Learning to Pour Using Warped Features

Einschenken lernen mit Hilfe von Warped Features
Bachelor-Thesis von Sascha Brandl aus Erbach
April 2014

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Learning to Pour Using Warped Features
Einschenken lernen mit Hilfe von Warped Features

Vorgelegte Bachelor-Thesis von Sascha Brandl aus Erbach

1. Gutachten: Prof. Dr. Jan Peters
2. Gutachten: M.Eng. Oliver Krömer

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 28.04.2014

(Sascha Brandl)

Abstract

If we want to use robots to perform everyday tasks they have to work with various objects. A robot can be trained using demonstration and reinforced learning on single objects. Though it would take too much time to train a robot for every object from scratch. The robots have to adapt and generalize their knowledge so they can perform the tasks with other object.

In this thesis an approach is introduced how to generalise the knowledge using high level features for a pouring task. The high level features depend on the shapes and parts of objects. They are directly computed from the parts of an object. Therefore the parts of an objects have to be identified. In this thesis model meshes were created using a 3D-scanner and the marching cubes algorithm. By labelling the vertices of a mesh to get the object parts the high level features can be directly computed using the labelled vertices. First a source mesh was manual labelled to avoid an initial error. Afterwards other objects were labelled using a warping approach and the already labelled mesh.

To test the high level features experiments were run on a simulator. The simulator is based on the bullet physic engine and uses another project to integrate a fluid simulation. In the first experiment a classifier was trained on one object to identify fluid particles which will increase the fluid volume in a container. In the second experiment another classifier was trained on one object to identify when a container starts to pour. Both classifier were used on other objects to see how well the high level features generalises. For both experiments the learned classifiers were able to generalise between the objects and were producing good results.

Contents

List of Figures

1 Motivation

In the future, it is most likely that robots will be used to help in the household by executing everyday-tasks. For example, a robot could be used to serve drinks to guests. One way to achieve this is to train the robot with demonstration and reinforced learning on single objects (see [KOP10]). Every household will have various objects though. It would take too much time to train the robot for every object from scratch. Therefore, robots will have to adapt and generalize their knowledge so they can perform the task with other objects.

This thesis is about learning to pour. Pouring is not an easy task as it involves manipulating fluids. The task depends on the overall shape of the object. For example, it is important to consider the rim of a cup to know when it starts to pour and where the fluid will end up. When pouring into another object the fluid volume inside this object is also an important factor. The handle is another part of the cup but is generally not important for pouring. Because the robot has to consider multiple object parts, pouring is a complex manipulation task.

As humans, we know from our own experience what is important for pouring. For example, the lower lip of the rim of one cup has to be above the opening of the other container. Using this knowledge, high level features can be defined and used for the learning process. These high level features depend on the shape of an object and its parts. After the robot has learned pouring with one object, this knowledge can be generalized by using a warping algorithm. In this thesis the non-parametric warping from Hillenbrand is used. Warping an objects means to transform the shape of one object into another. The warping algorithm is used to find the corresponding object parts of an unknown target shape using an already labelled source shape. Therefore the high level features for the unknown target shape can be computed as they depend on these corresponding object parts.

It is difficult and dangerous to do experimental pouring tasks on a real robot. For this thesis a simulator is therefore used for the fluid simulation. The simulator is based on the *bullet physic library* and uses an extension to integrate the FLUIDS V.2 algorithm for a fluid simulation based on the *Smoothed Particle Hydrodynamics* method (see [bul13]). To run the experiments and get the data from the simulator some models are needed. Chapter two is about the model generation. In the third chapter the warping is explained in more detail. A modified version of the warping algorithm from Hillenbrand is used to determine correspondences between objects. The fourth chapter introduces the high level features and how they can be used for generalisation between objects.

2 Related Work

There are different approaches which are related to this work. First of all Tamosiunaite et al. were trying to learn a pouring task on a robot arm by combining goal and based shape learning for dynamic movement primitives. When changing the object they had to relearn but they had to do less trials as learning completely anew. They came to the conclusion that reinforced learning may be successful for modifying a learned trajectory with respect to small changes in the configuration of the task (see [TNUW11]).

Another approach was done by Rozo et al.(see [RJT13]). They tried to teach a robot to pour 100ml drinks from a bottle into a plastic glass. A force-torque sensor was attached at the wrist of the robot. In the training phase the human teleoperates the robot using a haptic device. The data from the demonstration is statistically encoded by a parametric hidden Markov model. This way the robot can learn to adapt to different filling heights of the bottle. They were also be able to generalize between different bottles.

The work from Kunze et al. is about using a physic simulation to predict and consider physical effects for a robot action (see [KDGB11]). The idea is to use a temporal projection system which translates naive physic problems into parametrized physical simulation tasks. The data is logged during the execution of the simulation and provided as a symbolic time-interval-based first-order representation.

Another interesting work was done by Zöllner et al. (see [ZPKD05]). They wanted to build a knowledge base of manipulation tasks by extracting knowledge from demonstrations of manipulation problems. Thee goal was to compare the newly acquired skill or task with already existing task knowledge. Based on the comparison the system decides to whether add a new task representation or expand an existing one.

3 Generating the 3D models

The generation of the 3D shape models is divided into several steps. The first step is to scan a real object with a kinect camera to create a point cloud of the object. The second step is to learn an implicit surface function from this point cloud. The third step is to generate the surface mesh and normals from the learned implicit surface function. In the last step the generated mesh is smoothed using a LaPlacian smoothing algorithm.

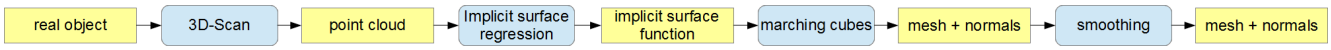
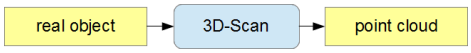


Figure 3.1: Model Generation Pipeline

3.1 3D-Scan



The first step for creating the models is to scan real objects. Therefore a 3D-Scanner is used. The hardware set-up of the 3D-Scanner is a kinect camera with a turning-table. The software was written during the *Bachelorpraktikum 2013* at the *TU Darmstadt*. The software uses the *pcl library* to process the point clouds delivered from the kinect camera. Some examples for scanned objects are shown in figure 3.4.

The 3D-Scanner is used to get point clouds from the real objects as a first model. The objects used in this thesis are shown in figure 3.3. The point clouds will be used to create a surface. The problem is that only the outside of the object is scanned. So the inside of the object has to be computed.

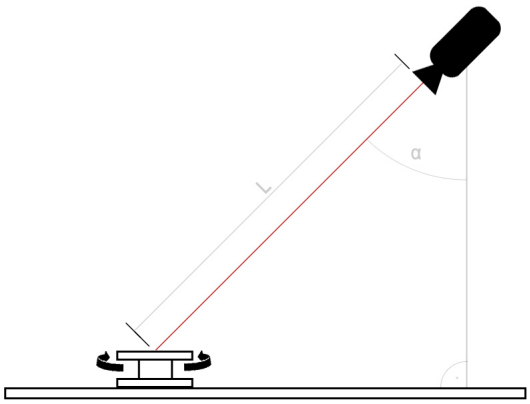
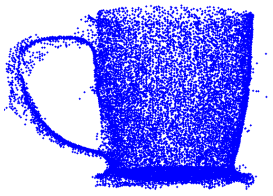


Figure 3.2: Hardware set-up of the 3D-Scanner

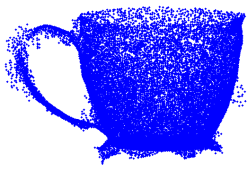
Figure 3.3: All objects used in this thesis

(a) Red cup



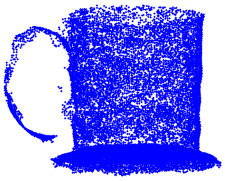
(b) Scanned red cup

(c) Broad cup



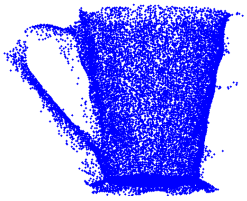
(d) Scanned broad cup

(e) High cup



(f) Scanned high cup

(g) Cup with small base and broad rim

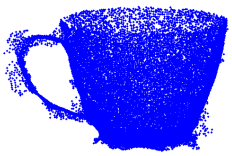


(h) Scanned cup with small base and broad rim

(i) Standard cup

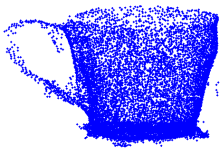
(j) Scanned standard cup

(k) Very broad cup



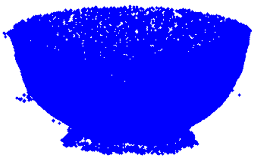
(l) Scanned very broad cup

(m) Very small cup



(n) Scanned very small cup

(o) Standard bowl



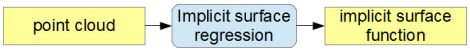
(p) Scanned standard bowl

(q) Square bowl

(r) Scanned square bowl

Figure 3.4: Scanned objects from the 3D-Scanner

3.2 Implicit surface regression



After the point clouds are collected a mesh of the object must be created. The marching cubes algorithm is used for the actual surface mesh generation. This algorithm uses a scalar field as an input. This scalar field is a function which tells the algorithm if a position is part of the object or not. The next step is therefore to learn this scalar field. In the paper *Gaussian Process Implicit Surfaces for Shape Estimation and Grasping* from Dragiev et al. Gaussian processes were used to represent uncertain shape estimates (see [DTG11]). They conditioned Gaussian processes to learn an implicit surface function from haptic, visual and laser feedback. The implicit surface function indicates if a position is part of the object. This function can be used as the scalar field for the marching cube algorithm and is defined as

$$f(x) = \begin{cases} = \tau & \text{x on the surface} \\ < \tau & \text{x outside the object} \\ > \tau & \text{x inside the object} \end{cases}$$

The value 0.99 was used in this thesis for τ. Instead of using Gaussian processes to learn this function, we used kernel regression with a Gaussian kernel. The Gaussian kernel is defined as ([Bis06, p.296]):

$$k(x, x') = exp(-||x - x'||^2/2\sigma^2)$$

The parameter σ determines the width of the Gaussian. We used 0.01 for σ in this thesis. If σ is too small the created meshes will have holes. On the other side choosing a too big σ the openings of the object will be closed. For example in figure 3.5 d) the opening of the handle was closed.

By adding a Gaussian at every given point from the point cloud an estimate of the implicit surface function can be learned. A weight value is computed for every Gaussian. The following function is used to compute the weight vector $\boldsymbol{\alpha}$ (see also [Bis06, p.294]):

$$\boldsymbol{\alpha} = (\mathbf{K} + \lambda \mathbf{I_N})^{-1}\mathbf{t}$$

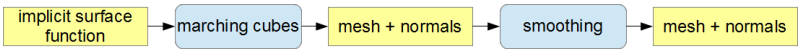
where $\mathbf{K}$ is defined as $K_{nm} = k(x_n, x_m)$, the input x_n and x_m are training data, $\lambda\mathbf{I_N}$ is the regularisation factor to avoid over-fitting and the vector $\mathbf{t}$ is the expected outcome for the implicit surface function. In our case $\mathbf{t}$ is a vector filled with 1 as our training data consists only of points from the inside of the object. After computing the weight vector the implicit surface can be computed with the following function:

$$y(x) = \mathbf{k}(x)^T\boldsymbol{\alpha}$$

where the n^{th} element of the vector $\mathbf{k}$(x) is given by $k(x_n, x)$.

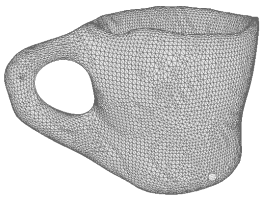
After the regression, the learned implicit surface function can be used as the scalar field for the marching cubes algorithm. By using this method the inner surface of the container will also be generated when using the marching cubes algorithm. Though there are some problems using this method. The inner surface is only estimated which results in an underestimation of the container. We assume that the inner and outer surfaces are close to each other without knowing the actual width. Also this method does not provide good results for all objects. In figure 3.5 d) and g) the openings of the handles were getting closed. Reducing the parameter σ did not improve the results. Instead the objects were getting holes in their surfaces. This method can be changed for a better one as it is only one step of the pipeline. However the results of this method will suffice for our purposes.

3.3 Marching cubes

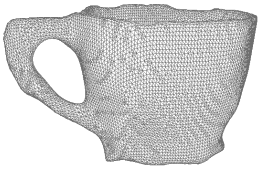


The next step of the model generation is the creation of the surface mesh and the computation of the surface normals.The marching cubes algorithm is used to create the mesh. The marching cubes algorithm proceeds through a grid of points. The algorithm takes eight neighbour points of this grid at a time to form an imaginary cube. The algorithm then determines how to fuse the points to create a surface. This is done by computing an index for a precomputed array of all 2^8 possibilities. The index is 8-bit long and every bit represents one of the eight neighbour point. If the neighbour point is part of the object the bit is set to 1. The scalar field f(x) is used to determine if a grid point is part of the object. By computing the gradient of the scalar field for each grid point we also get the normal vectors. See the work of [LC87] for more details.

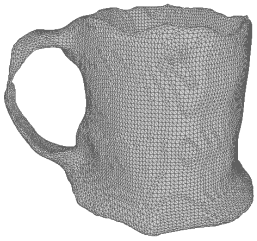
The marching cubes algorithm thus computes the surface mesh and the normal vectors for each point. After the initial mesh generation, the mesh is smoothed to reduce errors caused by the noisy 3D-Scanner which result in an uneven surface. For the smoothing approach the *Laplacian smoothing* is used. The algorithm averages the vertices using the nearest neighbours. This smoothing leads to the results shown in figure 3.5



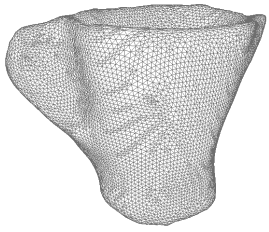
(a) Red cup



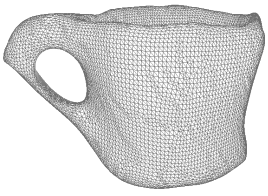
(b) Broad cup



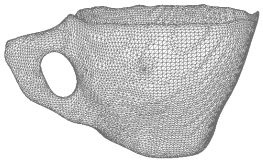
(c) High cup



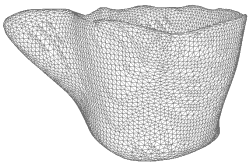
(d) Cup with small base and broad rim



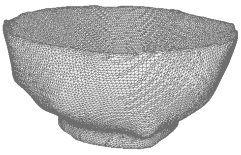
(e) Standard cup



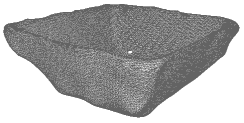
(f) Very broad cup



(g) Very small cup



(h) Standard bowl



(i) Square bowl

Figure 3.5: The meshed objects

4 Warping

Having generated meshes of our objects, we now need to determine correspondences between the points on different objects. The idea of warping is to transform the shape of one object into another. The high level features depend on the parts of an object. Therefore they have to be identified. The idea is to use a source mesh where the parts are identified and warp this source mesh into an target mesh, such that the parts of the source mesh are warped onto the corresponding parts of the target mesh. In this manner, the parts of the unknown target mesh can be identified. For the warping, the vertices of the surface mesh will be used. Therefore the high level features depend on the vertices of the corresponding object parts. The warping algorithm will be used to label the vertices. The object parts which will be identified to compute the high level features are the *rim*, *handle* and *container*. The *rim* is the part where the fluids will start to pour out. The *container* is the inside part of the object which will contain the fluid.

The warping algorithm used in this thesis is based on the work of U.Hillenbrand (see [Hil10]). The warping algorithm consists of three steps:

1. Deformation-tolerant pose estimation
2. Correspondence estimation
3. Point mapping

In the first step, the rotation matrix and translation vector are computed to align the objects. In the second step the estimation of the correspondences between the points is computed. This step is performed by using the normals of the source and target points and minimizing the distances between the points. Finally in the last step the points from the source object are mapped to the corresponding points of the target object.

The correspondence estimation and the surface-point mapping is the same as for the original warping approach. However the first step is modified to compute a scaling of the object to compensate for the size difference of objects. The following pipeline is used to compute the warping:

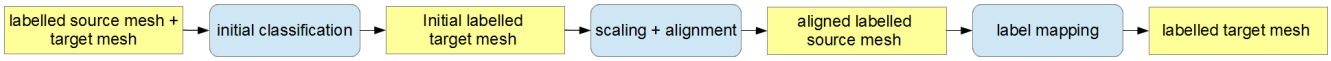
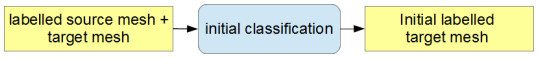


Figure 4.1: Warping Pipeline

As the input, the pipeline is provided with a mesh as the source object and the target object mesh. The parts of the source object are assumed to be labelled. In the first step of the pipeline the target mesh will be initial labelled using a classifier learned from the labelled source mesh. Using this initial labelling, both objects can be aligned according to the corresponding object parts. The source mesh will be scaled to the size of the target mesh. After the scaling, the source mesh will be aligned to the target mesh according to the corresponding objects parts which were identified during the first step. In the last step the labels of the source mesh will be mapped to the vertices of the target shape according to the correspondence estimation. After the mapping is done the object parts of the target mesh are identified.

4.1 Initial Classification



The alignment of both objects is done by matching the corresponding object parts. This is bootstrapping the algorithm because the parts are identified by the warping approach. However a rough labelling for the target mesh can be achieved using a classifier. The idea is to learn a classifier from the source mesh to identify the object parts of the target mesh. This way both meshes can be aligned using the corresponding object parts. Once the parts are aligned, the warping will obtain a better result for the final labelling of the target mesh. The following section is about the classifiers used in this thesis.

4.1.1 Features for the initial classification

The initial classification is based on local features only. The classifier is trained to recognize the points of an object part. Thus the vertices of the meshes can be labelled according to the corresponding object parts. The three features used for this approach are defined below.

Shift to Mean

The feature *Shift to Mean* computes the distance from the given point and the mean point of the neighbours. The intuition behind this features is to identify edges. If the given point is at an edge the computed distance to the mean point will increase. If the given point is in a surface without an edge the distance will be near zero. See also figure 4.2 for an example.

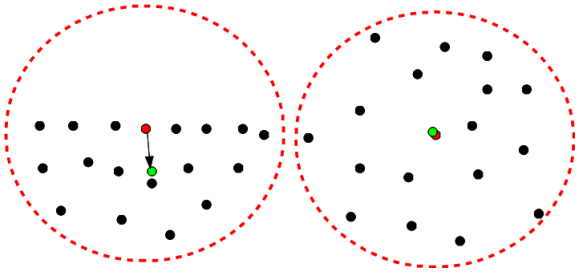


Figure 4.2: Example for the *Shift to Mean* feature

Radius Scale

For the feature *Radius Scale* the number of points in a given radius are counted. After this the number of points in the doubled radius are counted and the scale between the number of points is computed. The intuition behind this is to identify "1D" - structures like handles. The figure 4.3 shows an example behind this intuition.

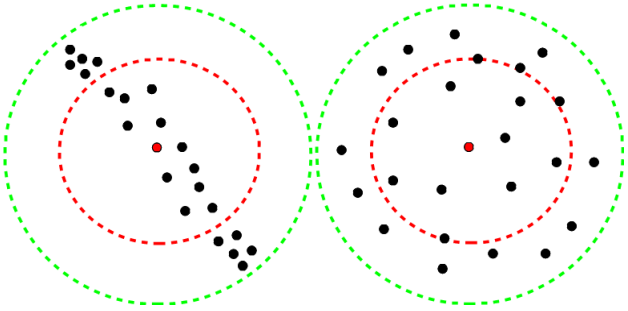


Figure 4.3: Example for the *Radius Scale* feature

Ordered Eigen Values

The last feature is the *Ordered Eigen Values*. Given an input point and a radius it computes the covariance matrix of the points inside the radius. The ordered eigen values represents the scaling of these points in the direction of all axes. See also figure 4.4 for an example:

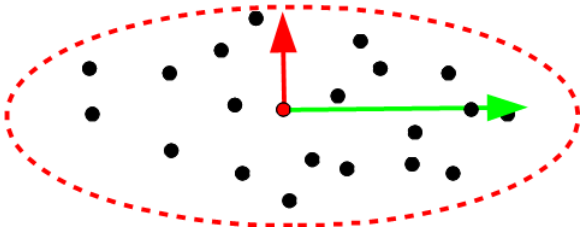


Figure 4.4: Example for the *Ordered Eigen Values* feature

4.1.2 Classifier

In this thesis two different classifiers are used to perform the initial identification of the objects parts. Both classifier are trained from a set of data consisting of two different classes C_1 and C_2

Logistic Regression

The first classifier used is *Logistic Regression*. This classifier uses the sigmoid function to compute a probability of input x being of Class C_1. The function to compute the probability $p(C_1|x)$ is defined as

$$p(C_1|x) = \sigma(\mathbf{w^T}\boldsymbol{\phi}(x))$$

where $\boldsymbol{\phi}(x)$ is the feature vector of input x, $\mathbf{w}$ the weight vector and σ(x) the sigmoid function. The weight vector is learned using training data. The training data consists of an input x and the expected outcome t for $p(C_1|x)$. This implies that t is set to 1 when x is element of C_1. Otherwise t is set to 0. There is no closed-form solution to compute the weight vector. Therefore the weight vector is computed in several iterations using an update formula in every iteration. The formulas are from the book of Christopher M. Bishop (see [Bis06, p.207ff]).

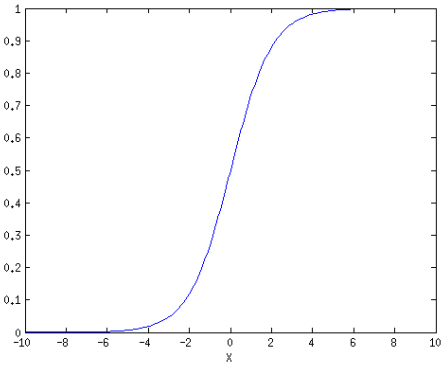


Figure 4.5: Sigmoid-Function

To compute the update of the weight factors the Newton-Raphson update formula is used.

$$w^{new} = w^{old} - \mathbf{H^{-1}}\nabla E(w)$$

The gradient is given by

$$\nabla E(w) = \sum_{n=1}^{N}(y_n - t_n)\phi_n + (\lambda w^{old}) = \mathbf{\Phi^T}(\mathbf{y} - \mathbf{t}) + (\lambda \mathbf{w^{old}})$$

where $\mathbf{\Phi}$ is a $N \times M$ matrix with N being the number of samples and M the number of features. The n^{th} row is given by ϕ_n^T. The elements of the row-vector $\mathbf{y}$ consists of the results of the classifier for all input x from the training data using the last computed weight vector. The elements of the row vector $\mathbf{t}$ consists of the expected outcome t from the training data.

The Hessian is given by

$$\mathbf{H} = \sum_{n=1}^{N} y_n(1-y_n)\phi_n\phi_n^T + (\lambda I) = \mathbf{\Phi^T R \Phi} + (\lambda \mathbf{I})$$

where the matrix $\mathbf{R}$ is a N x N diagonal Matrix with elements $R_{nn} = y_n(1-y_n)$. The factor λ is used for regularisation and punishes large weight values. The greater lambda the smaller the weight values will be. This will help to avoid over-fitting. The weight values are starting with an initial value. The Newton-Raphson update formula is used until the weight values are converging.

Generative Classifier

The second classifier is a *generative classifier*. The classifier learns from a training set with two different classes. For both classes a multivariate normal distribution $p(x|C_1)$ and $p(x|C_2)$ is computed. This can be done with the following formula:

$$p(x|C) = \frac{1}{\sqrt{(2\pi)^{\frac{3}{2}}|\Sigma|^{\frac{1}{2}}}} exp(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)$$

where Σ is the covariance matrix of the training data and $|\Sigma|$ the determinant of it. μ is the mean of the training data. The mean and the covariance are computed with

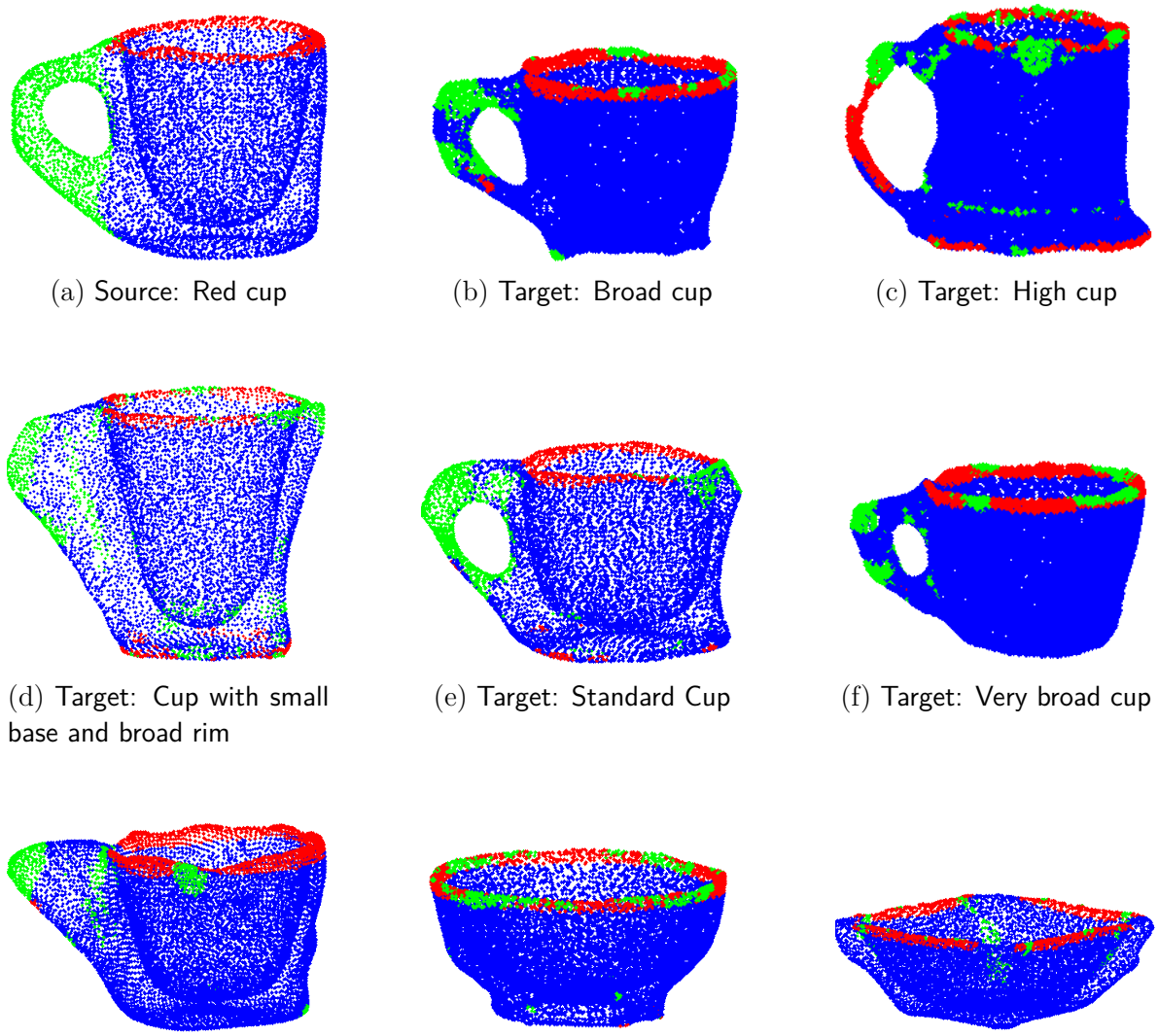
$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$
$$\Sigma = \frac{1}{n}\sum_{i=1}^{n}(x_i-\mu)(x_i-\mu)^T$$

With the two multivariate normal distributions and the a priory probabilities $p(C_1)$ and $p(C_2)$ the classifier can compute the probability of the input x being a specific object part. This is done with the Bayes' rule:

$$p(C_1|x) = \frac{p(x|C_1)p(C_1)}{p(x|C_1)p(C_1) + p(x|C_2)p(C_2)}$$

4.1.3 Results

The *Logistic Regression* was used to identify the rim points as it produced the best results. To identify the handle points the generative classifier was used instead. Both classifiers were trained using the red cup. The red cup was hand-labelled to avoid initial errors. As shown in figure 4.6 both classifiers are producing false classifications. For example in figure 4.6 c) some rim points are labelled as handle points. On the other side a lot of handle points are getting labelled as rim points. Also the edges of the bottom are getting labelled as rim points. Therefore the classifiers are only useful for a rough labelling. The warping is still needed to improve the results. This results will suffice for the alignment of the objects.



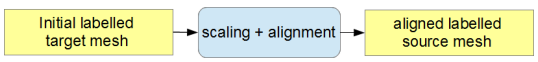
(a) Source: Red cup (b) Target: Broad cup (c) Target: High cup

(d) Target: Cup with small base and broad rim (e) Target: Standard Cup (f) Target: Very broad cup

(g) Target: Very small cup (h) Target: Standard bowl (i) Target: Square bowl

Figure 4.6: Example: Initially classification of our objects using the red cup

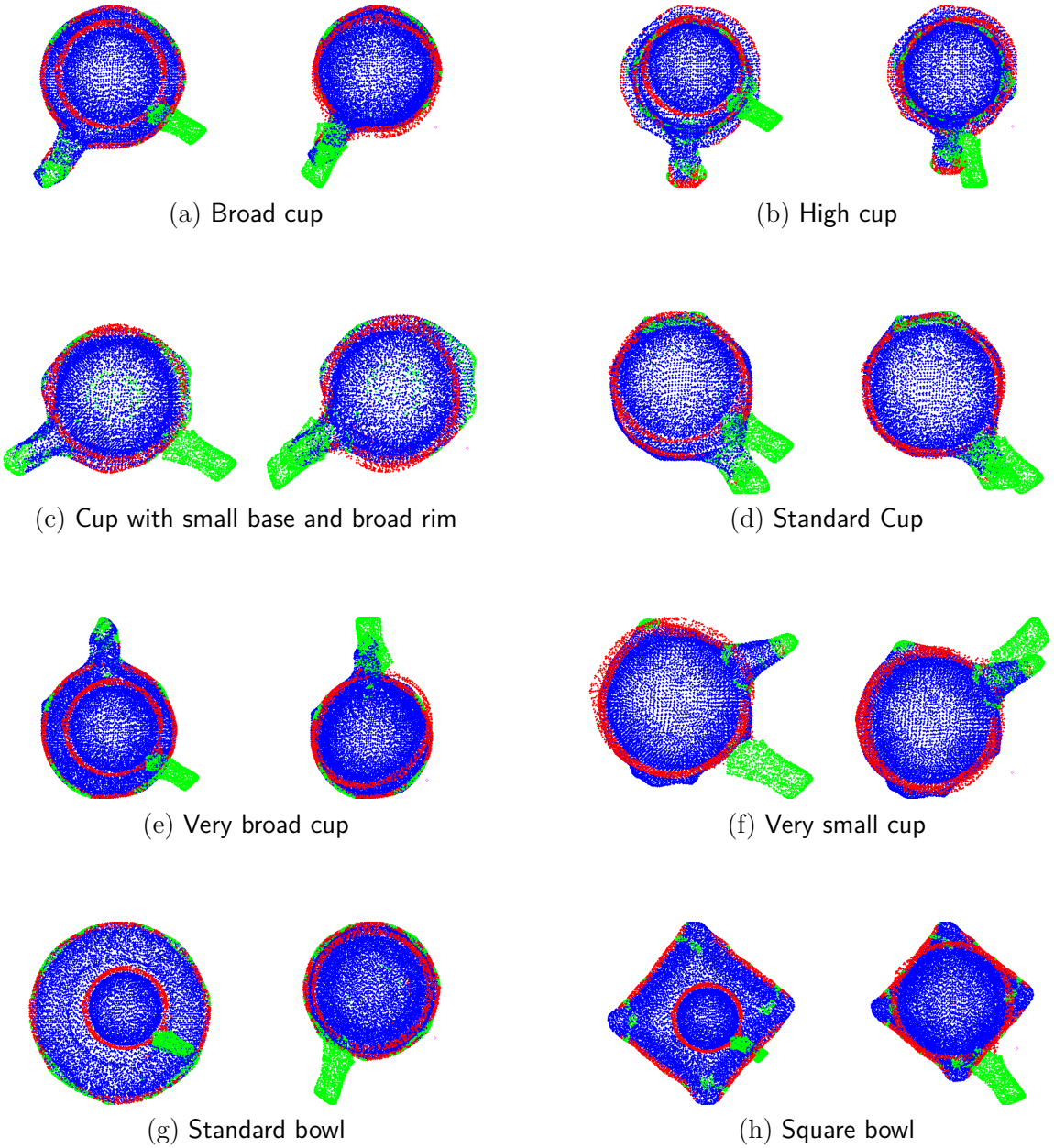
4.2 Scaling and alignment



By using the initial classification of the corresponding object parts, the centre of all parts can be computed by using the labelled vertices. This is achieved by computing the mean point of the labelled vertices from an object part. First both containers will be aligned. The centre of the source container is simply shifted to the centre of the target container. After this the objects have to be aligned according to their rims. A rotation can be computed to compensate for the angle between both rim centre. If the objects have handles they can also be aligned by computing another rotation.

After the aligning the next step is to scale the source mesh to the same size as the target mesh. A scaling factor of a mesh can be achieved by computing the standard deviation of the vertices in x, y and z direction. By computing both scaling factors for both meshes the source mesh can be scaled to the size of the target mesh.

Even though the target mesh was only roughly labelled, the alignment still produces good results. In figure 4.7 our example objects are aligned with the source cup using the alignment and the scaling as described above:



(a) Broad cup

(b) High cup

(c) Cup with small base and broad rim

(d) Standard Cup

(e) Very broad cup

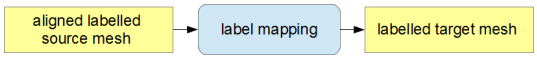
(f) Very small cup

(g) Standard bowl

(h) Square bowl

Figure 4.7: Example: Alignment of our objects with the red up

4.3 Label mapping



After the alignment is done the correspondences between the vertices can finally be computed. This is achieved by finding the corresponding source vertex for every target vertex. The label of the corresponding source vertex than can be mapped to the target vertex. The alignment as described above is good enough

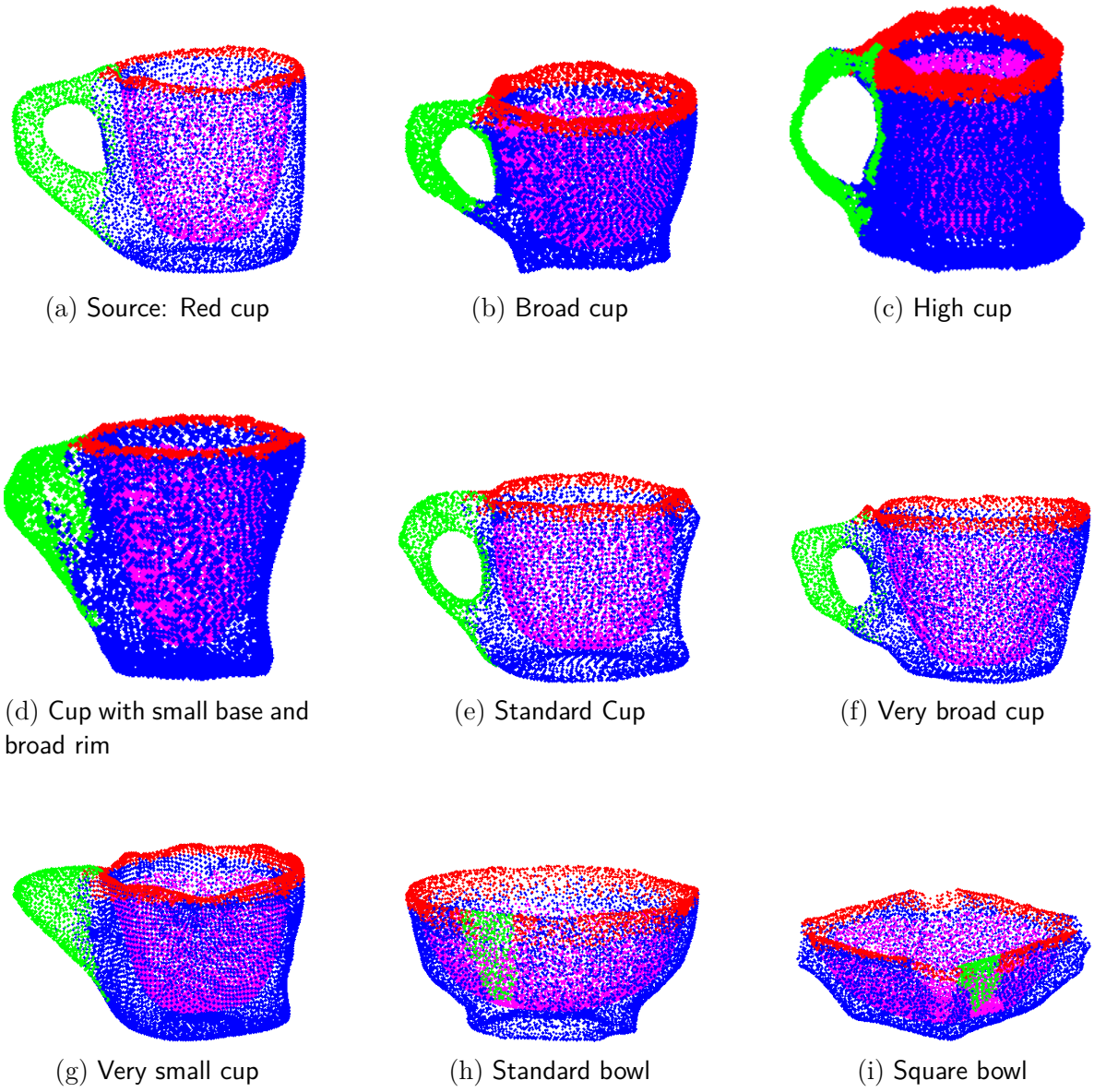
to find the correspondences by using the vertex normals and minimizing the distances between the vertices. First the set of source points with a normal vector u(s) oriented at most an angle δ away from v(t) is defined. The function u(s) delivers the normal vector of a source point s while v(t) delivers the normal vector of a target point t. The sets S and T represents our source(S) and target meshes(T). The set is defined as

$$S(t)_\delta = \{s \in S | u(s) \cdot \boldsymbol{v}(t) > cos(\delta\}$$

With this set the *forward correspondence* can be computed. The function delivers the closest vertex $s \in S(t)_\delta$ for a target vertex t. The function is defined as

$$C_f(t) = \text{argmin}_{s \in S(t)_\delta} ||t - s||_2$$

The forward correspondence can be used to find the corresponding source vertex s for every target vertex t and therefore compute a label mapping. For the following examples in figure 4.8 the parameter δ was set to $\frac{\pi}{2}$. As the source object the red cup was used.



(a) Source: Red cup (b) Broad cup (c) High cup

(d) Cup with small base and broad rim (e) Standard Cup (f) Very broad cup

(g) Very small cup (h) Standard bowl (i) Square bowl

Figure 4.8: Example: Finished labelling of our objects using the red cup for the warping

object	real height	computed height	total error	relative error
Red cup	8.70cm	7.60cm	1.10cm	12.64%
Broad cup	8.80cm	8.20cm	0.60cm	6.82%
High cup	10.20cm	9.10cm	1.10cm	10.78%
Cup with small base and broad rim	10.0cm	9.20cm	0.80cm	8.00%
Standard cup	8.30cm	7.00cm	1.30cm	15.66%
Very broad cup	8.90cm	7.80cm	1.10cm	12.36%
Very small cup	6.70cm	5.60cm	1.10cm	16.42%
Standard bowl	8.00cm	7.40cm	0.60cm	7.50%
Square bowl	6.50cm	5.80cm	0.70cm	10.77%
average			0.93cm	11.22%

Table 4.1: Measurements of the object height

object	real radius	computed radius	total error	relative error
Red cup	4.00cm	3.70cm	0.30cm	7.50%
Broad cup	5.00cm	4.90cm	0.10cm	2.00%
High cup	4.50cm	4,30cm	0.20cm	4.44%
Cup with small base and broad rim	4.50cm	3.90cm	0.60cm	13.33%
Standard cup	4.35cm	4.00cm	0.35cm	8.05%
Very broad cup	5.60cm	5.10cm	0.50cm	8.93%
Very small cup	3.85cm	3.50cm	0.35cm	9.09%
Standard bowl	8.5cm	8.00cm	0.50cm	5.88%
Square bowl	7.50cm	7.80cm	0.30cm	4.00%
average			0.36cm	7.02%

Table 4.2: Measurements of the object radius

object	real volume	computed volume	error	relative error
Red cup	265ml	153ml	112ml	42.26%
Broad cup	452ml	297ml	155ml	34.29%
High cup	435ml	313ml	122ml	28.05%
Cup with small base and broad rim	296ml	162ml	132ml	44.59%
Standard cup	324ml	167ml	157ml	48.46%
Very broad cup	450ml	280ml	170ml	37.78%
Very small cup	170ml	90ml	80ml	47.06%
Standard bowl	1038ml	532ml	506ml	48.75%
Square bowl	680ml	505ml	175ml	25.74%
average			179ml	39.66%

Table 4.3: Measurements of the object volume

At the end of this chapter the measurements of the real object are compared with the created models. They can be found in table 4.1 to 4.3. There is an average relative error of 11.22% for the computation of the height. This is mainly because the mean of the rim points is used for the computation. For some objects the warping algorithm produces a broad rim when mapping the labels. The reason for this is that one source vertex can correspond to more than one target vertex. This is for example the case when the target mesh has more vertices than the source shape as shown in figure 4.8 c).

There are also some errors in the radius computation resulting in an average relative error of 7.02%. The biggest errors is made for the volume computation with an average relative error of 39.66%. There are two reasons for this cause. First the labelling of the source cup is underestimating the container in our case. This also results in an underestimation for the target results. But the main cause for this is the implicit surface function used to create the inside of the object. There is no point data of the inside from the 3D-scanner so the Kernel Regression can not fit the function to get a better estimation of the inner surface. So there is still room for improvement on the model generation.

Of course only a simple warping algorithm was used for the label mapping. By using a better warping algorithm the results can improve. As the warping is part of a pipeline it can easily be change for a better algorithm. For our purposes the results will suffice.

5 Learning to pour

The last chapters were about the model generation and how to identify the object parts of an unknown mesh using the warping approach. At the end of this process, the unknown mesh is fully labelled and all object parts are identified. Given the meshes of the object parts, the high level features now can be computed. In this chapter, we will explain how the warped meshes can be used to learn pouring. First, the high level features will be introduced which are used for the learning approach. Afterwards, two simulated experiments will demonstrate how the high level features can be used for learning. The first experiment focuses on learning if a particle would increase the volume of the fluid in a container. The second one is to learn if the container will start to pour when it is tilted.

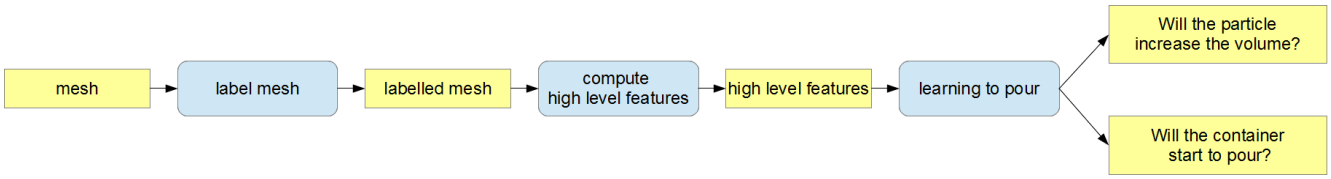


Figure 5.1: Pouring Pipeline

5.1 High Level Features

The high level features depend on the shape of an object and the object parts. Therefore the labelled meshes can be used to compute these high level features. The centre of the rim can be obtained by computing the mean of the rim vertices. By computing the mean distance to this rim centre a radius can be obtained. Also the height of an object can be obtained by computing the vertical distance of the rim centre to the lowest vertex of the mesh. To get the volume of the container the container vertices can be used to create a convex hull. This convex hull can be used to compute the volume. The high level features used in this thesis are defined below and are all dimensionless.

Relative Distance

The first feature is the relative horizontal distance(d) of a fluid particle(p) to the rim centre(c). The relative horizontal distance is divided by the radius(r) to get a dimensionless value. This feature is well-suited for circular rims but can also be used for other rim shapes.

$$f_1(d, r) = \frac{d}{r}$$

Furthermore, the following applies to f_1:

$$f_1 = \begin{cases} = 1 & \text{particle above the rim} \\ < 1 & \text{particle above the opening} \\ > 1 & \text{particle not above the object} \end{cases}$$

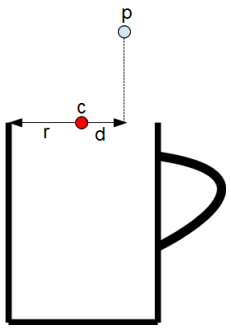


Figure 5.2: *Relative Distance* feature

Relative Height

The second feature is the relative vertical distance(d) of a fluid particle to the rim centre. The distance d is divided by the height(h) which results in a dimensionless value.

$$f_2(d, h) = \frac{d}{h}$$

Furthermore, the following applies to f_2:

$$f_2 = \begin{cases} > 0 & \text{particle above the object} \\ = 0 & \text{particle has the same height as the rim centre} \\ < 0 & \text{particle below the rim centre} \\ < -1 & \text{particle below the object} \end{cases}$$

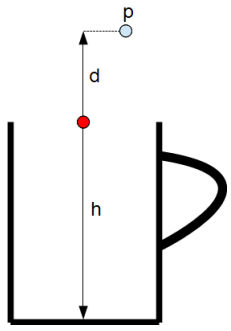


Figure 5.3: *Relative Height* feature

Fluid Volume

The third feature is the fluid volume of the particles in the container(v_c) relative to the container volume(v_f). This features tells how filled the object is.

$$f_3(v_f, v_c) = \frac{v_f}{v_c}$$

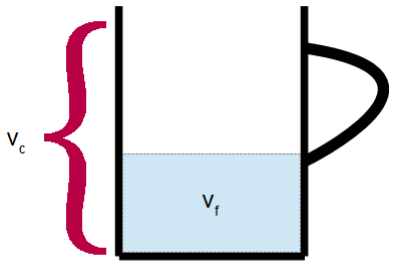


Figure 5.4: *Fluid Volume* feature

Fluid Volume tilted Container

The fourth feature is the fluid volume of the particles in the container above the lip point(v_a) relative to the container volume(v_c). The lowest rim vertex is used as the lip point.

$$f_4(v_a, v_c) = \frac{v_a}{v_c}$$

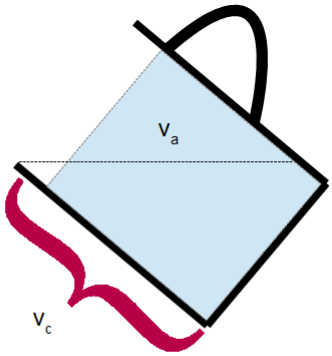


Figure 5.5: *Fluid Volume tilted Container* feature

5.2 Experiments using the bullet simulator

The experiments were run using a simulator based on the bullet physic engine library. The standard bullet engine itself does not provide support for fluid simulations. However, there is a project which integrates FLUIDS V.2 into bullet which allows for fluid simulation (see [bul13]). FLUIDS V.2 is an implementation of the Smoothed Particle Hydrodynamics(SPH) method. The created meshes are imported into the simulator using a convex hull decomposition. Thus, the fluids can interact physically with the meshes. The SPH was originally used in astrophysics to simulate star formations and interactions of different galaxies (see [LXSR12]). As a result the fluids in FLUIDS V.2 are compressible. This leads to the problem that the density changes with the amount of fluids in the container. Therefore, the fluid volume can not be computed by just counting the particles. Instead a local volume around every particle needs to be computed in order to approximate the global volume. To get the local volume a local density around every particle is estimated. This is done by finding the amount of neighbours of the particles in a specific distance. A normal distribution is used for this as shown in figure 5.6 for a 2D-case.

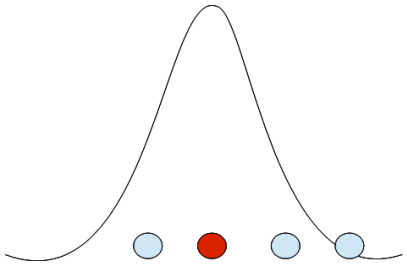


Figure 5.6: Volume approximation using a normal distribution

For a particle p in the container the local density $d_l(p)$ is computed with

$$d_l(p) = \sum_{i=1}^{n} inRange(p, p_n) * F(||p - p_n||_2)$$

where n is the number of particles inside the container and p_n the n^{th} particle. $F(x)$ is the normal distribution as shown above with the parameters $\mu = 0$ and σ.

The function $inRange(p, p_n)$ is defined as

$$inRange(p, p_n) = \begin{cases} = 0 & ||p - p_n||_2 \geq 3 * \sigma \\ = 1 & ||p - p_n||_2 < 3 * \sigma \end{cases}$$

Assuming that a particle has the mass of 1 the fluid volume can be computed with the following formula:

$$\nu = \frac{1}{\sum_{i=1}^{n} d_l(p_n)}$$

In this thesis σ was set to 0.01 as it provided the best results for the volume estimation.
The simulator is used to track the position of the fluid particle during the experiment. The learning and the evaluation of the results are performed in Matlab.

Experiment 1: Will the particle increase the volume?

The first experiment is to learn a classifier for which particles will increase the volume of the container. For this experiment, only the first three features are used as the fourth feature would be a constant 0. This experiment shows if the features are suitable for learning where to pour in order to pour into the container.

Set-Up: One object is imported into the simulator at a fixed position. After the object is imported, fluid particles appear at random positions around the object. The starting and the end position of a fluid particle is saved. The fluid particles that result in a filled object container after the simulation are not deleted.

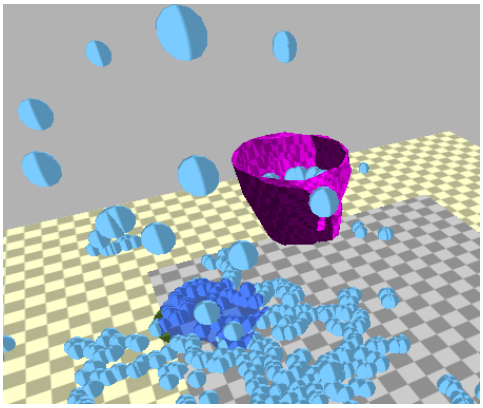
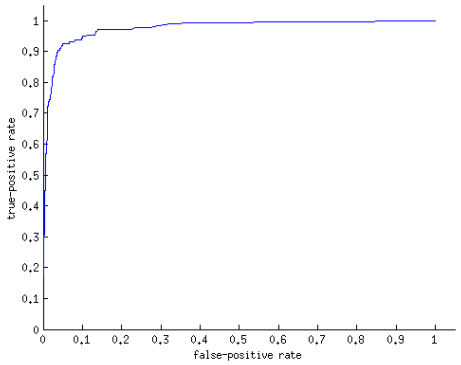


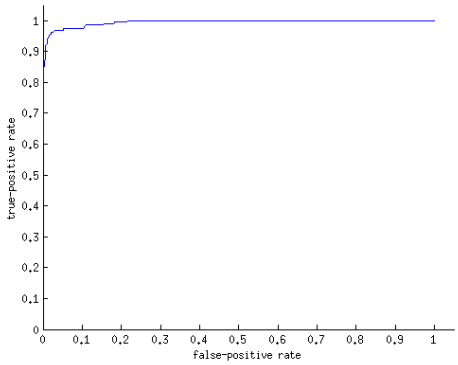
Figure 5.7: Experiment 1

Analysis: After the simulation, the trajectories of the fluid particles are evaluated to see which particles increased the volume of the fluid in the object container. With this evaluated information a classifier can be trained to learn if a particle will end up in the object container. The logistic regression is used as the classifier for this experiment. After the classifier is learned it was used on other objects within the same experiment to see how good the features generalise.

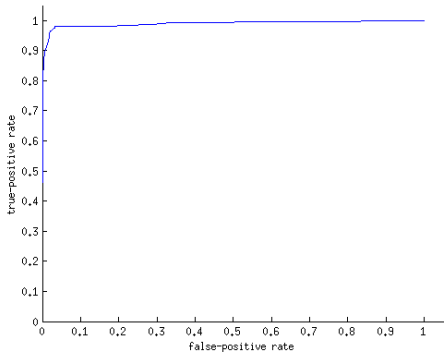
Results: As the classifier provides a probability for every input a threshold has to be used to decide whether the fluid particle will increase the volume or not. To compare the results, a Receiver Operating Characteristic curve (ROC-curve) is computed. The ROC-curve is created by iterating over different thresholds and plotting the false-positive and true-positive rate. The true-positive rate tells us how many particles are correctly classified as increasing the volume of the container. On the other hand the false-positve rate tells us how many particles are classified as increasing the volume but wouldn't. Therefore it is important to minimize the false-positive rate because this will result in pouring fluids outside of the object. It is also good to maximize the true-positive rate but not as important as minimizing the false-positive rate as the consequences are not as severe. In figure 5.8 the ROC-curves of the first experiments for our objects are shown. The classifier was trained with the red cup.



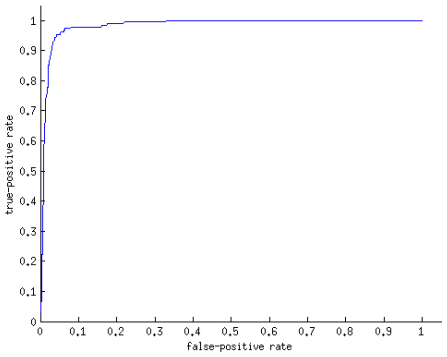
(a) source: Red cup



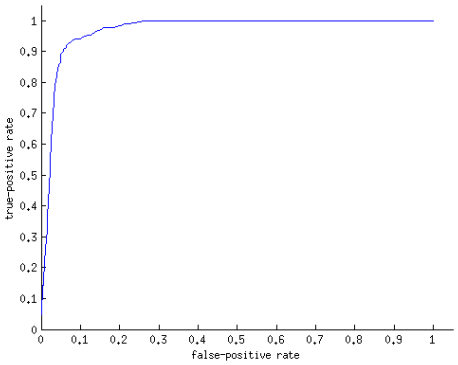
(b) Broad cup



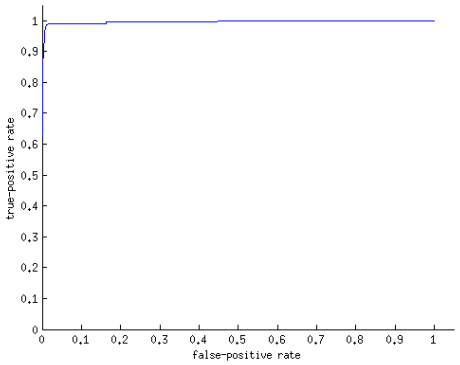
(c) High cup



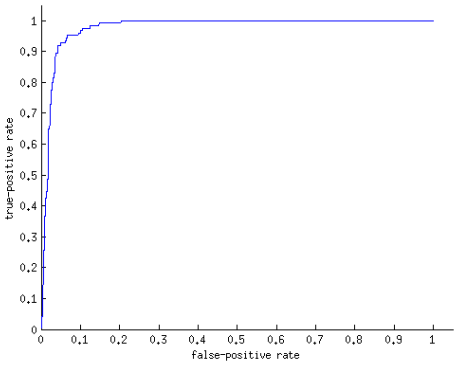
(d) Cup with small base and broad rim



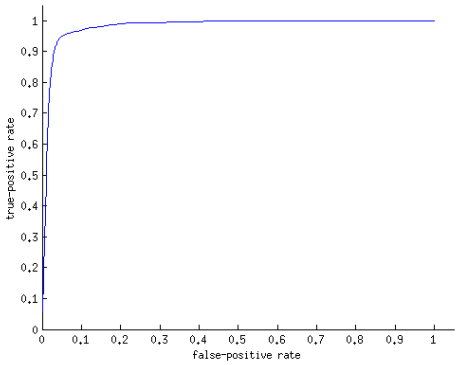
(e) Standard Cup



(f) Very broad cup



(g) Very small cup



(h) All cups

Figure 5.8: Experiment: Will the fluid particle increase the volume of the container?

The classifier obtained good results when tested on the individual cups. This is even the case when used on all cups at once. When the threshold is chosen too high, the particles dropping near the rim will be classified as *not increasing the volume of the container*. Although it would be the case. Therefore the cups with a bigger radius tend to have better results. There are some errors though caused by the simulator. Most of them occur at the beginning of the simulation when the container is still empty. Only when there is a certain amount of fluid particles, the individual fluid particle will behave correctly as a fluid. The first particles of the simulation tend to bounce out of the container after falling into it. This happened not

as often for the bigger cups. This effect does not occur any more after some fluid particles are inside the container.

For the pouring task it is most important not to spill the fluid. The experiment shows that it is possible to choose a threshold that reduces the false-positive rate near zero while obtaining a high true-positive rate for all objects. This leads to the conclusion that the trained classifier using the first three high level features generalises well between objects.

Experiment 2: Will the container start to pour?

The second experiment is about learning when the container will start to pour. Here the fourth feature will be used and tested. The first two features are not defined for this situation and the third one would be a constant.

Set-Up: One object is imported into the simulator at a fixed position. After the importing the object container is filled with fluids. The object than was tilted slowly by a specific angle into a fixed direction. The particle trajectories are saved for the starting position and the end position. This simulation is repeated for different angles.

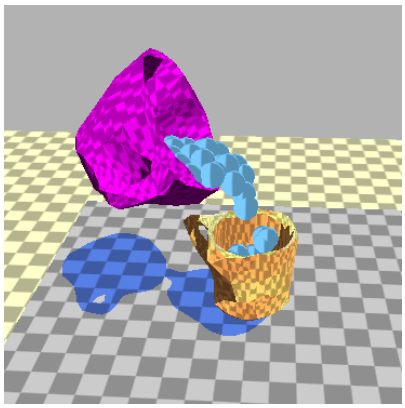
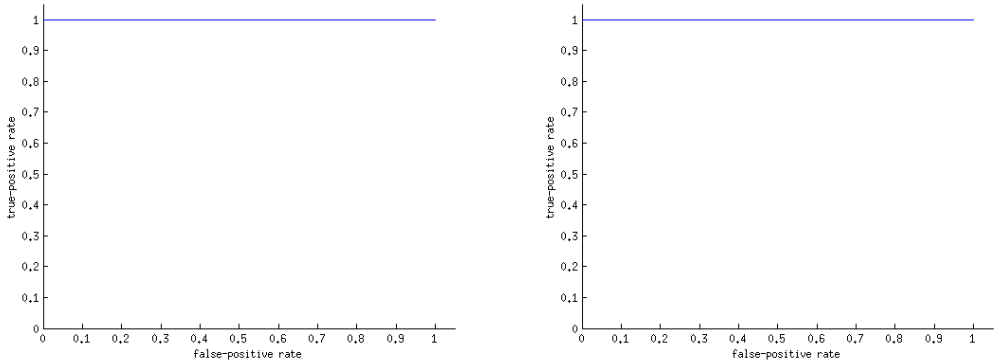


Figure 5.9: Experiment 2

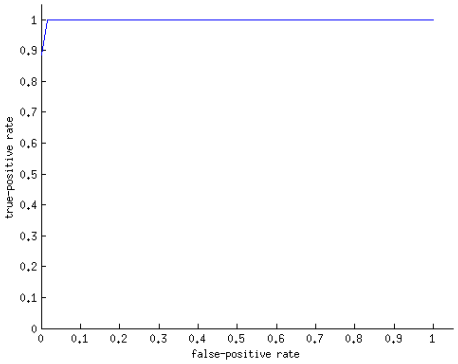
Analysis: After the simulation the trajectories of the fluid particles are evaluated to see when the object container starts to pour. The starting position was used to compute the fourth feature. The object and the fluid trajectory at the starting position are rotated by a specific degree in Matlab into the fixed direction used in the experiment. By using the lowest rim vertex as the lip point the fourth feature can be computed to predict if the object container will start to pour. Similar to the first experiment the logistic regression is used as the classifier.

Results: Similar to the first experiment, the ROC-curves show the results. The classifier was trained on the red cup and applied to the rest of the objects.

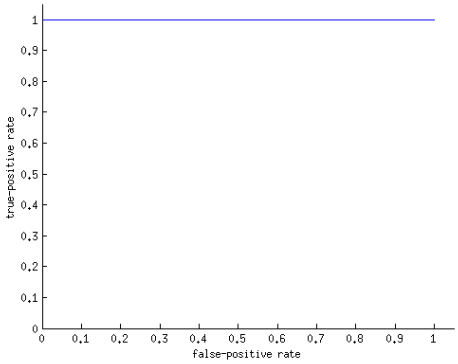


(a) source: Red cup

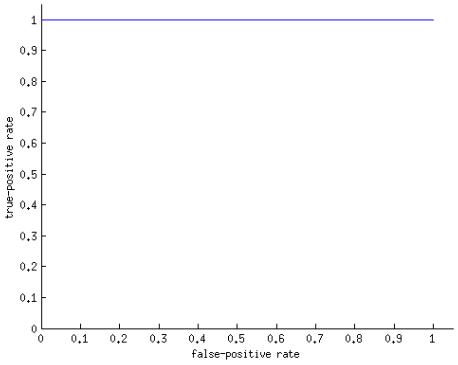
(b) Broad cup



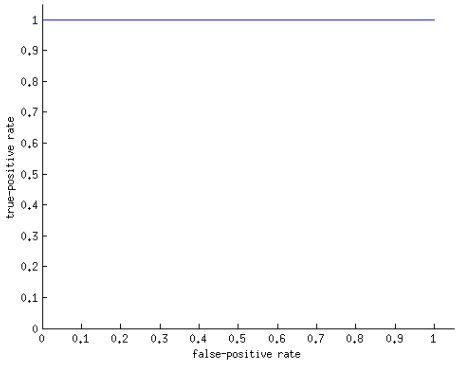
(c) High cup



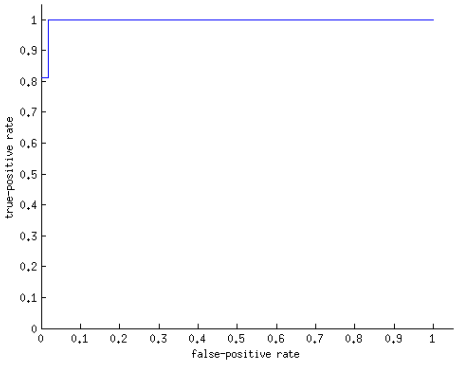
(d) Cup with small base and broad rim



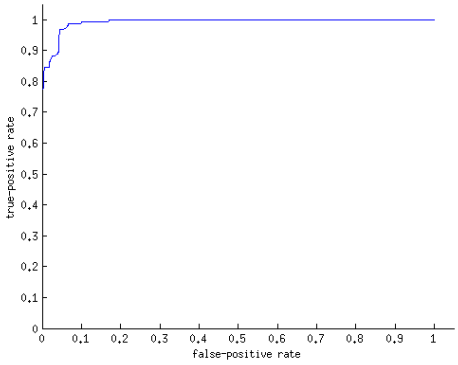
(e) Standard Cup



(f) Very broad cup



(g) Very small cup



(h) All cups

Figure 5.10: Experiment: Will the container start to pour?

For the second experiment the trained classifier obtains very good results for every individual objects. When using all objects at once the results are only slightly worse. The fourth feature is suitable to train a classifier for this task. The classifier also generalises well between the objects.

The results of the two experiments show that it is possible to learn with the help of the high level features. As a proof of concept we now use the classifiers to define a policy for a pouring movement. By using the high level features the policy should generalise to different objects.

The set-up of the simulation are two objects. The first one is filled with fluid and will be used to pour the fluid into the second object. The first one will follow a trajectory to a specific point p. After reaching this point the object will be tilted for a specific angle δ and a specific duration t in order to pour the fluids into the second object. After this the object will be tilted back to the starting position.

The parameters p, δ and t are computed using the high level features. The target is to pour 20 fluid particles into the second object. So a function is needed to predict the amount of particles pouring outside of the object. This function will use the amount of particles above the lip point and the duration of tilting as an input. For an estimation for this function the kernel regression from chapter can be used again. The following function was attained by sampling the amount of particles pouring out using the simulator for random durations and random angles for the red cup:

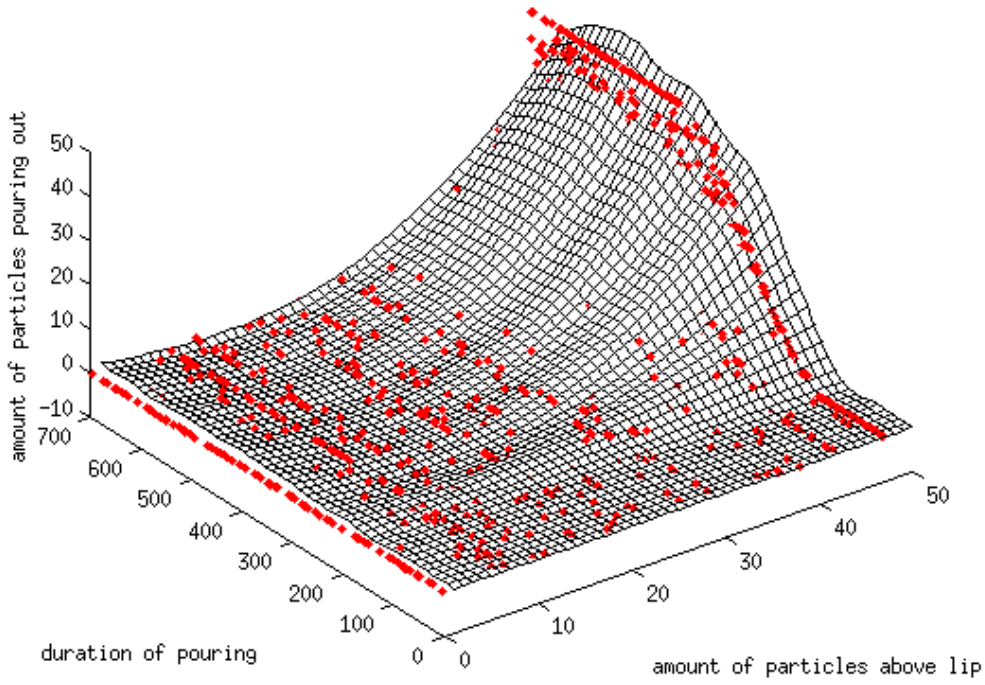


Figure 6.1: Estimation for amount of particles pouring out

By using this function the parameters δ and t for a specific amount of particles pouring out can be computed for every object. After getting δ the lip point can be used to obtain the parameter p. This is done by finding a position where the fluid particles will increase the volume of the second object. Here the classifier

from experiment 1 is used.
The simulation was executed as described above using the red cup to learn the estimation function for the amount of pouring particles. The result is shown in table 6.1.

object	δ	duration	target	outcome	error	inside	outside
red cup	63°	5.08s	20	18	2	18	0
broad cup	90°	4.12s	20	16	4	16	0
high cup	68°	9.47s	20	11	9	11	0
cup with small base and broad rim	61°	4.5s	20	19	1	19	0
standard cup	60°	4.12s	20	16	4	16	0
very broad cup	71.5°	4.12s	20	19	1	19	0
very small cup	49°	7.13s	20	21	1	21	0
average	66.07°	5.51s	20	17.14	3	17.14	0

Table 6.1: Results for the proof of concept

By using the high level features and an estimation function a policy could be defined. The position p was obtained by using the classifier from the first experiment. When pouring from this position no fluid particle was spilled for all objects as shown in table 6.1. The angle δ and the duration t was obtained using the estimation function. This was done by iterating over the function and finding the value closest to 20 for every object. Although there were some errors for the actual amount of poured particles the result was still near the desired value of 20. In addition the amount of poured particles only slightly exceeded the desired value for the very small cup. In some situations pouring a larger amount than expected can result into overflowing fluid. Our defined policy performed the worse on the high cup. This is due the height of the cup as the fourth feature does not compensate for the height. When tilting an object the fluid particles will start to fill out the space until the lip point is reached. When the space is used up, the rest of the particles will start to pour out of the object. The volume of the space depends on the height of the object.

The goal of this proof of concept was to show that a policy can be defined using the high level features. Looking at the results this goal was achieved. For example by using a policy search the results can be improved.

7 Conclusion

First we started with the model generation. For this a turning table and a *Kinect*-camera was used as a 3D-scanner to get the first point clouds. Only the outside of the objects was scanned so the inside of the object had to be computed. Hence an implicit surface function was learned using a *Kernel Regression* with a *Gaussian kernel*. The learned function was used in the *Marching Cubes* algorithm to create the surface mesh and the surface normals. In this process the inside of the objects was created. The surface meshes were smoothed afterwards using the *Laplacian smoothing* to reduce the errors caused by the 3D-Scanner.

After the model generation, the warping was used to identify object parts from unknown objects using known objects. A modified version of the warping algorithm from Hillenbrand was used. In the first step, both objects were aligned using the corresponding object parts. As the object parts from the source shape were known and the object parts of the target shape were still unknown a rough identification of the object parts was performed using local classifiers. Afterwards both meshes were aligned according to the estimated object parts. Using this method, a good alignment between the objects was achieved.

The next step of the warping was the point mapping. The vertices of the source shape were mapped to the corresponding vertices at the target shape by looking at the vertex normals and the distances between the vertices. At the end of the mapping the object parts could be identified and achieved a better result than just using the classifiers.

Given labelled meshes the high level features were introduced and tested in two different experiments. The high level features depends on the corresponding object parts and were computed using the labelled vertices. The first experiment focused on learning a classifier which determines if a fluid particle will increase the volume of an object container. The classifier was learned using the red cup and applied to other objects. The classifier produced good results for the red cup itself and also produced good results for the other objects. The results showed us that these three features generalise well between objects.

The second experiment was to learn a classifier which tells us if an object container will start to pour. The *Fluid Volume tilted Container* feature was used for this experiment. Similar to the first experiment the classifier was learned on the red cup and used for the other objects. This feature also produced good results for the generalisation.

At the end a proof of concept was introduced. The goal was to define a policy using the high level features. The parameters for the policy were learned from the high level features of the objects. The defined policy was used in the simulator. The goal was to pour 20 fluid particles from one container into another. To predict how many fluid particles will pour out of a container an estimation function for this was learned using a *Kernel Regression*. The training data was obtained using the red cup. After this function was learned the angle δ and the pouring duration t could be computed using the high level features for all objects. The results of the simulation showed us that the goal to define a policy can be achieved but there are still errors in the estimation of the poured particles. The high and the very broad cup had the biggest outliers but were also the most different cups from the red cup. During the simulation no fluid particle was poured outside of the object. This experiment shows that it is possible to define a policy using the high level features.

The whole pipeline process going from the model generation to the pouring produced good results for our objects. To use high level features and warp objects to identify object parts is a feasible approach to generalize between objects without relearning the task for every object. This was proven by our experi-

ments. This was also shown by implementing a proof of concept. But there is still room for improvement in the pipeline. For example using a better fluid simulation or improve the model generation will lead to better results at the end of the pipeline.

8 Future work

As a future work a policy search algorithm can be used to find better policies. The proof of concept only shows that is is possible to define a policy using the high level features. Hopefully the results can be improved using a policy search. It is planned to later do this on a real robot. Using a real robot the results of this thesis can be validated as we were only using a simulator. Doing the experiments on a real robot is more difficult as there is no information available for every fluid particle. The simulator can be used though to train and than execute the movement on the real robot.

Also some more high level features can be defined to improve the results for other objects. For example the feature *Relative Distance* is well-suited for circular rims which applies for our cups. This feature also worked for the square bowl but the results were worse.

Besides the results of this thesis can be used and tested for other tasks to see how well they apply there. Therefore other high level features needs to be defined. The constraint is that the task has to depend on the shape of an object to use the warping and the high level features.

Bibliography

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[bul13] Bullet-fluids project. https://github.com/rtrius/Bullet-FLUIDS/, October 2013.

[DTG11] Stanimir Dragiev, Marc Toussaint, and Michael Gienger. Gaussian process implicit surfaces for shape estimation and grasping. Robotics and Automation (ICRA), 2011 IEEE International Conference on, May 2011.

[Hil10] Ulrich Hillenbrand. Non-parametric 3d shape warping. International Conference on Pattern Recognition – ICPR 2010, 2010.

[KDGB11] Lars Kunze, Mihai Emanuel Dolha, Emitza Guzman, and Michael Beetz. Simulation-based temporal projection of everyday robot object manipulation. AAMAS '11 The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 1, 2011.

[KOP10] J. Kober, E. Oztop, and J. Peters. Reinforcement learning to adjust robot movements to new situations. In *Proceedings of Robotics: Science and Systems (R:SS)*, 2010.

[LC87] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. Computer Graphics, Volume 21, Number 4, July 1987.

[LXSR12] S.J. Lind, R. Xu, P.K. Stansby, and B.D. Rogers. Incompressible smoothed particle hydrodynamics for free-surface flows: A generalised diffusion-based algorithm for stability and validations for impulsive flows and propagating waves. Journal of Computational Physics 231 (2012) 1499–1523, 2012.

[RJT13] Leonel Rozo, Pablo Jim´enez, and Carme Torras. Force-based robot learning of pouring skills using parametric hidden markov models. IEEE-RAS Intl Workshop on Robot Motion and Control (RoMoCo), 2013.

[TNUW11] Minija Tamosiunaite, Bojan Nemec, Aleš Ude, and Florentin Wörgötter. Learning to pour with a robot arm combining goal and shape learning for dynamic movement primitives. Robotics and Autonomous Systems 59 (2011) page 910–922, 2011.

[ZPKD05] R. Zöllner, M. Pardowitz, S. Knoop, and R. Dillmann. Towards cognitive robots: Building hierarchical task representations of manipulations from human demonstration. Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on, 2005.