Applied Soft Computing*, 6(2):119–138, 2006.

[2] Dmitry Berenson, Rosen Diankov, Koichi Nishiwaki, Satoshi Kagami, and James Kuffner. Grasp planning in complex scenes. In *Humanoid Robots, 2007 7th IEEE-RAS International Conference on*, pages 42–48. IEEE, 2007.

[3] Pavel B Brazdil and Carlos Soares. A comparison of ranking methods for classification algorithm selection. In *Machine Learning: ECML 2000*, pages 63–75. Springer, 2000.

[4] River Bank Computing. Pyqt. *PyQt is available online at http://www. riverbankcomputing. co. uk/, visited on June*, 13, 2004.

[5] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, 7(Jan):1–30, 2006.

[6] ROS Documentation. http://wiki.ros.org/. Accessed: 2016-01.

[7] David Fischinger. Enabling autonomous robotic grasping based on topographic features, phd thesis. *Vienna University of Technology*, 2014.

[8] David Fischinger and Markus Vincze. Empty the basket-a shape based learning approach for grasping piles of unknown objects. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 2051–2057. IEEE, 2012.

[9] David Fischinger, Markus Vincze, and Yun Jiang. Learning grasps for unknown objects in cluttered scenes. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 609–616. IEEE, 2013.

[10] David Fischinger, Astrid Weiss, and Markus Vincze. Learning grasps with topographic features. *The International Journal of Robotics Research*, page 0278364915577105, 2015.

[11] freenect Documentation. http://wiki.ros.org/freenectcamera. Accessed: 2016-01.

[12] Raul Garreta and Guillermo Moncecchi. *Learning scikit-learn: machine learning in python*. Packt Publishing Ltd, 2013.

[13] Arnulf Graf, Alexander J Smola, and Silvio Borer. Classification in a normalized feature space using support vector machines. *Neural Networks, IEEE Transactions on*, 14(3):597–605, 2003.

[14] L. Guo, M. Zhang, Y. Wang, and G. Liu. Environmental perception of mobile robot. pages 348–352, 2006. cited By 2.

[15] James A Hanley and Barbara J McNeil. The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology*, 143(1):29–36, 1982.

[16] Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin, et al. A practical guide to support vector classification, 2003.

[17] Kai Huebner and Danica Kragic. Selection of robot pre-grasps using box-based shape approximation. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 1765–1770. IEEE, 2008.

[18] Sašo Karakatič and Vili Podgorelec. Improved classification with allocation method and multiple classifiers. *Information Fusion*, 31:26–42, 2016.

[19] Sotiris B Kotsiantis, I Zaharakis, and P Pintelas. Supervised machine learning: A review of classification techniques, 2007.

[20] Andy Liaw and Matthew Wiener. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

[21] Andrew T Miller, Steffen Knoop, Henrik Christensen, Peter K Allen, et al. Automatic grasp planning using shape primitives. In *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, volume 2, pages 1824–1829. IEEE, 2003.

[22] Qt Nokia. A cross-platform application and ui framework, 2009, 2009.

[23] A Python Computing Package Numpy. http://www.numpy.org/. Accessed: 2015-08.

[24] Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[25] openni_camera Documentation. http://wiki.ros.org/opennicamera. Accessed: 2016-01.

[26] Chavdar Papazov, Sami Haddadin, Sven Parusel, Kai Krieger, and Darius Burschka. Rigid 3d geometry matching for grasping of known objects in cluttered scenes. *The International Journal of Robotics Research*, page 0278364911436019, 2012.

[27] pcl Documentation. http://wiki.ros.org/pcl. Accessed: 2016-01.

[28] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research*, 12:2825–2830, 2011.

[29] Markus Przybylski, Tamim Asfour, and Rüdiger Dillmann. Planning grasps for robotic hands using a novel object representation based on the medial axis transform. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 1781–1788. IEEE, 2011.

[30] Riverbank Computing Limited PyQt. https://riverbankcomputing.com/software/pyqt/intro. Accessed: 2015-08.

[31] A cross-platform application framework Qt. https://en.wikipedia.org/wiki/Qt. Accessed: 2015-08.

[32] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5, 2009.

[33] John R Rice. The algorithm selection problem. 1975.

[34] Alejandro Rosales-Pérez, Jesus A Gonzalez, Carlos A Coello Coello, Hugo Jair Escalante, and Carlos A Reyes-Garcia. Multi-objective model type selection. *Neurocomputing*, 146:83–94, 2014.

[35] rviz Documentation. http://wiki.ros.org/rviz. Accessed: 2016-01.

[36] Machine Learning in Python scikit learn. http://scikit-learn.org/stable/. Accessed: 2015-08.

[37] The Python Tutorial. https://docs.python.org/2/tutorial/. Accessed: 2015-08.

[38] Guido Van Rossum et al. Python programming language. In *USENIX Annual Technical Conference*, volume 41, 2007.

[39] Ingrid Visentini, Lauro Snidaro, and Gian Luca Foresti. Diversity-aware classifier ensemble selection via f-score. *Information Fusion*, 28:24–43, 2016.

[40] Xu-Cheng Yin, Kaizhu Huang, Chun Yang, and Hong-Wei Hao. Convex ensemble learning with sparsity and diversity. *Information Fusion*, 20:49–59, 2014.

[41] Zhengyou Zhang. Microsoft kinect sensor and its effect. *MultiMedia, IEEE*, 19(2):4–10, 2012.

APPENDIX A

# Appendices

*Code Part A.1: How select a classifier*

```
SVM*, DecisionTree, KNN, SGD, RandomForest
```

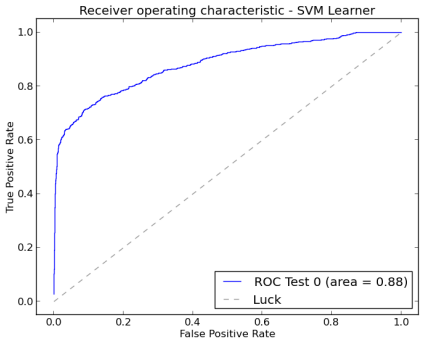
*Code Part A.2: How to define parameters for a new classifier*

```
def PrepareParameters():
    LearnerParameters.update(
                  {
                      'kernel': LearnerParameterUnit('kernel', 'Kernel
                         function', 'linear'),
                      'C' : LearnerParameterUnit('C', 'Penalty
                         parameter C ', '1.0'),
                      'max-iter' : LearnerParameterUnit('max-iter',
                         'Maximum iteration', '-1'),
                      'test-por' : LearnerParameterUnit('test-por',
                         'Test set portion', '0.5')
                  })
```

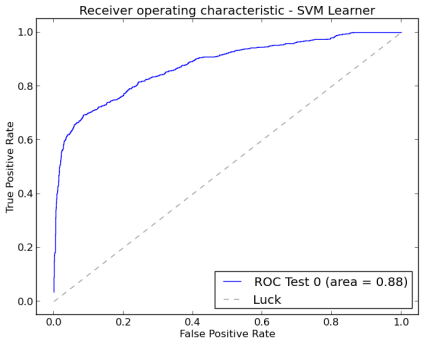
*Code Part A.3: How to declare the new classifier*

```
class SVMLearner(LearnerBase):
```

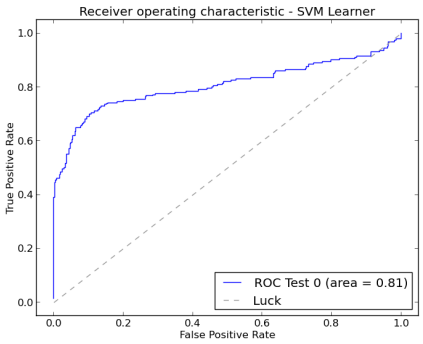
```
2
3       def __init__(self):
4
5           super(SVMLearner, self).__init__()
6
7           self.learner_name = 'SVM Learner'
8
9           self.clf = svm.SVC(
10                              kernel=
                                    LearnerParameters['kernel'].value,
11                              C=
                                    float(LearnerParameters['C'].value),
12                              max_iter=
                                    int(LearnerParameters['max-iter'].value)
13                              )
```



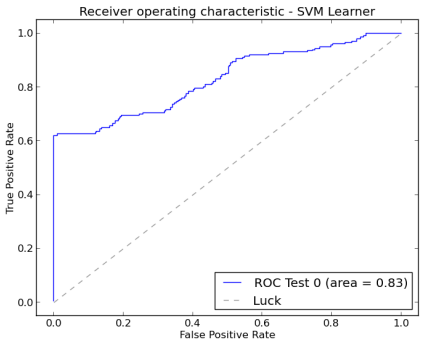

*(a) HAF*



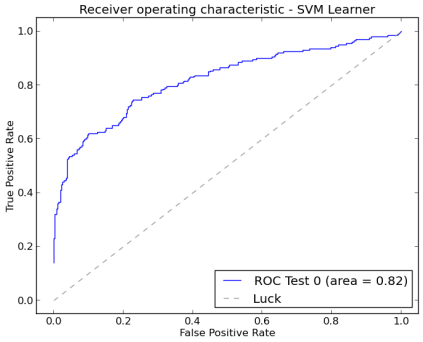

*(b) HAF-SHAF*



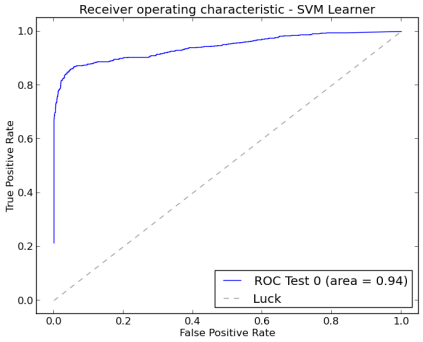

*(c) CF*




*(d) DICF*

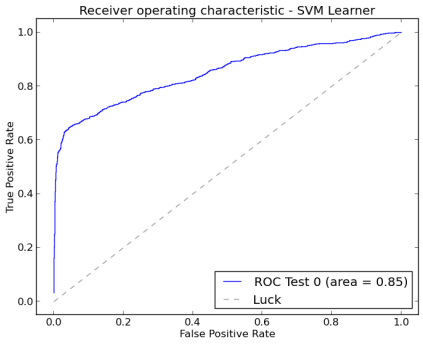



*(e) BCF*

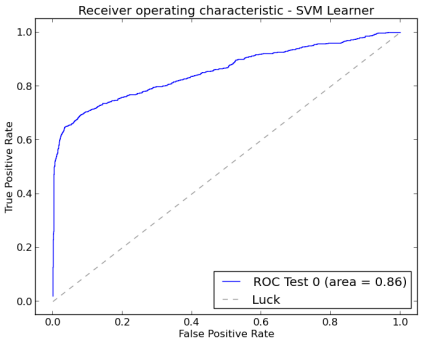



*(f) All*

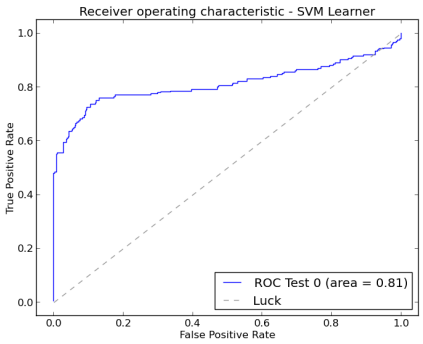
*Figure A.1: ROC curves of the SVM with Linear Kernel trained models over final data set with* 6 *different sets of feature definitions.*



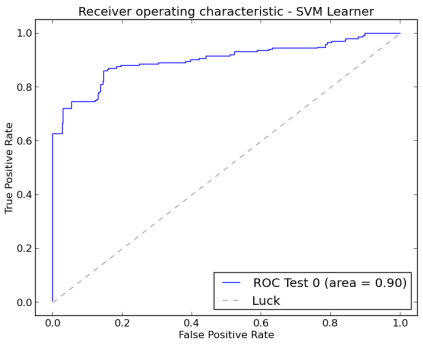

*(a) HAF*



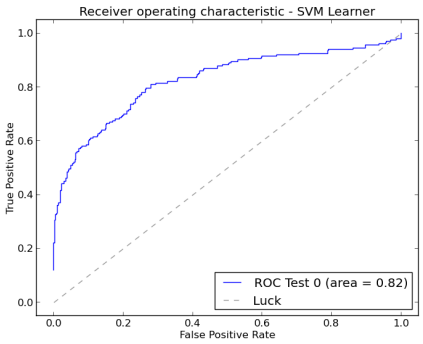

*(b) HAF-SHAF*



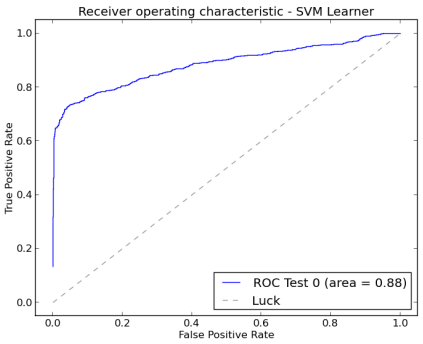

*(c) CF*




*(d) DICF*

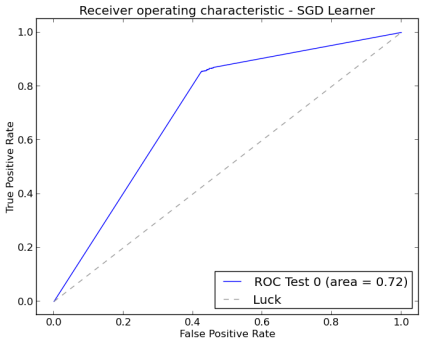



*(e) BCF*

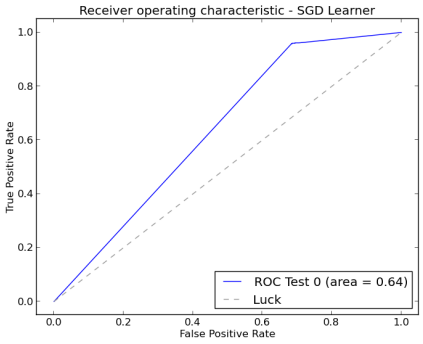



*(f) All*

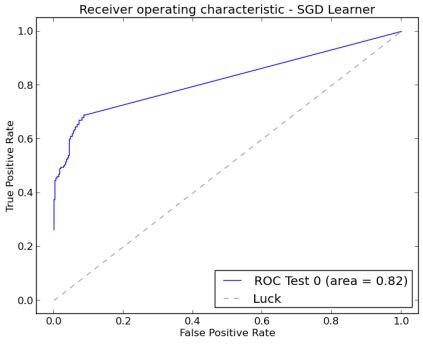
*Figure A.2: ROC curves of the SVM with RBF Kernel trained models over final data set with 6 different sets of feature definitions.*



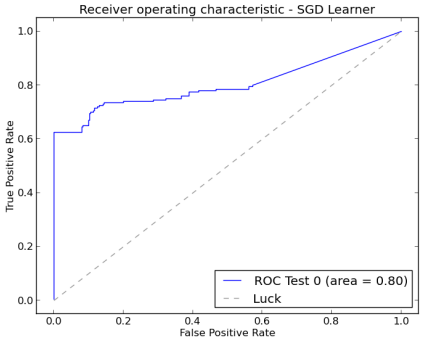

*(a) HAF*



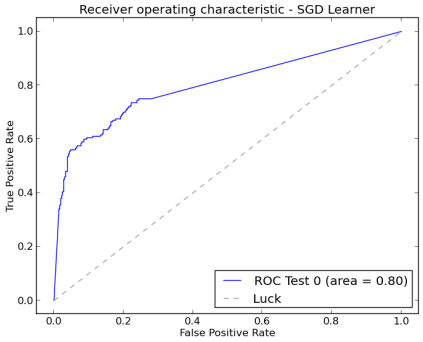

*(b) HAF-SHAF*



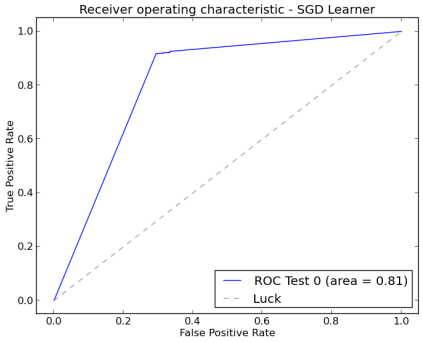

*(c) CF*




*(d) DICF*

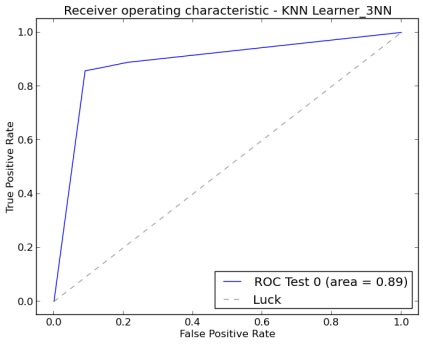



*(e) BCF*

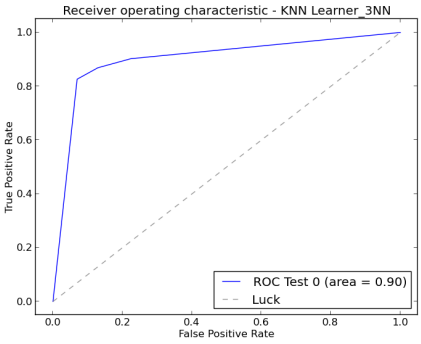



*(f) All*

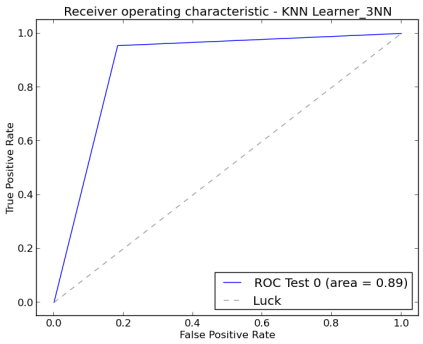
*Figure A.3: ROC curves of the SVM with SGD training trained models over final data set with* 6 *different sets of feature definitions.*



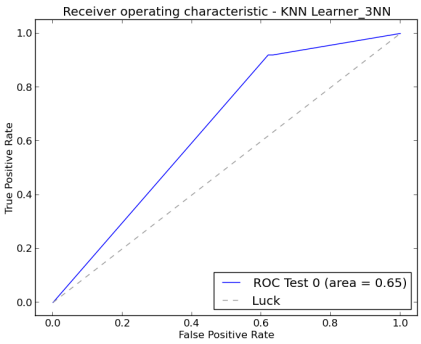

*(a) HAF*



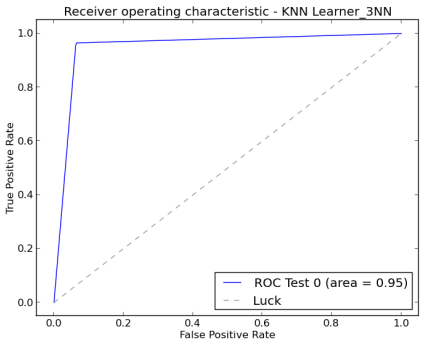

*(b) HAF-SHAF*



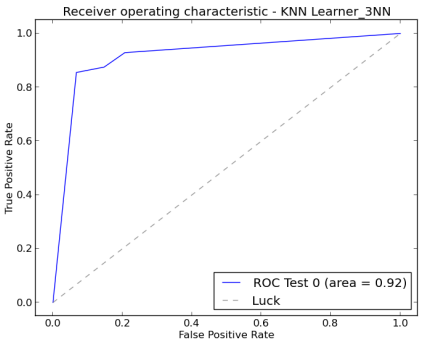

*(c) CF*




*(d) DICF*

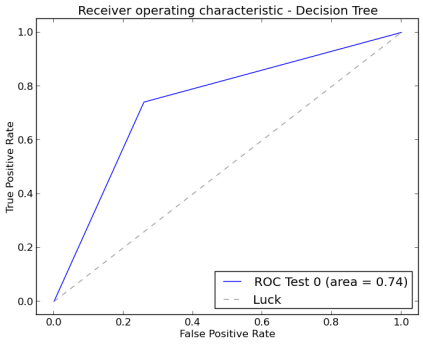



*(e) BCF*

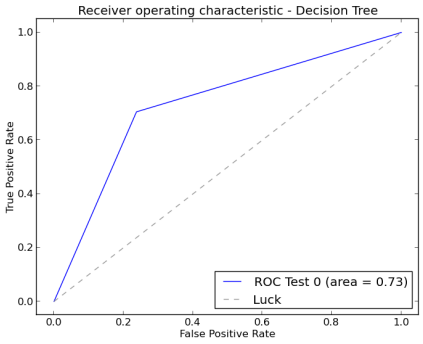



*(f) All*

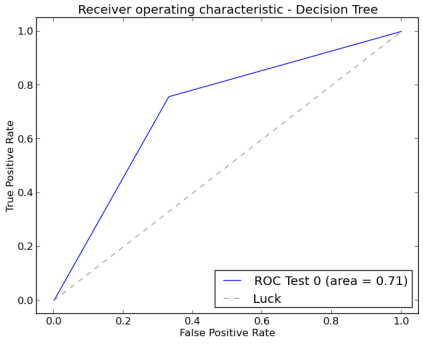
*Figure A.4: ROC curves of the 3−NN trained models over final data set with 6 different sets of feature definitions.*



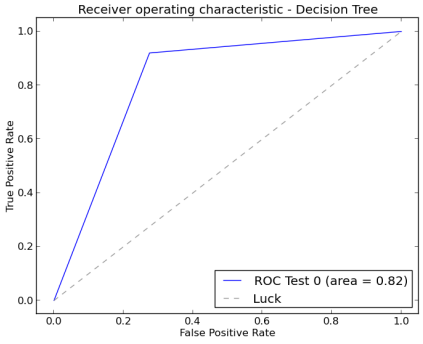

*(a) HAF*



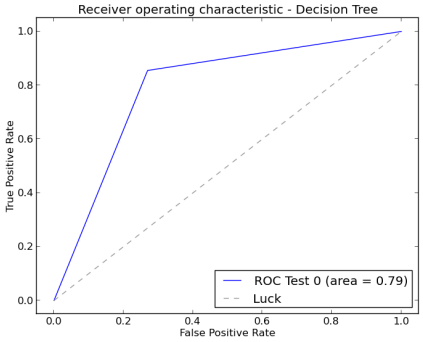

*(b) HAF-SHAF*



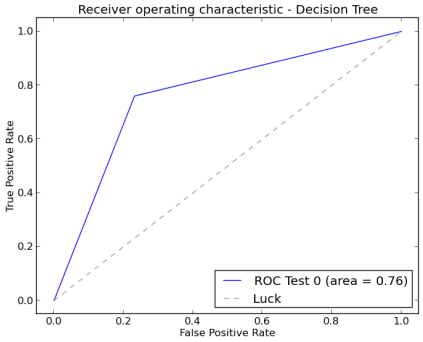

*(c) CF*




*(d) DICF*

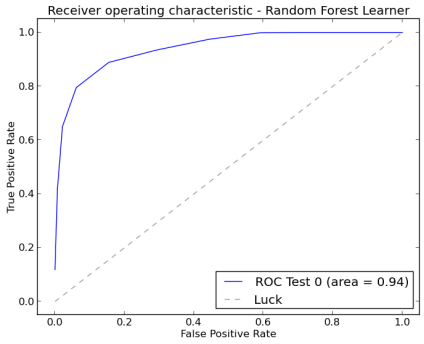



*(e) BCF*

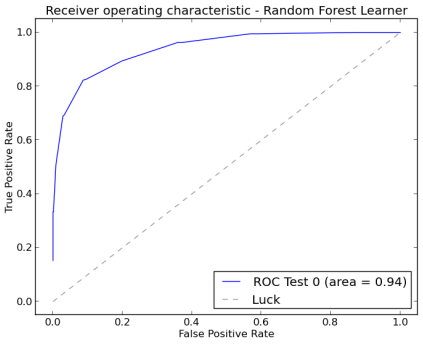



*(f) All*

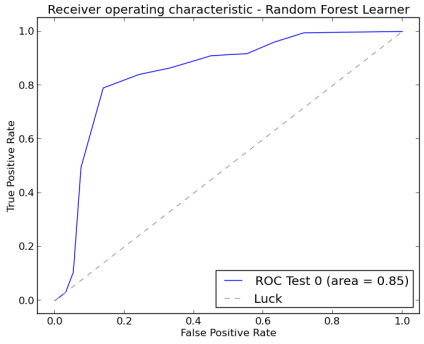
*Figure A.5: ROC curves of the Decision Tree trained models over final data set with 6 different sets of feature definitions.*



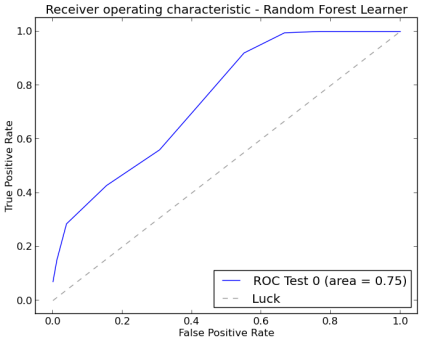

*(a) HAF*



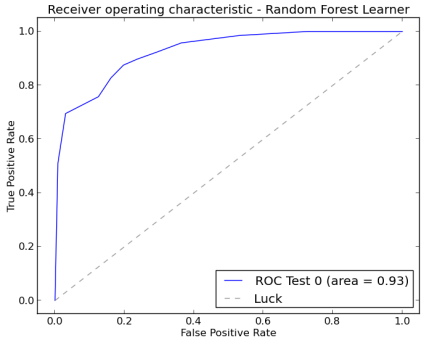

*(b) HAF-SHAF*



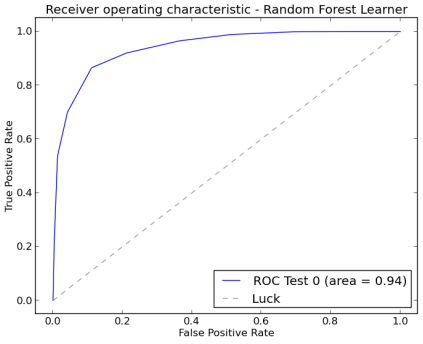

*(c) CF*




*(d) DICF*




*(e) BCF*




*(f) All*

*Figure A.6: ROC curves of the Random Forest trained models over final data set with 6 different sets of feature definitions.*