

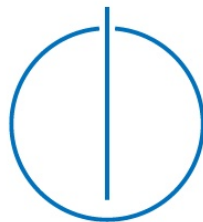
**Technische Universität
München**

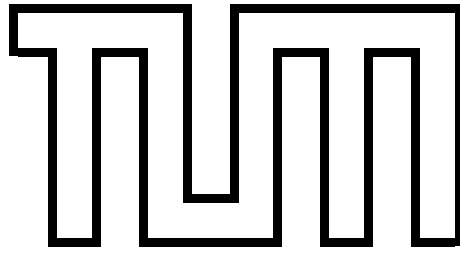
Fakultät für Informatik

Master's Thesis in Biomedical Computing

Natural marker extraction and learning for binocular images and
6D pose estimation

Mahdi Saleh, B.Sc.





**Technische Universität
München**

Fakultät für Informatik

Master's Thesis in Biomedical Computing

Natural marker extraction and learning for binocular images and
6D pose estimation

Extraktion von Referenzpunkten zur adaptiven
6D-Posenberechnung für Zweikamerasysteme

Author: Mahdi Saleh, B.Sc.

Supervisor: Prof. Dr. Nassir Navab

Advisor #1: Benjamin Busam, M.Sc.

Advisor #2: Federico Tombari, Ph.D.

Submission: 15.03.2017

I assure the single handed composition of this master's thesis only supported by declared resources.

München, 15.03.2017

(Mahdi Saleh, B.Sc.)

Abstract

In modern computer vision there are various tasks such as object detection or tracking which are beholden to keypoint detection. Recently there is a trend on learning 2D keypoints/features which have shown to be a good alternative to handcrafted features [1],[2],[3]. We aim to learn stereo keypoints of different poses for objects and use them as surface markers as input for other tasks such as 6-DOF pose estimation using marker point cloud registration. To find relevant keypoints we use a pool of different keypoints which we triangulate to find the 3D position of all extracted keypoints. Subsequently, we use different criteria to find the most reliable keypoints which can be used as selected markers and project them back on 2D object images to create the dataset. We train patches of different keypoints with a Deep Learning network to distinguish marker patches from non-marker patches. And finally we test the whole pipeline for the task of 3D pose estimation, which has 6 DOF.

Inhaltsangabe

Aus modernen Computer Vision Publikationen lässt sich ein Trend hin zu learning-basierten 2D Keypoint-Extraktionsalgorithmen beobachten, die eine potente Alternative zu regelbasierten Ansätzen bieten. Die vorliegende Arbeit beschäftigt sich mit der automatischen Extraktion von Keypoints für Objekte in verschiedenen Posen. Diese Referenzpunkten fungieren als Input für einen optischen Tracking-Algorithmus, der mittels einer 3D-Registrierung simultaner Stereobilder 6D-Objektposen berechnet. Um relevante Markerpunkte bestimmen zu können, wird zunächst ein Vielzahl potentieller 3D Punkte trianguliert, welche anschließend abhängig von verschiedenen Kriterien derart gefiltert werden, dass die verlässlichsten übrig bleiben. Diese werden danach auf ihre ursprünglichen Bilder projiziert um einen vollumpfänglichen Datensatz zu erhalten, der dazu dient, ein Deep Learning Netz zu trainieren. Das anhand von lokalen Patches trainierte Netz wird schließlich eingesetzt, um relevante von nicht-relevanten Bereichen zu unterscheiden und Marker automatisch zu extrahieren. Am Ende wird die gesamte Pipeline im Hinblick auf adaptive 6D-Posenberechnung für Zweikamerasysteme getestet.

Contents

1	Introduction	1
1.1	Challenges	3
1.2	Structure of thesis	3
2	Related literature	5
2.1	Handcrafted features	6
2.2	Feature extraction using machine learning	7
2.3	Pose estimation from monocular or depth images	9
2.4	Pose estimation from binocular images	11
3	Object video dataset creation	14
3.1	Synthetic dataset	16
3.2	Real dataset	18
4	Keypoint point cloud creation and processing	24
4.1	Camera model and calibration	25
4.2	Stereo calibration and epipolar geometry	27
4.3	2D keypoint extraction	28
4.3.1	Data preparation	29
4.3.2	Feature extraction pipeline	29
4.4	Building keypoint point cloud	31
4.4.1	Triangulation	31
4.5	Finding keypoint clusters	33
4.5.1	Mean-shift	34
5	Natural marker criteria and selection	38
5.1	Keypoint reprojection	39
5.1.1	Rendering ideas	39
5.2	Keypoint scoring	44
5.2.1	Distinctness	45
5.2.2	Repeatability	46
5.2.3	Distribution	47

CONTENTS

6	Marker learning and pose estimation	50
6.1	Marker dataset creation	51
6.2	CNN architecture	53
6.2.1	Introduction to CNN	53
6.2.2	Proposed Architecture	56
6.2.3	Training	56
6.3	Pose estimation procedure	57
7	Evaluation and Conclusion	63
7.1	Marker extraction	64
7.2	Marker training	66
7.3	Pose estimation	67
7.4	Conclusion	69
7.5	Future work	70
	List of figures	71
	List of Tables	71
	Bibliography	71

Chapter 1

Introduction

Tracking has always been an important computer vision task, with several applications including medical applications and traffic control. In a conventional tracking problem, an object is detected and finding the location of this object in the 2D frame sequence given by a video is the goal. Recently with the the boost of mobile computer vision and augmented reality solutions, which is already in many everyday applications, 3D object tracking plays a significant role. With such systems that are visually augmenting the objects and images, inaccuracies in tracking or pose estimation are rather unsightly. These application are also usually versatile and demand tracking of different objects, some of which may have evolved geometries. A lot of task are handled using active and passive markers [4], [5], [6], [7]. Markers are usually chosen to be detected easily and are proven to work more precisely than markerless solutions. However marker-based tracking is not generalizable and may not work in high occlusions. Model based object tracking using 3D CAD models is also another approach in computer vision and in use in industrial applications and augmented reality.

3D sensors are recently used in order to achieve a smoother object tracking [8],[9]. Microsoft Kinect and Intel Realsense as well as Google Tango are some of the depth sensors that can deliver high frame rate depth channel information alongside with RGB images [10]. Mostly model based approaches are used by this means to fit the 3D models to the point clouds and finding the best matching translation and rotation to estimate the 3D pose of the object[11], [12]. Stereo vision systems also provide potential 3D information with a pair of images. Using Stereo matching of pixels based on epipolar geometry, followed by 3D triangulation, one can get to a rough 3D point cloud of the object[13]. The quality of disparity or depth is relatively of higher precision[14], but the task of stereo matching is considered as an expensive and ill-posed problem. Handcrafted feature extractors such as SIFT[15], SURF[16] or BRISK[17] can also be used for stereo triangulation, but for large viewpoint changes these sparse features are not reliable keypoints for an object.

There are already systems working in realtime and accurately using markers and stereo images like [4]. In their work some stickers are stuck to an object's surface and their positions are found in a stereo infrared setup using stereo triangulation and epipolar geometry, they calculate the 3D location of markers and define a reference point cloud. Then acquiring any binocular frame, the system finds the evaluation point cloud and registers this to the reference point cloud using every minimization approach which can be refined with an iterative closest points method [18]. The transformation to the reference point cloud is counted as the estimated 3D pose of the object. If more than enough markers are attached to the object, the system can work in partial occlusion as well.

Here we want to develop another approach like the afformentioned one that has the

potential of being ubiquitous and can learn objects with different models and textures. Instead of attaching markers to an object or attaching stickers on the surface, we try to extract a set of natural markers all over the object. As an input, we can use a video sequence of a single object in different 6-DOF poses and learn some of the selected keypoints. There has been some work due to learn 2D or 3D keypoint/features using conventional machine learning approaches and recently with the power that deep learning brings, keypoints from different objects and its characteristic can also be learned by machines. Following these ideas, we train several salient keypoints/markers on the stereo images and then evaluate the output in the task of 3D pose estimation with several objects.

1.1 Challenges

Of course finding keypoints for every objects is a very challenging task for several reasons. As we need at least 3 keypoints in both stereo views that are not in a line in order to perform 3D registration, acquiring enough keypoints from all different angles of the object is something that is challenging if an object is plainly colored on a side or has few textured region. On the other side repeating textures with the same color are also a pitfall for this task. A keypoint somewhere in or around a repeating texture has a potential of detection in other false areas. Symmetric objects (both geometrically or texture-wise) are also some problems in pose estimation. If an object is symmetric like a fully white glass, finding the pose of the object is impossible even for a human being. Adding to this a good pose estimation algorithm should work for textured as well as texture-less objects. In this work we want to focus on different objects with varying texture density. Also here we want to concentrate more on the task of best keypoint extraction so that the output of our machine learning section are keypoints rather than the direct 3D pose. although we feed these keypoints to a pose estimator to see how this works for different objects with different shapes and texture.

1.2 Structure of thesis

In chapter 2 we give a brief overview of different related approaches from handcrafted features used in pose estimation to machine learning that helps for feature extraction. We mention some work in the field of 3D pose estimation from 3D model based approaches to stereo vision solutions and finally give an overview on the literature on machine learning and CNNs to find best keypoints and pose estimations . In chapter 3, we mention the

details of creating two datasets of stereo video from objects with their ground truth pose, one from rendering 3D object models and one with a multi-camera setup of real objects and finding their pose with a marker-based system. Starting our algorithm from chapter 4, we create a pool of different keypoints and match them in a binocular view, different handcrafted features are used for every frame and they are matched based on a fixed feature descriptor. Later on, we get the 3D location of extracted points using 3D triangulation and epipolar geometry and process the point cloud in order to get to a candidate set of keypoints. Subsequently, we look at different criteria to find the best fitting markers and decide how to model and score them with our candidate set in chapter 6. In chapter 7, we use the marker patches with their labels and non-marker patches and try to classify them with a deep learning structure and use the output of the network to find a evaluation point cloud to estimate the 6-DOF pose of the object. Finally in chapter 8, we evaluate our pipeline and keypoints using one of the datasets created for the task of 3D pose estimation and conclude our work and strategy regarding other works and the state of the art.

Chapter 2

Related literature

In this chapter we review the literature related to our work. We start from 2D handcrafted features and their use in pose estimation, we mention why these methods are now not longer used directly and we introduce more enhanced pose estimation approaches from 2D images. Subsequently, we look at the methods using 3D information such as object models and depth images. Afterwards, we introduce stereo vision techniques and how they have been applied to the task of pose estimation. Finally we study different machine learning approach to learn keypoints, features and 3D pose estimation.

2.1 Handcrafted features

Early works in computer vision demanded finding interest points and matching them for various tasks including detection and tracking. There are a lot of papers to extract features as edge or corner detectors using Gaussian or Laplacian operators. **Harris** corner detector is one of the first attempts to extract interest points for detecting patterns [19] which is still used for corner detection, but not considered as a real keypoint. Harris detector can differentiate flat, edge or corner regions by scoring intensity shiftings in derivatives of the image. Harris is invariant to 2D rotation but is not invariant to scale which often is needed for real applications. One of the early applications is the Viola-Jones implementation on pedestrian detection[20].

The problem of Human detection initiated another feature extraction method with Histogram of Oriented Gradients(**HOG**) [21].HOG features work by calculating the gradients and forming the histogram of gradient directions and creating a feature descriptor using block randomization. Such a descriptors brings in a lot of useful information for matching keypoints and tracking. This work started the challenge to develop more ingenious features with more informative descriptors.

Scale-invariant feature transform or **SIFT** is considered as a breakthrough in feature extraction and even computer vision. [15]. Lowe solved the scale invariance problem by creating octaves of images at different scales and applying a difference of Guassians (DoG) to the layers to find keypoints. The rather slow process leads to locations of keypoints, their orientations and a descriptor. The paper presents a whole pipeline with its own matching method, which is robust but has problems with severe view-port changes. SIFT initially is created for industry applications and therefore requires license fees upon usage [22]. The relatively low speed of SIFT made it inapplicable for realtime problems, so that a lot of follow-up methods aimed to improve SIFT. Speeded up robust features **SURF** [16] is one of the successors of SIFT which has nearly the same robustness but with a

significant speed up.

FAST (Features from Accelerated Segment Test)[23] is another technique with not only speed improvements but claims to be more robust to viewpoint changes and is used in various computer vision tasks. It looks at a circular neighborhood of the pixel and figures out if it is a corner [22]. FAST has a real-time implementation that helps tracking applications like mobile augmented reality and realtime robust corner detection. Other feature descriptors like **BRISK** [17] and **FREAK** [24] also try to find scalable binary features and visual perceptions to outperform SIFT, SURF or FAST methods, but are less suitable for generic applications with regard to robustness or speed. Another different feature extractor used in computer vision is Maximally Stable Extremal Regions (MSER) [66], which unlike the other aforementioned methods looks for blobs or stable connected component regions in the image. MSER has been applied to object recognition and stereo matching tasks based on blob detection.

The application of keypoints and feature extractors are broad from image stitching to object tracking and recognition. But for the task of 3D pose estimation these methods usually fail with big view-port changes and full view object perception. To tackle this in pose estimation applications, **LineMOD** is developed [25] which uses depth map to find templates for each object for the task of 3D pose estimation and outperforms other handcrafted features in the task. LineMOD calculates surface normals and tries to assign gradient features on these normals per object. In this regards LineMOD trains every object with its features and is a start for machine learning based keypoint extraction.

2.2 Feature extraction using machine learning

Machine learning has a short history of influence on computer vision. Before the prevalence of deep neural networks, some other machine learning methods were used to extract keypoints per object. Various architectures are proposed such as Viola-Jones [20] or Boosting from weak learners using AdaBoost [26] used for face detection. Moreover, in the field of keypoint or feature learning One of the first works is the from Gabriella Csurka et al [27], in which they do visual characterization based of bags of keypoints. The classification of the feature vectors are done in general different contents using Naive Bayes[28] and Support Vector Machines (SVM) [29]. Later Leppetit et al. proposed Randomized trees for keypoint recognition for objects [30]. Initial template images are used for training keypoints and later these are used as a baseline for classification. These keypoints and approach is used for object recognition rather than 3D pose estimation.

Another concept here is creating or evaluating keypoints by machine learning rather than learning handcrafted keypoints on the objects. Mian et al. investigates repeatability and quality of keypoints with a pipeline of detecting keypoints from partial views of 3D objects and finding local features for these keypoints [31]. The features are listed for the object after forming a principle component analysis (PCA) subspace and pinned to different locations. These features are used for matching with a query of images for an object retrieval task. Another effort is using random trees for keypoint recognition by Mustafa Ozuysal et. al[32]. They use patches around keypoints and a naive Bayesian framework for classification. The method using 20 FERNs give similar results and is sometimes better than SIFT for the finding of homographies and 3D objects [33]. As the number of FERN structures grow, the recognition rate decreases significantly and can be used for 3D scene annotations and visual Simultaneous localization and mapping (SLAM). SLAM is a task in robotics which involves computing the scene maps and localizing an agent simultaneously [34].

Another use of machine learning algorithms for the task of object detection and pose estimation is using Exemplar-SVM. Malisiewicz et. al designed a classification routine of extracting HOG features and supervised training using Exemplar-SVM [35]. Another work is done for detecting chairs [36] where they use exemplars of 3D objects with different models and views to match with the 2D images of chairs. They use a large CAD dataset of different chairs and align query images based on confidence rate of different parts. Later research is mainly under the influence of the capabilities of deep neural networks or CNNs. With the high adjustability of learning range of deep architectures, different information can be learned and extracted from images. In the last few years there has been some work done in finding keypoints using CNNs or extracting features from a layer of the deep neural networks.

Richardson et. al used CNNs to detect interest points in images [37]. Here they prove that hand engineered feature extractors can be implemented using convolutions, a process that can be similar to a learning scheme. These trained features are matched in different test set areas like office spaces or conference rooms and then compared with baseline methods like FAST, SIFT or SURF. The test errors show that the method outperforms handcrafted features in 3 out of 4 cases. A work from Salti et. al focuses on descriptor based 3D Keypoint detection [38]. The authors investigate 3D keypoint description and to find a measurement to consider distinctiveness, rather than taking care of geometry information and repeatability. The pipeline selects keypoints in different RGB-D views of an object and eliminates non-distinctive keypoints from the 3D point cloud and trains these keypoints using random forest algorithm. The outcome out-performs baseline saliency approach

regarding error and speed.

Another inspiring achievement is the paper from Verdie et. on TILDE - a temporally invariant learned DETector [2]. This work uses images of a single view in drastic illumination changes and finds handcrafted features such as SIFT in a set images from a fixed view. The positions which are frequently detected as keypoints in different conditions are chosen for training. These keypoint samples are trained using three different algorithms, a piece-wise linear regressor, regression trees and convolutional neural networks. The results show a great improvement to the state of the art methods such as FAST or SIFT. TILDE is used for 2D image matching and stitching and results are reported for many different datasets.

2.3 Pose estimation from monocular or depth images

3D pose estimation with 6 degrees of freedom (DOF) is a rather advanced task in computer vision and there are only a few papers in this regard using 2D image information only. Most of the research with images is restricted to viewpoint classification and camera angle estimation. Early work is using handcrafted features such as SIFT or SURF and matching keypoints in 3D space to find the object pose [39, 40]. These handcrafted features do not guaranty full 360° pose change around the objects and usually fail for the 6-DOF pose estimation. A research from Chunhui Gu et. al [41] shows an approach based on mixture of templates for viewpoint classification. HOG templates are calculated per object in different views and the features are trained with SVM in supervised, semi-supervised and unsupervised manner and the proposed pipeline improves the state-of-art in object viewpoint classification.

Pose estimation can be a costly task and sometimes using 3D information such as depth channel data or 3D CAD models, voxel or point cloud models of an object helps a lot for matching and registering objects. There are a lot of ways for matching and registration of point clouds, iterative closest points (ICP), for instance, is one of the most widely used algorithms [18]. The algorithm matches a target point cloud with a reference point cloud in an iterative fashion while the distance between them minimizes. The algorithm directly gives the translation and rotation to the reference which is needed. ICP is very dependent on initial solution and has problems matching sparse point clouds. Some other variations of ICP try to solve this problem [42, 43].

Choi et. al for example use ICP for 3D pose estimation of daily objects with RGB-D cameras [44]. The method shows exploiting color information alongside with depth

information enhances the quality of 3D pose estimations. The evaluation is performed for heavily cluttered scenes and without having accurate 3D CAD models. The later work from Hinterstoisser creates LineMOD features for pose estimation. The paper concentrates on model based training and pose estimation of textureless objects using depth RGBD images in heavily cluttered scenes [45]. They initially use LineMOD feature templates from 3D model objects and store these templates as object training. For testing, they use Microsoft Kinect data and color information to match with objects, this helps to detect the object class and later on estimate the 3D pose of the object.

When we have 2.5D (2D + depth image) or 3D information, we can also extract features and descriptors from the geometry of the 3D surface. Handcrafted features in this area usually use histograms to match 3D structures. The problem with 3D data is the difficulty to predict the amount of noise with the current sensors which makes it very challenging to use depth images. Intelligent approaches by means of machine learning for 3D geometry descriptors can also help to filter noise and improve the resolution limit. Recently a paper on learning local geometric descriptors has been published by Zeng et al. [46]. They have scanned areas using depth sensors from different angles and found the matching correspondences in two views. They convert them to a volumetric representation and train a 3D-CNN on these to extract descriptors. These geometric descriptors can be used in many tasks such as 3D reconstruction. Volumetric representation consists of voxels in 3D space. A voxel a unit similar to pixels in 2D and states the presence of object in the representation point.

Another inspiring work that reflects the task of 3D pose estimation besides feature descriptors is the recent work of Wohlhart et al. [1]. The authors use RGBD images from different view angles of an object as shown in Figure 2.1. The method uses 2 layers of convolutions and pooling and inputs the whole set of object images and outputs the 3D pose with a mapping of descriptors. The descriptors can be compared using Euclidean distance rather than manifold-based distances, and can thus be searched using nearest neighbor search which eases the task of pose estimation. The approach is compared to LineMOD and HOG features and outperforms them accuracy-wise when using RGB, RGBD or depth images alone.

However the publication shows an innovative architecture, but the results and accuracy can have improvements specially for industrial application such as robotic grasping, that demand higher accuracy and more a robust pipelines. In this regard, stereo methods can be useful as they prove to perform more accurately in many application.

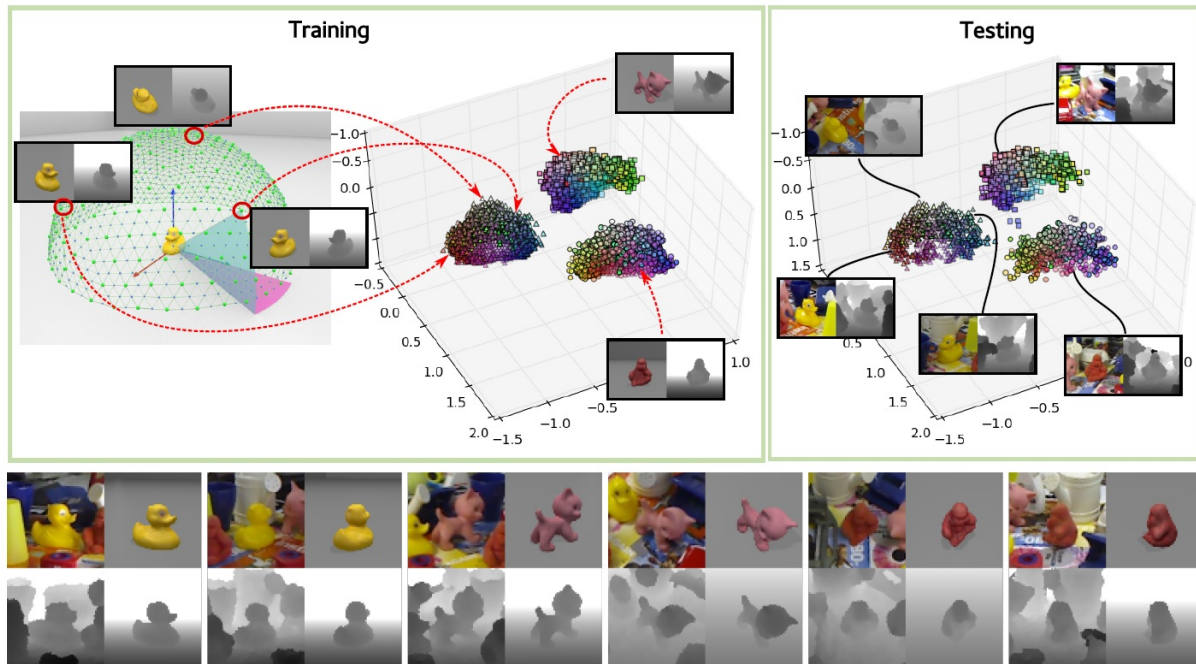


Figure 2.1: Sample object models and their depth maps from different angles in bottom, and their trained descriptor and object space in top-right. Later some images with complicated background are used to test the performance as shown in top-left. Figure from Wohlhart and Lepetit [1]

2.4 Pose estimation from binocular images

Another approach to gain 3D information about the objects and surroundings by using stereo vision with two cameras: binocular vision. Using mathematical concepts of camera models and epipolar geometry one can get to nominate mutual correspondences in both images [13]. With the same concept the task of stereo matching can be defined, and one can get a depth map from disparities of correspondent pixels in two images. Some more details on epipolar geometry and the correspondence problem are described in chapter 3. What makes stereo visions used less in computer vision systems, especially ones demanding a lot of 3D information like pose estimation, is the performance cost, complexity and the problem of task to be ill posed. Although most of the concentration in stereo vision system in research is stereo-matching and finding depth maps, as existing numerous implementations on KITTI[47] and middlebury dataset [48], but still there are a few sample scenarios where the use of stereo vision systems is beneficial.

A paper of Lecun et al. studies different methods for pose invariant object recognition in stereo images [49]. They have a dataset of 15 toy models in different azimuth elevation and lighting condition for binocular views. They assess different methods such as nearest

neighbors, SVM and convolution networks at a time, when they were not that popular. The latter gave the best results in object recognition. Regarding the CNNs, they feed the left and the right images in 8 feature maps of the first convolution layer.

Later Shao et al. apply stereo vision as robotic sensors to detect obstacles [50]. While for the detection of object models and depth maps from stereo is rather expensive and difficult, handcrafted features help a lot and are used for stereo correspondence matching. They use a simple approach of having SIFT keypoints detected, match the stereo views and by the means of epipolar geometry, the 3D location of objects are calculated. The system is designed to detect, track and find the distance to the obstacles while an acceptable working quality.

Some other papers discuss similar approaches with some improved functionality. Hsu et al. [51], for instance, tries to extract SIFT keypoints in both stereo images and match them together with a database of keypoints from different objects to achieve a 3D object recognition. The authors have tried to design a real-time recognition pipeline and add some other extracted information such as object size and distance.

Automotive is an active research area where stereo vision systems and stereo matching is used widely. While there is a lot of work to create accurate depth maps from road images [47], Barrois et al. peruses 3D pose estimation of vehicles using stereo vision [52]. They calculate optical flow and separate static objects and moving ones in 4D sparse-time. Then they use a cuboid model as a car and iterative closest point algorithm to match the points. They also try polar distance instead of Euclidean distance and show improved results in tracking and 3D pose estimation of vehicles. In the best configuration, the yaw error is around 3 degrees and the distance error is around 0.1 meter.

One of the powers of stereo vision systems is their potential in accuracy . Two works of Busam and Esposito et al. demonstrates a system used for medial applications such as movement therapy and needle biopsy surgery [4],[53]. They attach reflective 2D markers on an object's surface and by using several of them and by detecting these markers in a pair of stereo images a 3D point cloud of markers is reconstructed. These markers can be matched or registered with a reference point cloud using an energy minimization approach with potential ICP refinements and can be used to get a very accurate and real-time 6-DOF pose of the object even in case of occlusion, Figure 2.2. This thesis project gets its motivation from this project and is intended to build the marker point cloud without attaching physical markers on the object. A system running on an improved version of the algorithm [4] (FRAMOS Optical Tracking System) is also used in section 3.2 to create ground truth for the pose estimation.

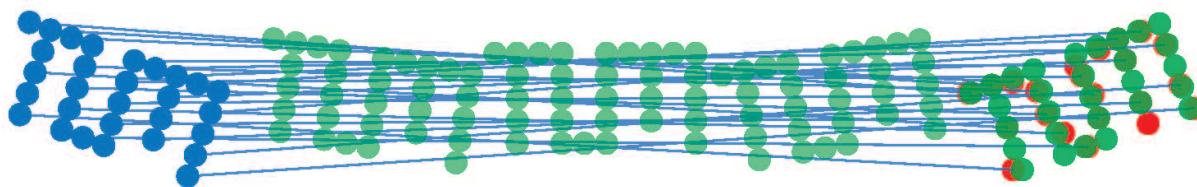


Figure 2.2: A network of markers from right is matched to a reference point cloud on the left. The system estimates the pose when enough matching markers are found. The illustration of the algorithm by Busam et al. [4].

One last example that has inspired this work for the correspondence problem is the research done by Simo-Serra et al. [3]. This project is using deep networks to learn point descriptors. The learning is done on patches of keypoints without using handcrafted features such as SIFT. The dataset consists of pairs of images of a rigid stereo setup from different angles. The patches are then fed to a Siamese network shown in figure 2.3 that compares the patches and forms a descriptor that can be integrated in test and training stage using L2 distance. The patches of negative and positive matches are inserted and the network later is used as a descriptor generator. The research outperforms conventional matching methods such as handcrafted features.

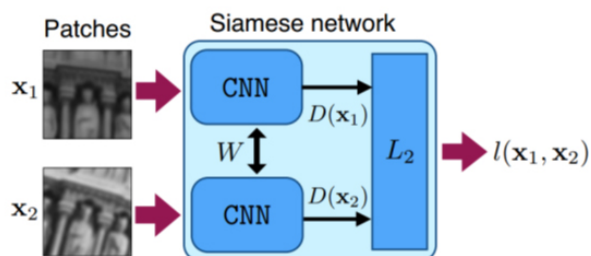


Figure 2.3: A Siamese network architecture for matching corresponding patches, employing two CNNs with identical parameters and treating the CNN outputs as patch descriptors. Image taken from Simo-Serra et al. [3]

Chapter 3

Object video dataset creation

There are various of publicly available dataset for object detection and recognition but for the task of pose estimation there are some available like the Linköping University CVL Object Pose Estimation Database [54] or TUM Hinterstoisser ACCV dataset [45]. The former has a set of 16 objects rendered from different rotation angles with steps of 5 degrees along two directional axes. The dataset provides monocular images of segmented objects in black background or with crowded backgrounds. In the latter, Hinterstoisser provides 15 objects with their 3D models and captured RGBD images with different poses mainly for showcasing the proposed LineMOD algorithm for pose estimation [55]. The dataset also provides some object models and voxelized point clouds. Another dataset example is Rutgers APC RGB-D dataset [56] which contains images with depth channel output for 25 objects captured by a robotic arm and a mounted Kinect sensor. The dataset is created for object detection and pose estimation mainly for warehouse pick-and-place applications. All these datasets, however, cannot be used for our task simply because they just provide monocular images and the 3D pose is estimated usually by sequential information or by including some prior 3D information of the objects.

There are only a few publicly available stereo vision datasets of objects and they usually do not provide 3D pose tags. The Rigid Pose dataset [57] contains stereo images of objects rendered in a sequence of 6-DOF poses with a stereo video background. As illustrated in figure 3.1, it contains six objects, four of which are highly textured and ideal for the task of keypoint extraction based on textures of the object's surface. However, there are two problems with the dataset that make it incompatible with our solution. Firstly, the viewpoints and the poses of the camera are not uniformly distributed around the object and they are mostly from the object's frontal view. Secondly, dataset also provides rather few frames for the task of keypoint estimation and learning which demands a longer video with preferably variable view angles.



Figure 3.1: 3D models of Rigid Pose Dataset, Figure from Pauwels et al. [57]

In order to tackle our task of marker extraction and having a large enough input set

for learning phase, we need to create our own image set with different textured objects and have them captured from ideally different well distributed 3D poses. In this section, we explain how we created our dataset as a ground truth for the supervised learning hereafter. Firstly, we introduce the synthetic dataset with 5 objects rendered from a sequence of poses. Then in the second section we mention a way to obtain a real object dataset with ground truth. We also append depth map images to provide a possibility to compare with RGB-D based techniques on the same dataset.

3.1 Synthetic dataset

For creating an appropriate stereo dataset for object marker extraction and learning we should select some objects with different texturing outward. These objects should have also different 3D geometry and complexity as shown in figure 3.2. We use a free online sources, Turbosquid [58], to download some 3D models namely a dwarf, a can of beets and the earth . We also create some 3D models and assign textures to them like a pillow and a box. Although model and 3D information is not going to be used as an input for the pipeline, these objects are selected from basic primitive shapes (sphere, cube, cylinder) to some more complex structures.

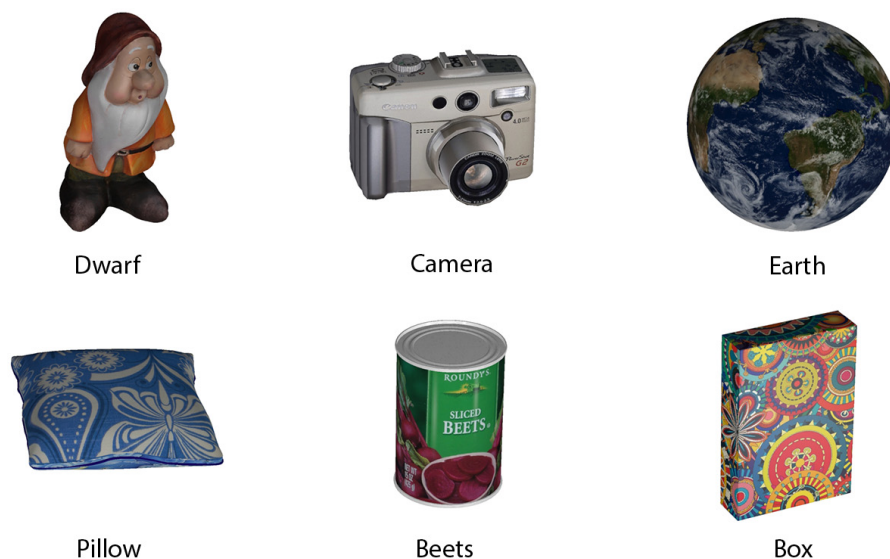


Figure 3.2: 3D models for our synthetic dataset

The objects are imported and created in 3Ds Max 2016 [59]. A pair of cameras also is added to the view scene of the objects to capture the images. The left camera is placed

at (0,0,0) and the right camera is located aside with a baseline of 70 mm. The focal length for the cameras are 70mm. Both cameras look in X direction and the objects move and rotate locally around their pivot point. The pivot point of the objects is their 3D centroid as shown in figure 3.3. In order to transform the objects frame by frame, we need to create a sequence of random 6-DOF poses.

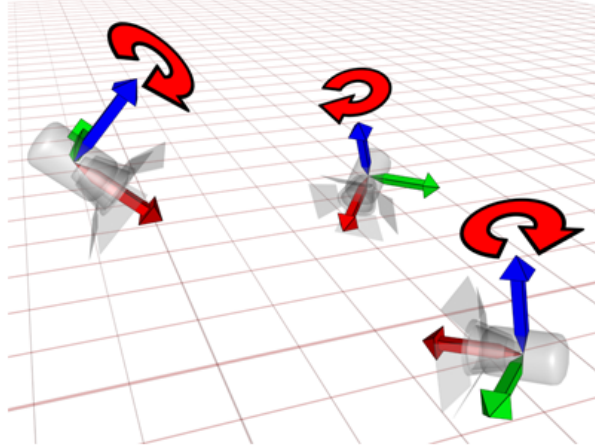


Figure 3.3: Rotation in 3Ds max is applied to pivot point in object's local, Figure from Autodesk knowledge network [60]

We generate vectors of translation and rotation quaternions for all the frames that are rendered. Each pose vector consist of a translation vector and a quaternion vector $[Q, T]$. The translation elements are in millimeter scale and have some restriction such that the objects are fully visible in the rendered scene later on. The rotation vector is a quaternion with four elements $[Qw, Qx, Qy, Qz]$. The quaternion notation is widely used in computer graphics and game industry and avoids gimbal lock problems of Euler systems [61]. It also provides more effiecent computation and storage [62]. Throughout the thesis, we are using quaternions to represent rotations. At some steps we might convert them to Euler angles or rotation matrices for easier calculation or specific representations. The poses are given by:

$$P = [Tx, Ty, Tz, Qw, Qx, Qy, Qz]$$

subject to:

$$0 < Tx < 200mm$$

$$-100mm < Ty < 100mm$$

$$-100mm < Tz < 100mm$$

1000 vectors of translation together with quaternions are created using a MATLAB script.

The quaternions are normalized, concatenated with translation vectors and written to a file for processing of dataset creation and pose ground truth checks. 3Ds Max used an exclusive scripting language to program and automate the scene and workspace. A Maxscript program is created to automate the process of reading poses for each object, doing the transformation and rendering the stereo camera views as well as saving them to the hard disk. In a loop of the script for a selected object, the pose information is obtained from the ground truth file, the object initially resets the translation and rotation to zero and the frame's local rotation and translation is applied to the particular object. Afterwards, new background image is loaded and the scene is rendered from both the left and right camera. The rendered images are saved to image file sequences.

The images of the background are chosen from home interior views of the MIT-CSAIL database of objects and scenes [63], which is the same image for the left and right image. Some augmentations like mirroring is performed, in order to the images for our 1,000 batch dataset. The images are rendered in 1024x768 pixel resolution and later converted to video files of the left and right camera for simplicity. The background is added to the objects to clutter the scene. Some of the different poses for one of the objects can be seen in Figure 3.4.

As we are using stereo vision, one initial part of any solution is camera calibration. For this task, usually a checkerboard is used [64]. Multiple images from different angles and distances are usually taken and these images are used to make an internal and external camera calibration. Here we add another sequence to our dataset as the calibration sequence. We first create a plane with a known size of 100x120 mm with 8x7 checkerboard squares in 3Ds Max. The checkerboard's known size is furthermore used for external camera calibration. We need to see the checkerboard from different viewpoints to have a good camera calibration, therefore the calibration images are taken exactly with the same script of the object rendering, with the exception that less images are needed.

3.2 Real dataset

In this part we describe how to capture real image frames from multiple objects. We simultaneously create a precise pose estimation for every single frame. For this purpose we use retro-reflective markers and a monochrome stereo camera system with infrared illumination boards as discussed in [4]. We mount an additional RGB camera pair from SMARTEK Vision Croatia to the marker based tracking system to find the ground truth of the object 3D poses using markers. The frame rate of the cameras are kept to 5 frames



Figure 3.4: Rendered images of 4 different poses in left and right views for the dwarf object.

per second in 1936×1216 pixel resolution. The markers are not attached to object surface, but rigidly around the target object as shown in Figure 3.7.

Also as a matter of comparison to later methodologies of 3D pose estimations using depth channels and as a contribution to the computer vision community, we include a depth map image from a Microsoft Kinect v2 sensor, which is standard RGB-D sensor on the market with 512×424 pixel resolution. This is mounted on top of the stereo camera head as visible in the picture in Figure 3.5. Moreover, we are not going to add object CAD models, in order to have a more general approach of learning objects and keypoints rather than matching 3D models to the objects. Also in practice mostly the CAD model does not exist. The object size in the images should be big enough to be visible in all 4 cameras simultaneously and also in the working range of the Kinect sensor, which restricts the whole to a minimum distance of 0.5 meter to 1 meter.

To provide a rigid attachment for the object to for stick markers, we use a hard cardboard and attach objects to the the middle of the board. The circular markers are randomly attached around the center point of the plane. A total number of 20 markers is used to improve the matching chance and quality with the occlusion of the object as shown in Figure 3.6. For using the marker based tracking system (OTS) we use the FRAMOS software framework which we connect to the cameras, collect images, perform marker training, pose estimation and finally writing data to files. The left monochrome camera runs in continuous acquisition mode and the other three cameras are triggered by its output. With this setup, the left mono camera acts as a master acquisition camera and thus a synchronized acquisition as all images can be generated since. The exposure



Figure 3.5: Framos OTS camera mounted between Microsoft Kinect v2 and two RGB cameras.

time for the monochrome cameras is lowered ease the process of marker segmentation. The framerate is set to 5 frames per second (Hz) and the image batches are saved to files for consecutive processing.



Figure 3.6: The cardboard plane with 20 retro reflective markers attached to it

A number of eight objects is chosen for this task. The objects as in section 3.1, are objects and toys with different geometry and texture levels. The objects are chosen to be rigid and stayed firmly on the board with glue. A dwarf, a plastic toy figure, a mug, a textured cube, a toy pickup rally truck, a tea-mug and a rubic cube are among the initial dataset objects. A view of the objects are seen in figure 3.7. The objects are attached to the middle of board as the axes point using double sided glue tape. The glue tapes, markers and surroundings can be filtered based on the location to the axes center, as we do the point cloud processing task.

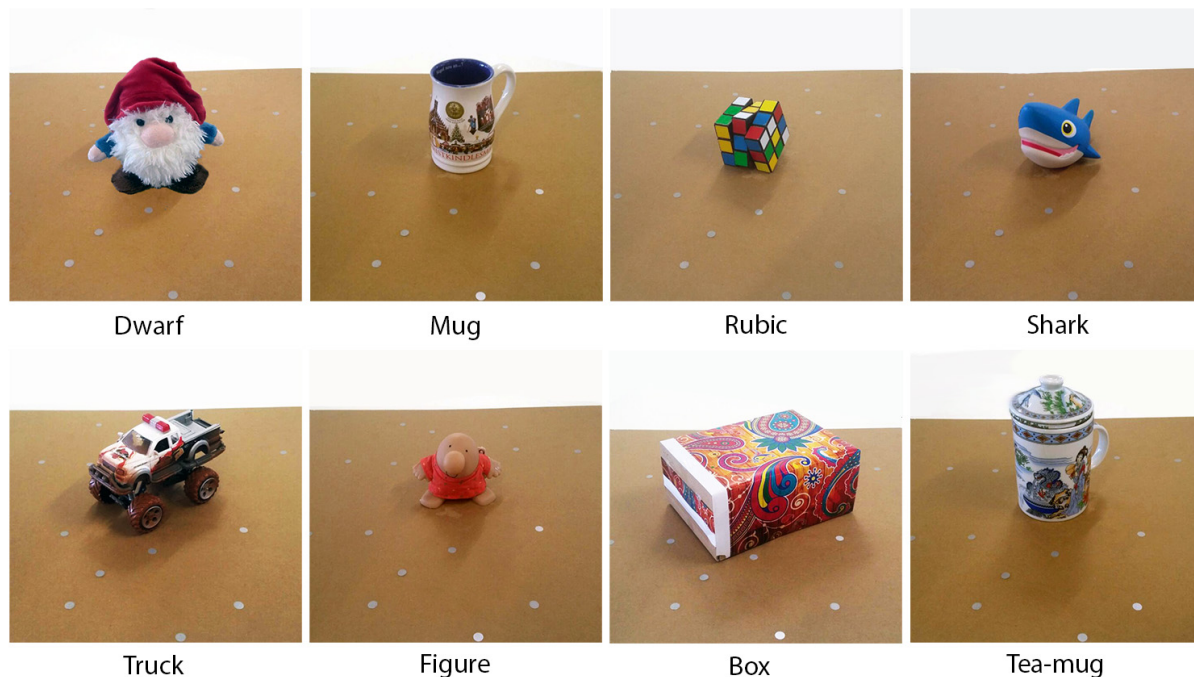


Figure 3.7: The set of dataset objects attached to the tracking plane.

In the first badge a calibration board is captured for mutual stereo camera calibration. A checkerboard is moved in range of all five cameras, including the color stream of the Kinect. All the cameras are calibrated using the same video sequence with the checkerboard movements in a bounded region. The calibration sequences are pairwise inserted into the MATLAB calibration application, which gives camera calibration matrices alongside with pairwise calculated stereo calibration parameters. The stereo calibration parameters are essential for image rectification and stereo triangulation and thus for multiple parts of the pipeline. MATLAB's reprojection errors for the stereo calibration are around 0.2 pixel which is considered as a reliable calibration error.

The second batch is for the training of the marker board. The algorithm for the pose estimation firstly needs a marker point cloud of the object or surface. The point cloud is calculated with a sequence of stereo marker images. The markers are segmented and triangulated to form a reference point cloud. Several frames are taken from different angles to gain a robust point cloud. Here, we acquired more than 200 image pairs of monochromatic images and train the object model using FRAMOS software framework to get an optimized initial pose and object cloud for the plane surface.

After these preprocessing steps, the system is calibrated and ready to collect dataset images. This time the two monochrome cameras are used to register point clouds to

the learned reference point cloud to find the object’s 3D pose. In Figure 3.8 a frame of RGB and monochrome images with markers are shown. The color cameras are taking the dataset RGB images of an object and the Kinect is giving us a depth stream to have a depth channel of the view. For every object we capture more than 1,000 images from different angles with varying azimuth and elevation. Beside the distance variation of the object is tried to cover the entire camera setup field of view.

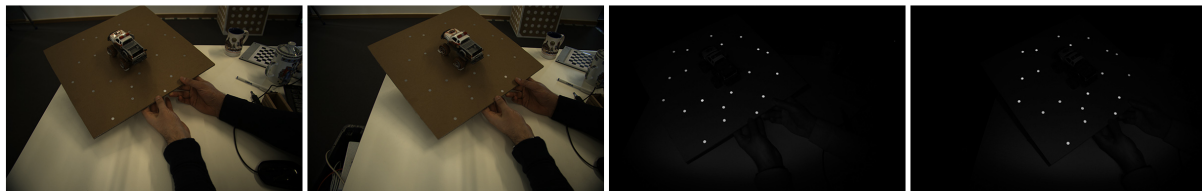


Figure 3.8: A frame quadruple of the Truck object in a pose captured by four cameras, the markers on the filtered OTS images are shown in contrast

For pose estimation we use a soft-assign method [4] followed by an ICP refinement. The soft-assign solves point cloud registration robustly especially when we have a sparse point cloud like here. The ICP is used when we have an initial good guess and is likely to provide a local minimum, if the initialization is close to it. Here we use soft-assign firstly to tackle this issue. Since the ICP relies on the previous frame’s pose, it may fail when a marker is occluded or a rapid pose change happens. The soft-assign initializes the pose estimation with a rough estimate and the output is passed to ICP which refines the pose. Nevertheless around 30% of monochrome image pairs fail provide a reliable pose estimate, which makes them useless for our dataset sequence. Some reasons for this may be synchronization issues or invisibility of markers in very close angles to the table which make the infrared reflection unseen. Possible improvements in this section can be of future works. The images with rejected ground truth are removed later. Figure 3.9 shows some frames of both RGB and Kinect depth images for four objects.

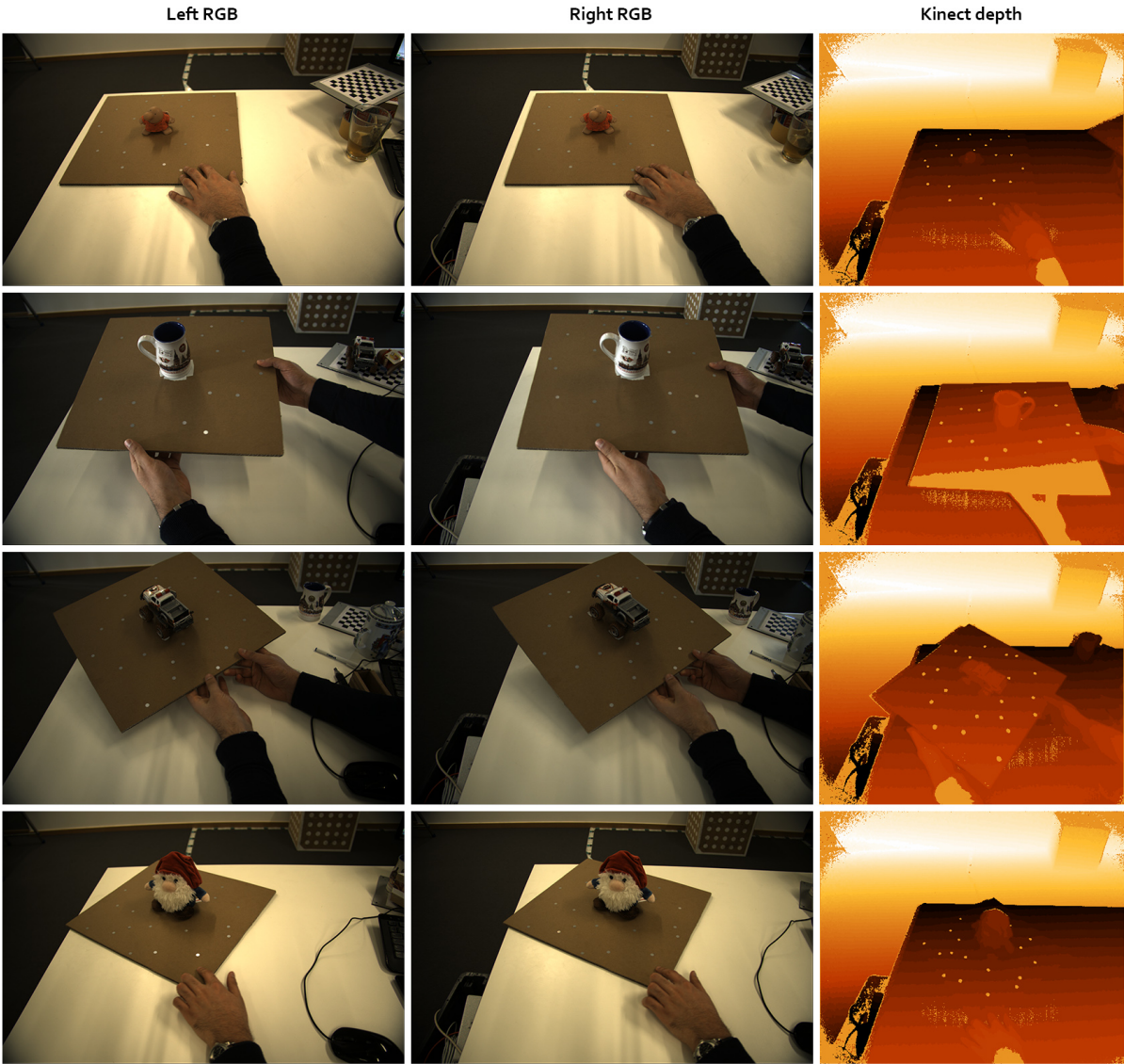


Figure 3.9: Four frames of specific pose for different objects in Kinect and RGB cameras

Chapter 4

Keypoint point cloud creation and processing

In this chapter we start describing our algorithm pipeline. We have now two datasets, one with synthetic rendered images of objects and one with real objects captured with stereo cameras. Both sequences include ground-truth pose data. Firstly, we study basic concepts of camera calibration and epipolar geometry which are used through out this chapter. Then we process single frames of the sequences by extracting keypoints in section 4.3, match the keypoints to triangulate them and calculate the point cloud using the pose sequence in section 4.4. In section 4.5 of chapter we filter the point cloud to eliminate the outliers and find marker candidates using the mean shift algorithm.

4.1 Camera model and calibration

For stereo processing, we need to calibrate the cameras. The cameras can be calibrated using the checkerboard images from chapter 2. Each camera also needs to be calibrated individually which is modeled by intrinsic parameters. For most of computer vision task a pinhole camera model is used [13]. This model describes the relation between the 3D points and 2D points on the image plane and considers the camera as a point or hole in 3D space which rays pass through in a their reflection is mapped the image plane. As illustrated in Figure 4.1 the distance of image plane and camera center is considered as focal length of camera and the x_C is reflected to x on the image plane through the camera center O .

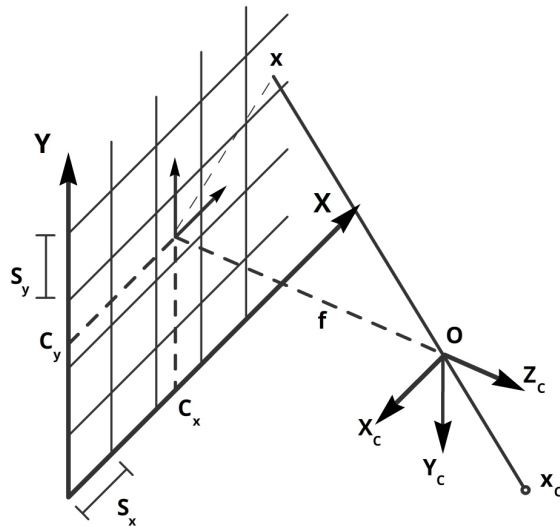


Figure 4.1: Pinhole camera model and projection of point x_C on image plane

MATLAB calibrator detects the corners of checkerboard squares and uses their

location to estimate camera parameters and to undistort the images. Proceeding Intrinsic camera parameter consider the camera pinhole model and structure. The output of the calibration is a calibration matrix K consisting of focal length in x and y direction f_x, f_y and nodal (or center) points c_x, c_y , in pixels.

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

All the optical cameras have lens distortion and therefore capture images with straight lines curved or distorted. Knowing the camera model through intrinsic parameters, this distortion can be negated and the corrected location of points are calculated. Matlab can perform image undistortion with the intrinsic calibration matrix and which is an essential pre-processing step for many computer vision tasks.

The other calibration is the extrinsic camera calibration, which uses the checkerboard size as well and maps the camera coordinate system to the world coordinate system. It consist of a rotation R and a translation T . The world coordinates are transformed to camera coordinates using these two factors. In total the intrinsic and extrinsic camera matrices form a camera matrix P which is used to map the world points to 2D image coordinates.

$$P = K \begin{bmatrix} R & T \end{bmatrix}$$

$$w \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = P \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

In the above transformation $[x, y, 1]^T$ are the image points and $[X, Y, Z, 1]^T$ are the world points and $w \in \mathbb{R}$ is a scale factor.

To calibrate a camera MATLAB does minimization iteratively to estimate the calibration parameters r [64]. For m images MATLAB detects the corner of checkerboards using Harris corner detector. For n detected corners in each image x MATLAB calculates the 3D points x_w and tries to minimize the summation over images as follows:

$$\min \sum_{j=1}^m \sum_{i=1}^n \|x_{i,j} - P(x_w^{i,j}, r)\|^2$$

4.2 Stereo calibration and epipolar geometry

Beside these individual camera calibrations we need a mutual stereo calibration which is the calibration of cameras with respect to each other. The distance between the camera centers, the so called baseline and the rotation of the camera coordinates are calculated in this process. To formalize we introduce the concepts of epipolar geometry [13]. The epipolar geometry is the projective geometry between two views. It depends on the intrinsic camera parameters and their relative pose.

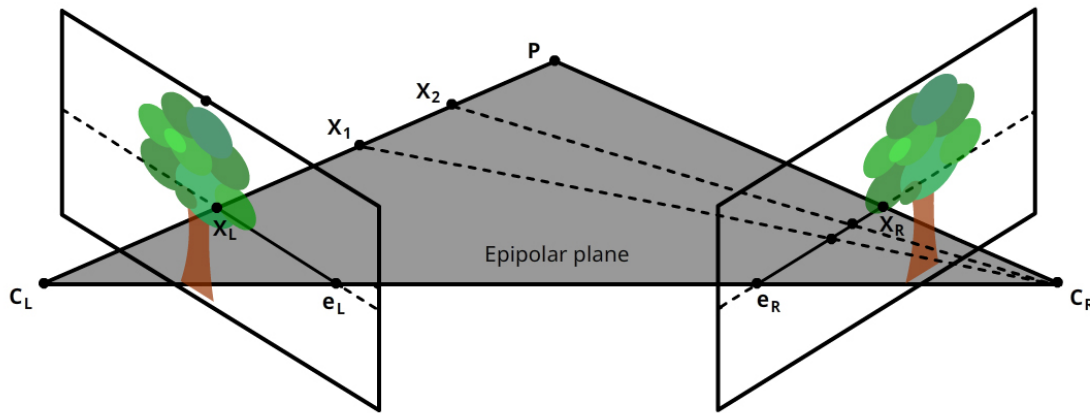


Figure 4.2: Epipolar plane of a stereo view restricts the point stereo matching problem for correspondences to an intersecting line.

As Figure 4.2 denotes: In a pinhole camera model, the epipoles e_L and e_R are the points where the line between the camera center points c_L and c_R meet the image plane. The points X_L and X_R are the projection of the world point P to the left and respectively the right image. The line $X_L e_L$ and $X_R e_R$ form epipolar lines. All epipolar lines intersect in the epipoles. Epipolar lines represent a restricted search space for stereo correspondences, meaning that the matching points in stereo images need to be on the same epipolar lines. In the figure X_1 and X_2 show two of the candidate examples for correspondent points of X_L in the right images along their ray. The same principle can be applied other way round to find matching points of X_R in the left image.

The outcome matrix for camera calibration is the fundamental matrix $F \in \mathbb{R}^{3 \times 3}$, which is used to transform the image points of the cameras and encapsulates the whole epipolar geometry for the given stereo setup. The homogeneous matrix F is of rank 2 and it can be assumed for epipolar lines that $e_R = F x_L$ and $e_L = F x_R$. If P_1 and P_2 are matching points from a stereo image pair lying on the same epipolar line, it holds

$$\begin{bmatrix} P_1 & 1 \end{bmatrix} F \begin{bmatrix} P_2 & 1 \end{bmatrix}^T = 0$$

Another important use of epipolar geometry and as a pre-processing for most of the stereo matching pipelines we can rectify stereo images, as Figure 4.3. Epipolar lines usually can be different from one image to another. The process of image rectification helps us having the epipolar lines in the same fixed horizontal scanline. The images are undistorted and transformed to have all correspondence points at the same y coordinate in the image. This helps scanning the images and finding stereo matching points to a great extent.

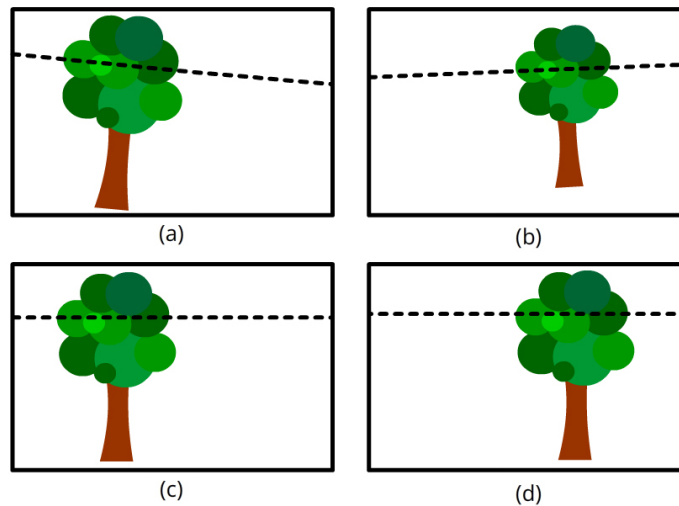


Figure 4.3: Schematic (a) and (b) are the left and right images before rectification. (c) and (d) are the left and right rectified images, where corresponding points can be found within a fixed line.

4.3 2D keypoint extraction

Towards learning keypoints all around an image, we need to have 2D keypoints from stereo views first. Epipolar geometry helps significantly in our case, to filter and to extract 3D information, with which we can process our extracted keypoints later. But first we need to prepare our images to have a foundation for our keypoint extraction process.

4.3.1 Data preparation

We input the videos from the dataset image sequences first alongside their 6-DOF poses. We also have the calibration parameters with which we can rectify our stereo images. This is vital since we want to filter the keypoints later based on stereo matches. Each view is rectified first using the concepts described in section 4.2 . It is important to note that the images from the rendered dataset need no processing for rectification. The reason for this is simply because the camera views in the 3D software are perfectly aligned and parallel with respect to each other such that these images are already rectified. The real dataset, however, is subject to slight tilts and rotations, thus we need to rectify the image to align the views horizontally.

4.3.2 Feature extraction pipeline

By using MATLAB we can directly use a handful of handcrafted feature detectors from the vision toolkit [65]. This gives us the ability to extract different features and generate a pool of the keypoints. Here the feature description is not of high importance, since we want to learn any form of surface spot which is salient and robust. A keypoint coming from different methods helps us to achieve an inclusive set of keypoints.

We choose four different keypoint extraction methods for our pool, namely Harris Corner detector[19], SURF [16], BRISK [17] and MSER [66]. Harris concentrates on corners and edge information, extracts and provides corner points on the objects. SURF and BRISK, as described in section 2.1, use more advanced routines to extract 2D features and are more robust to viewpoint changes. These keypoint extractors detect not essentially the same keypoints and generate different features. BRISK has a feature description of size 32 and the SURF descriptor space is 128 dimensional [67]. Maximally stable extremal regions (MSER) looks for regions containing blobs of different shapes. This helps us finding shape regions and blobs which are usually of a different nature than the aforementioned features. From MSER we generally get a list of blobs and connected components and use the center points as our entry keypoint.

For all four feature detectors we use the implementation of MATLAB [65] which works for grayscale images. Therefore, we first convert our image pairs to grayscale images and then extract keypoints. The location of the keypoint is listed in a 2D array for the left and the right image and passed to the next stage as keypoint matching.

We need pairs of matching keypoints for both stereo triangulation and 3D processing.

As a common ground for keypoints we should pick a descriptor to describe all the keypoints. We use FREAK [24] descriptor as a basis of our matching and MATLAB's default for matching points. The descriptor is of SIFT nature and can be matched easier using an L2 metric. The distance in descriptor space is also later computed in section 5.2.1 by means of Euclidean distance. We match the points and after eliminating the rejected matches a fraction of keypoints as early matched keypoint remains.

Since the images are rectified, the stereo matching benefits the constraint of epipolar lines. The assumption helps us checking the y position of matching points and reject the ones that look similar but lay at distant scanlines. The accepted tolerance of $|y_L - y_R|$ is set to 2, meaning that the points whose y difference is more than 2 are filtered out. We also check the x -position and put a constraint as $|x_L - x_R| > 5$ to only keep points with disparity of more than 5. This a condition to remove the points with a large distance to the camera which are presumably not on the focused object.



Figure 4.4: Different poses for Dwarf object on rendered dataset and its keypoint correspondences.

We also save the RGB colors and feature descriptor of the keypoints in two other arrays besides the array saving the 3D location of keypoints for section 5.2.1 and 5.1. Some of the keypoints extracted from the Dwarf object for some views are shown in Figure 4.4. The images depict left and right images overlaid and the matching keypoints connected to each other with lines.

4.4 Building keypoint point cloud

In this section we want to locate keypoints found in 2D images in 3D space in order to form a 3D point cloud. The point cloud shows keypoints and concentration of keypoints in 3D space. To achieve this we need keypoints in all frames of the datasets from chapter 2 together with their ground truth pose, which we use to transform them in a common coordinate system.

4.4.1 Triangulation

Firstly, we need to triangulate point pairs to get to their 3D location in camera coordinates. For triangulation of rectified images we need to know some key parameters from the camera calibration process. The first is the focal length f . From the intrinsic camera calibration we have the two values f_x and f_y representing the focal length in relation with the pixel width and height, which for cameras with squared pixel is considered equal ($f = f_x = f_y$). Then we need the principal point or image center [13]. The principle point in a pinhole camera model is the intersection of the optical axis with the image plane. C_x and C_y are the components of the principle point in pixels, which in our case is in the middle of the images ($C_x = 512; C_y = 384$). Also the baseline, which is the distance of the camera centers in 3D space, is directly given from the synthetic dataset with $b = 70mm$.

For every pair of keypoints (k)_{*n*}, we define the term disparity $d(k)$, which is the difference of the movement of the point from one camera to the other. Here with rectified images the horizontal difference, thus the x -components of the keypoints gives the disparity:

$$d(k_i) = L_x(k_i) - R_x(k_i).$$

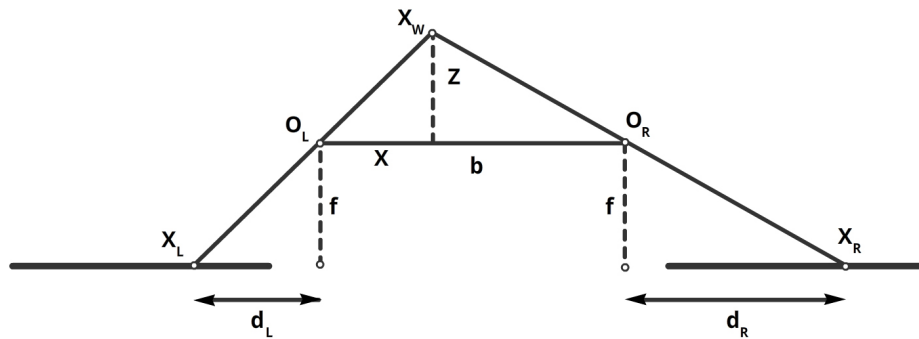


Figure 4.5: The 3D coordinates of a point with respect to focal length f and disparities d_L and d_R

As illustrated in figure 4.5 for every pair of keypoints we triangulate one keypoint in 3D space with three terms $X(k_i), Y(k_i)$ and $Z(k_i)$ using its disparity and the parameters that we have. The 3D coordinates are given by:

$$Z(k_i) = \frac{fb}{d(k_i)}$$

$$X(k_i) = \frac{(L_x(k_i) - Px)Z(k_i)}{f}$$

$$Y(k_i) = \frac{(\frac{L_y(k_i) + R_y(k_i)}{2} - Py) * Z(k_i)}{f}$$

Therefore, we now have a point cloud of keypoints in camera coordinates for every frame, namely $PointCloud(i)$. In order to build a point cloud of keypoints for the object we need to transform the keypoints according to the movement and rotation in the 3D software. This brings all the keypoints in the same reference coordinates and we can merge them. Figure 4.6 shows the pose applied to the object and its local coordinates with respect to the fixed camera coordinates. The world coordinates are defined as the coordinates of the left master camera, both during the dataset creation and later for the camera calibration.

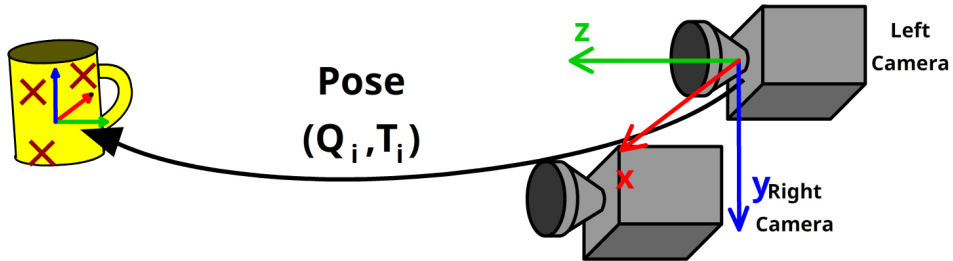


Figure 4.6: The stereo cameras, triangulated keypoints and the object pose in reference to the world coordinates

With every pose that we have per frame pair as $Q(i), T(i)$, we first need to apply the inverse of translation and then apply the inverse of the rotation to bring the object coordinates to the world coordinates; the opposite order was used for the initial displacement of the objects. The transformed point cloud, $PointCloud_{ref}(i)$, is computed as follows.

$$PointCloud_{ref}(i) = R^{-1}(i)(PointCloud(i) - T(i))$$

Here $R^{-1}(i)$ is the inverse of a 3×3 rotation matrix converted from the Quaternion matrix. Applying this transformation to all the keypoints of all the frames, a point cloud of keypoints is created in common reference coordinate system.

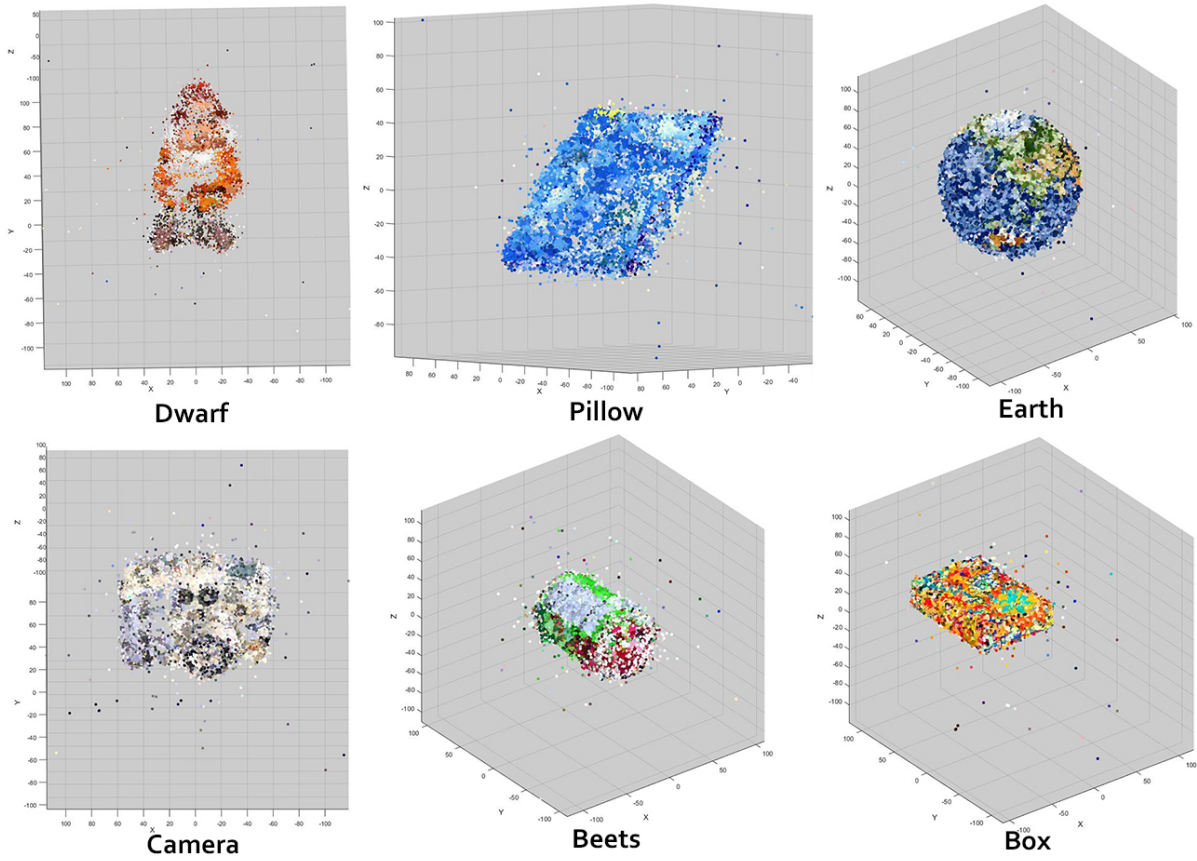


Figure 4.7: The keypoint cloud for 6 different objects. Colors show the mean color of the extracted point on both the left and the right image

The output point cloud has the size of thousand of keypoints depending on the level of texture of the object. Outliers that are positioned outside a pre-defined bounding box are directly removed. However, still some noise and outliers are left which are removed with the concepts detailed in section 4.5 after the mean-shift algorithm.

4.5 Finding keypoint clusters

In this section, we want to find marker candidates from a dense point cloud. In order to achieve this, we perform clustering on the point cloud generated in section 4.4. For this we made use of men-shift algorithm [67].

4.5.1 Mean-shift

The advantage of the mean-shift algorithm compared with other methods such as K-means [68] is that it is parameterless. We also tried other algorithms here, but mean-shift performs of best, mainly because the point cloud clusters are highly dependent on the object texture and complexity. Moreover, there is no assumption on how many keypoint classes are needed.

We first create a matrix of points and colors. each row of the matrix is the location of the point in 3D and the mean color of the keypoint in the left and right image. We add the color term to mean shift to have more robust keypoint classes. The result is that classes cannot contain keypoints that are located close to one another but differ in color. The same holds true if a class is so wide that the locations are considered different but the color is very homogeneous.

For mean-shift we use a Gaussian kernel and a bandwidth of 5. Even though mean-shift by itself is a parameter free clustering algorithm, it requires a bandwidth scale term to scan the input data which can be seen as the standard deviation for the individual clusters. The choice of bandwidth is always a matter of trade-off. Having a bigger standard deviation includes more noisy triangulated points in the clusters while a small value produces many small clusters.

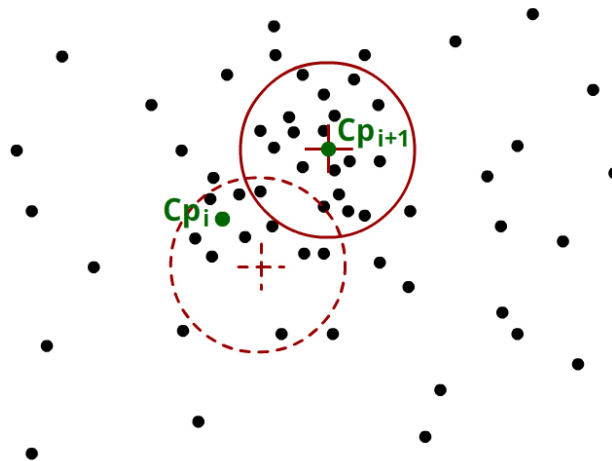


Figure 4.8: Mean-shift clustering algorithm of 2D point data. The algorithm terminates with the center of the clustering circle as the centroid of the points $Cp(i)$.

Mean-shift clustering works in a window fashion [67]. Here we can consider a circle for a 2D set of points as illustrated in Figure 4.8. Suppose that we want to find a cluster with the maximum point distribution inside the circle. The algorithm proposes that a

good distribution has the centroid of points (Cp_n) matched with the center of the circle. In each iteration the centroid of points inside the circle is calculated and if its away from the center, the center is updated to have the maximum number of points inside. As Figure 4.8 illustrates, in the $(i + 1)$ th iteration, we have the Cp_{i+1} at the center of the circle and the cluster has its most dense point distribution.

A similar window approach is taken for more dimensional spaces, such as our 6D input (3D location, RGB color). The ean-shift method returns the centroid of clusters and the points included in each cluster. This helps us count the points in each cluster in order to weight each cluster individually. The repetition of the points in the keypoint extraction module can also be evaluated with this term.

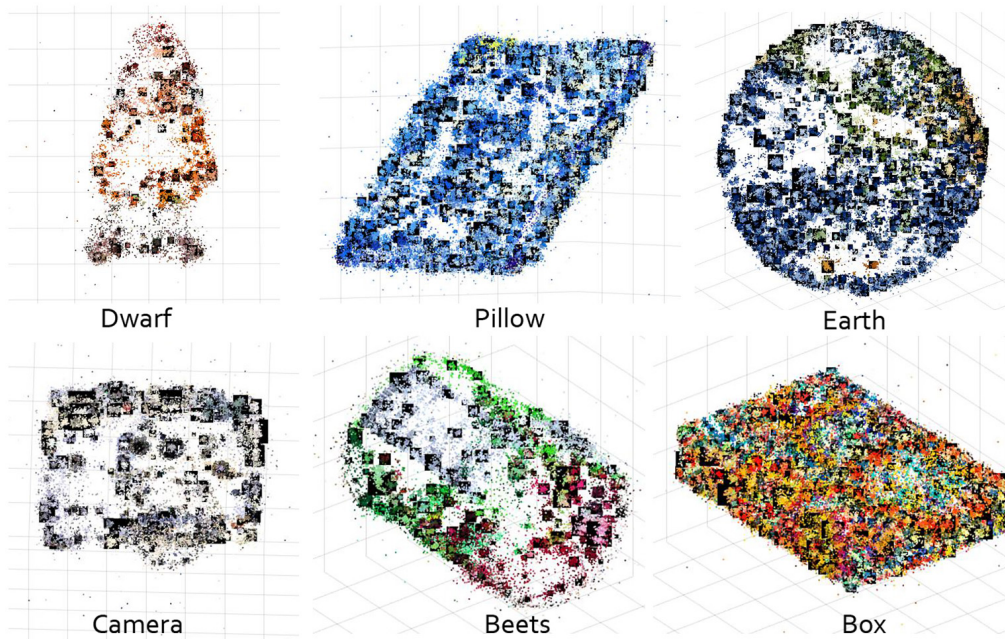


Figure 4.9: The keypoint point cloud and their mean-shift cluster sizes in black blocks

As a side effect, many points that stay as outliers stand in a singular clusters or very small cluster. Therefore, we can filter these points to have just the classes with relatively crowded clusters. In our datasets, we decided that a cluster that has less than 25 members is eliminated. For the ones for which the cluster is dense enough, the centroid of the points is chosen as the keypoints representative of the class. As a next step, we want to produce a candidate marker point cloud which means that we need one point per class. For this we need to compute the parameters from the points inside each cluster.

Beside the calculated position, each keypoint needs to have a mean rotation, mean color and mean feature descriptor. The mean pose is calculated as an average of the

poses of the points in class. Every pose has a quaternion component and we want to take the mean of these quaternions. Since the unit quaternion form a non-Euclidean space, this is not a straightforward method. We create a function to perform a spherical linear interpolation (SLERP) for multiple quaternions [69]. Slerping is a method for nonlinear interpolation of quaternions, which is mostly used in 3D graphics and gaming. Here we create a method to interpolate multiple quaternions of the class keypoint all with an equal weight.

In order to form the mean of color and features we use similar approaches. We also trim some of the outliers for calculating the mean to have more smooth outputs. Figure 4.10 shows the keypoint classes, their mean color and their number of occurrence depicted as their size. After the mean-shift and post filtering, there are tens of salient classes left for most of the objects in the dataset.

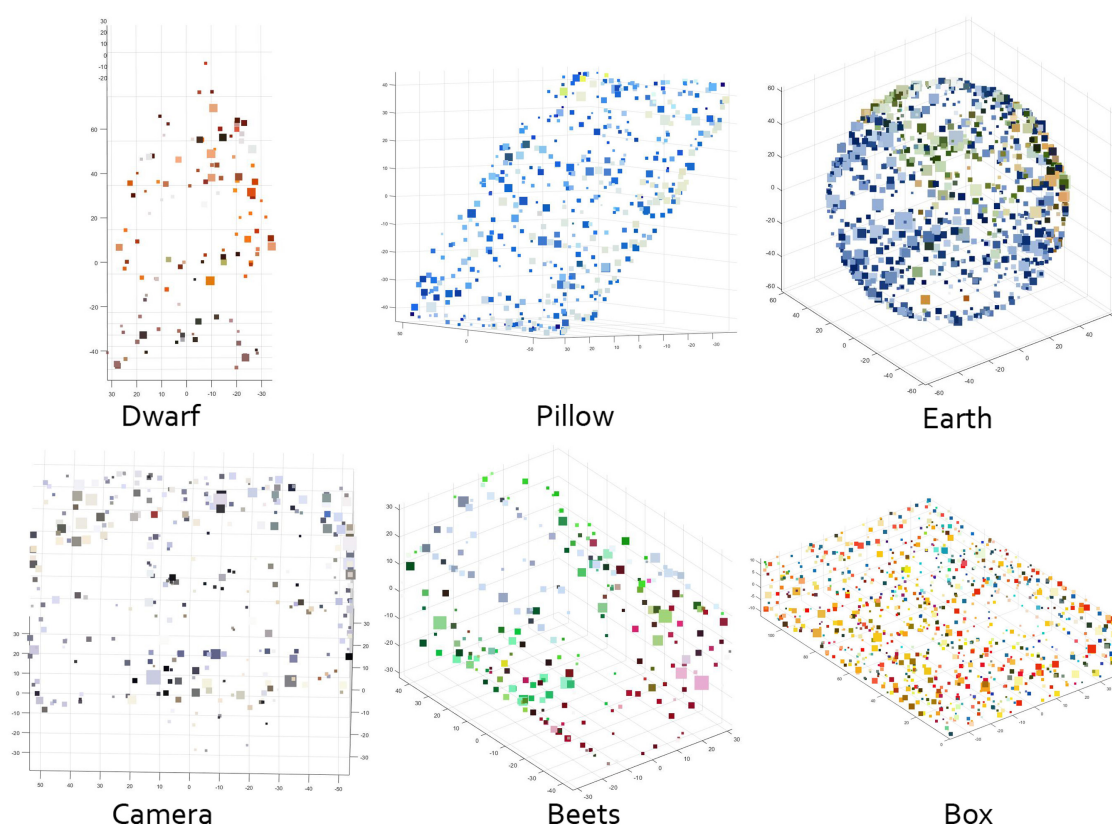


Figure 4.10: The marker candidates point cloud shown with their mean color and number their repetition as their size

Again in this section we see that the output is dependent on the texture level of the objects. For example the box or earth object have almost uniform point clouds with

nearly the same repetition per class. The camera object on the other hand has some faces with some high textures and some flat color faces which makes it non uniform. The Dwarf object also has less texture but also small details like the small hands or the face which are represented as clusters. The beets can has also a very complicated keypoint structure. While the two caps have a flat aluminum surface, the cover has some coarse textures and some texts as labels; the labels can be well distinguished in the marker candidate point cloud.

We have to select the best keypoints that in dependent of these variaties and can down-sample the point cloud to a less dense point cloud. In next chapter we look at how we can process these point cloud to designate the most robust markers on the objects.

After this section we have a filtered marker candidates point cloud that has salient keypoint representations with at least 25 keypoints included per cluster. The 3D location of marker candidate point cloud is saved for the next chapter to score the markers and subsequently select our final marker point cloud for training phase.

Chapter 5

Natural marker criteria and selection

In this chapter we want to use the marker candidates point cloud and find the best markers that are salient, well distributed, repeatable and also distinct. To achieve this we first reproject the keypoints onto the images and then use these reprojections to assign a score to the keypoints. Finally we pick our natural markers from the candidates by their score and location.

5.1 Keypoint reprojection

We need to reproject the keypoints mainly because our basic object information comes from the 2D images and their pose. Furthermore, the mutual presence of keypoints can be assessed. Some of the keypoint classes may not match with their reprojections and some geometry complexity may also hide keypoints. A keypoint should be picked such that it has a large field of sight. However, this cannot be taken into account using common handcrafted features which are sensitive to intensive viewpoint changes [3]. Thus, the test is to render our marker candidates on the stereo images and count their presence altogether.

5.1.1 Rendering ideas

Rendering or reprojection of a sparse point cloud is a challenging and problematic. In a conventional rendering scenario, a dense point cloud or surface and the presence of a point in the rendered image can be found by checking occlusions of other points or surfaces in front of them. In this case, however, we have a very sparse point cloud and in some regions there are only a few points, which does not give a reliable base to render a point. On the other hand, rendering on the images also does not help as much as one could initially expect. We have tried several different keypoint characteristics such as features, colors of keypoint and their reprojected location by checking whether these match and thus rendered the point on the image. But this does not perform well on most of the monotonic color surfaces and repeating textures, since the color of the back and front of the object is probably similar.

Also checking point locations only with regard to the whole point cloud can not be a reliable resource, since some objects have no keypoint in a region or are very thin, such that the location of points can neither be used. Moreover, we can use the pose information calculated in chapter 4, of all the view angles from which the point was extracted. This is also a complex setting, since the field of view in which a point is sighted depends on

the shape and geometry of the object and for some of the poses we do not have enough extractions from images for a representative statistical approach.

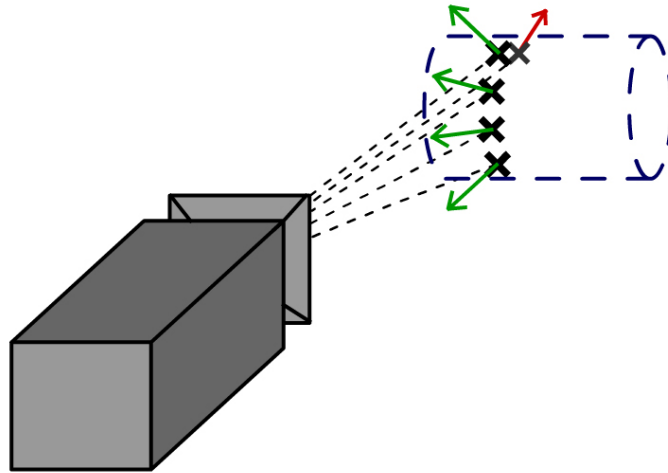


Figure 5.1: Ray-casting for rendering on the intersection of rays with the object surface

A widely used and rather old rendering pipeline is ray-tracing or ray-casting [70]. In this technique, rays are casted from a camera center and for each intersection with the object, it is checked whether the spot is visible or invisible for the rendered output. One way to check this, is to render only the first intersect point and ignoring the rest of the points along the ray. This works fine for meshed objects or dense point cloud rendering. Another term which is used often times in rendering techniques is the normal vector [71]. In geometry a normal vector is the vector perpendicular to a surface. This term can also be applied to points and 3D graphics. Particularly in rendering, the direction of the normal or the side on which the ray is coming out defines whether the surface is visible or not. In the following, we also want to use a similar principle to help the rendering of points for our markers.

Finding normal vectors for a sparse point cloud, however, is a challenging problem. To find a normal vector to a point we have to find a rough surface around the target point at first. To achieve this we perform a the neighborhood of points surrounding the point in 3D and use Principle Component Analysis (PCA) [72] to find the best fitting surface. PCA is widely used for dimensionality reduction. It is a statistical method to find principle components of a distribution using an eigenvalues decomposition. The algorithm can find a set of uncorrelated vector or principle components such that the data has its largest variance along them. The first two principle components can form a plane which can be fitted to the points.

For finding the nearest neighbors around a point, we use the MATLAB KNN search [73] implementation. KNN finds the K nearest neighbors of a point in a point list. The algorithm forms a k-d tree architecture to search the points in space. As we set $K = 10$ the algorithm returns the 10 nearest neighbors of the point. These points and the target point are then put into a PCA routine which returns the two principle components c_1 and c_2 . Consecutively, a cross product of the two vectors is calculated as $N = c_1 \times c_2$ and the vector N is saved as the normal vector to the target point. Figure 5.2 shows an illustration of a plane fitted to 11 points and the vectors on the plane show the principle components of the point distribution. The Normal vector N is a vector perpendicular to the surface that is fitted to the neighborhood distribution.

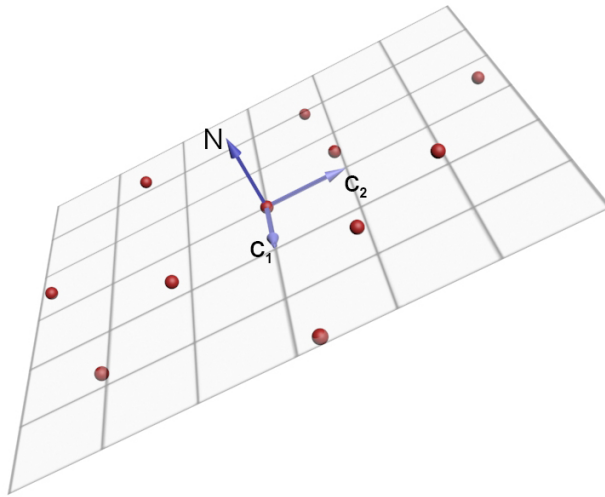


Figure 5.2: Finding the normal vector to a point using neighboring points and PCA.

The sign of the normal vector is important here, since it gives an assessment on the visibility of the point. For the direction or sign of the normal vector, we choose between N and $-N$ the one with a bigger angle towards the center of the object. This angle is computed from the point and helps finding which direction is not towards the center for a convex object. The mentioned method is applied to all the points in our point cloud and an estimation of normal vectors is then calculated. Figure 5.3 shows a visualization of the normal vectors for our input point clouds which produces reliable result for both simple and complicated geometries.

After this step we can use the ray-tracing concepts to render any points. As Figure 5.1 depicts, the rays from the camera center enclose an angle with the normal vector on the surface of the object. If this angle is bigger than 90° , the point is likely an invisible one and if it is smaller than 90° , it is probably in front of the object and therefore seen.

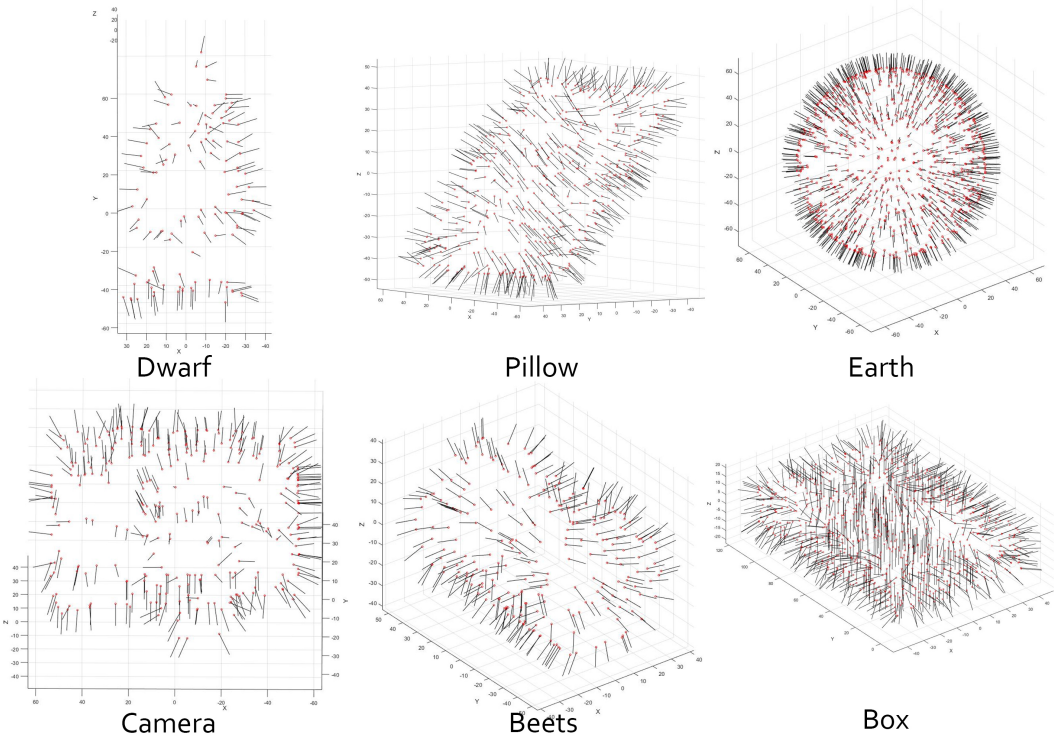


Figure 5.3: Normal vectors (black) from the keypoint point clouds (red) in red for six different objects of the dataset

To make this condition more reliable we can bring in some redundant checks for color and pose too.

For backprojection of the points, we first bring them points back to camera coordinates. This is done exactly as the inverse procedure described in the section 4.4 where we took the points from camera coordinates to the local reference coordinates:

$$PointCloud_{cam}(i) = R(i)PointCloud(i) + T(i).$$

Then we need to project the points onto the images. Since we have two cameras we need two projection matrices here. A camera projection matrix is a 3×4 matrix which maps 3D points to pixel coordinates. This can be used for us to map the point cloud to the stereo images one by one. The camera calibration matrix K which is computed in the calibration process described in chapter 4 is used again at this stage to calculate the projection matrix.

$$P = K \begin{bmatrix} R|T \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R|T \end{bmatrix}$$

We calculate projection matrices for the left and the right camera as P_x and P_y for the synthetic datasets, the two cameras have the same intrinsic but different displacement components. The amount of translation of the right camera to the world coordinates is the same as the baseline b . Also note again that the camera coordinates of the left camera are the world coordinates axes such that

$$P_L = K \begin{bmatrix} I & 0 \end{bmatrix}$$

$$P_R = K \begin{bmatrix} b \\ I & 0 \\ 0 \end{bmatrix}.$$

We calculate the projected location for all the keypoints i and cast the points on the image if they meet the conditions with normal angle $\angle N(i)$ and color. The color in the projected points are known as the projected left color $C_L(i)$ and the projected right color $C_R(i)$. If a color is in front of the object and seen, the projection is correct and the points should be correspondent and therefore with almost the same color. In case the projection is not visible the points probably do not match in the left and right images. We use a threshold for the left-right consistency check t_c and the color check works as an add-on

to the normal vector check to decide for the points which close to the threshold for the normal vector condition. $V_K(i) \in [0, 1]$ is the visibility array which states whether a point is shown or not.

1. To project points we need to check the normal vector angle with the camera ray and the left-right color consistency

Algorithm 1 Visibility Check

```

1: for  $i = 1$  to  $size(keypoints)$  do
2:   if  $\angle N[i] < 80$  then
3:      $V_K[i] = 1$  ▷ Project Point
4:   else if  $\angle N(i) < 100$  then
5:     if  $|C_L[i] - C_R[i]| < t_c$  then
6:        $V_K[i] = 1$  ▷ Project Point
7:     else
8:        $V_K[i] = 0$  ▷ Do not Project Point
9:     end if
10:  else
11:     $V_K[i] = 0$  ▷ Do not Project Point
12:  end if
13: end for

```

The aforementioned algorithm checks all potential reprojections on the images. In Figure 5.4 you can see the reprojections in some of the left frames for the dwarf object sequence. For every pixel which is reprojected, a reliability criterion of the keypoint should be increased. In order to do this together with other marker reliability scores, we define first the criteria for marker score first and subsequently explain the scoring pipeline in detail.

5.2 Keypoint scoring

Here we have three main criteria which we model through the section: Firstly, distinctness, which takes care of salience and how distinguishable a keypoint is among the other keypoints. Secondly, repeatability which is a criterion to model how often a keypoint is seen in the ground truth with different view angles. And thirdly distribution of the keypoints all around the object comes in by adding the position and geometry of keypoints to investigate the best ones. All properties are analyzed and combined to give a scoring to find the best natural marker collection.



Figure 5.4: Keypoints accepted and projected to the left images for some samples of the Dwarf object.

Another key note here is that not all the frames produce the same number of keypoints initially, but we need a similar number of keypoints to be detected later to have a more uniform distribution of keypoints. This makes us to keep track of the balance between frames for selecting the best keypoints, therefore we have to normalize the accumulated scoring per frame.

5.2.1 Distinctness

A good keypoint should be distinct from other keypoints or points. Repeating or similar texture on surface may lead to very similar keypoints. An approach should be taken that these similar keypoints have less chance to be selected. The keypoints that have similar keypoints in the candidate set get a lower score.

In order to define the first criterion we define a matrix of distinctness $U \in \mathbb{R}^{m \times n}$, where m is the number of keypoints. Each keypoint descriptor is the mean descriptor of the keypoint which we calculated in the extraction phase in section 4.4. Afterwards we need to calculate the distance of keypoints in the descriptor space of dimension 128. As we use FREAK descriptors we can take the Euclidean (L2) norm to find the similarity

between keypoints. We calculate the distance of each keypoint to all the other descriptions.

$$U(i, j) = \|FREAK(i) - FREAK(j)\|_2$$

We can accumulate all the difference for each keypoint to find a metric for scoring distinctness of keypoints.

$$U(i) = \sum_{j=1}^m U(i, j)$$

The larger the summation get, the bigger is the distance of the keypoint discription with respect to the rest of the keypoints. We call $U(i)$ the distinctness or uniqueness of the *keypoint*(i).

5.2.2 Repeatability

The second important property is the keypoint repeatability. A salient and robust keypoint should be reproducible so that it occurs very frequently in the ground truth acquiring stage. Being well sighted is of influence for this point as well.

In section 5.1 we found a way to count the number of frames a keypoint is spotted and mentioned that this as an important factor for a good keypoint. To score the keypoints, instead of incrementing the number for visibility in the dataset sequence, we integrate the distinctness to the repeatability. First we form a visibility matrix $V \in \{0, 1\}^{m \times n}$, where n is the number of frames, which in our case is 1000 and m is again the number of keypoints. When the *keypoint*(i) is sighted in frame j , we set $V(i, j) = 1$, and in case the keypoint is occluded or not visible, we set $V(i, j) = 0$. Having the matrix filled, we can calculate our combined repeatability and distinctness score integrated ($R_U(i)$).

$$R_U(i) = U(i) \sum_{j=1}^n V(i, j)$$

$R_U(i)$ is a criterion to demonstrate if a well distinct keypoint is sighted frequently. Of course if a keypoint is not perfectly distinct but well seen in a large number of frames, this criterion can compromise with the influence of repeatability.

5.2.3 Distribution

For the task pose estimation we need a certain amount of markers as a minimum requirement per single frame to have a fine pose estimation. This makes a valid criterion for marker evaluation. If the selected markers are well distributed all around the object, in each view there may be enough markers to detect. Thus the location of the markers in respect to each other also plays an essential role.

In our scoring criteria we have to integrate the distribution into our scoring method. Imagine an object that has a poorly textured side or face, which is the case for many ordinary objects: For example if a face of an object has 10 keypoints, and another face has just 3 keypoints, we have to put a privilege for the ones that are in the group of 3. In order to give this privilege, we first normalize the keypoints visibility by frames and also choose the final keypoints based on the score.

Normalizing the repeatability factor on the frames means that each frame can have a single vote. Therefore if three keypoints are detected in a view, each keypoint gets 1/3 of vote and if 10 keypoints are detected, each keypoint has a weight of $w = 1/10$, and is therefore less important. In our scoring metric, we normalize the V matrix on every frame, such that with we get a normalized repeatability score $R_{UN}(i)$:

$$R_{UN}(i) = U(i) \cdot \sum_j W_j V(i, j)$$

with:

$$w_j = \frac{1}{\sum_i U(i, j)}$$

Finally we want to pick our set of keypoints which we want to call our natural markers. We Already calculated the scores for each keypoint as $R_{UN}(i)$. As another factor of distribution, we also pick the points based on their 3D location with respect to the other keypoints. We introduce a minimum distance parameter min_{dist} . This parameter is the minimum 3D distance two picked keypoints can have in 3D space. This is important for having a well distributed keypoint network and also training dataset for the learning pipeline hereafter; We set this parameter to 20 mm.

The Next step is to sort the keypoints based on their score. The top scored keypoint is picked for sure and the keypoints that are in the neighborhood of this point in a radius of min_{dist} are eliminated so that they can not be chosen. Then we iteratively look for the highest scored keypoints available and eliminate the neighbor keypoints until we either have the maximum number of possible markers $max_{markers}$ picked or we run out

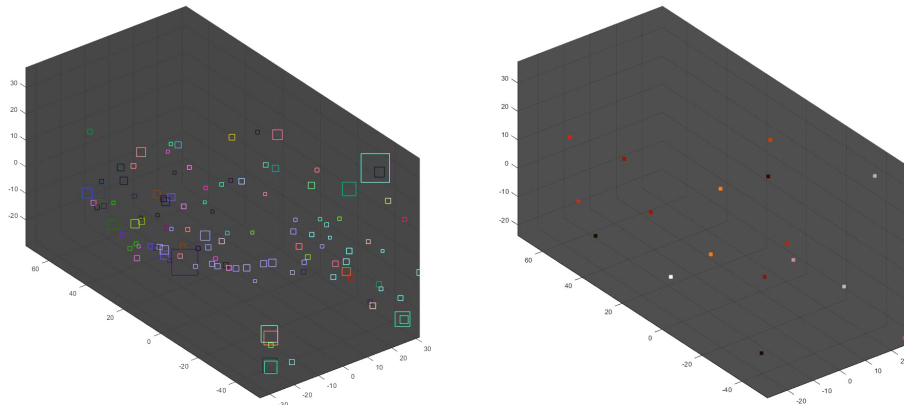


Figure 5.5: Marker candidates point cloud and their scores on the left and the selected best markers on the right.

of keypoints. For our implementation we set $max_{markers} = 20$, which is enough for the task of pose estimation and does not make the pipeline very costly. Figure 5.5 shows the initial marker candidates set for the dwarf object with their score and the selected set of markers for the same object. Moreover Figure 5.6 illustrates the point cloud of natural markers for the objects we have extracted keypoints from.

As the outcome of chapter we filtered and scored out marker candidates to achieve the best set of markers using metrics obtained from the dataset in chapter 4. Furthermore, we integrated these standards to an adaptive marker scoring approach. The quality of markers and an evaluation on the markers is discussed in section 7.1. The selected marker sets are fed into our training architecture in chapter 6, as investigate how we can classify the markers and distinguish marker patches from non-marker patches.

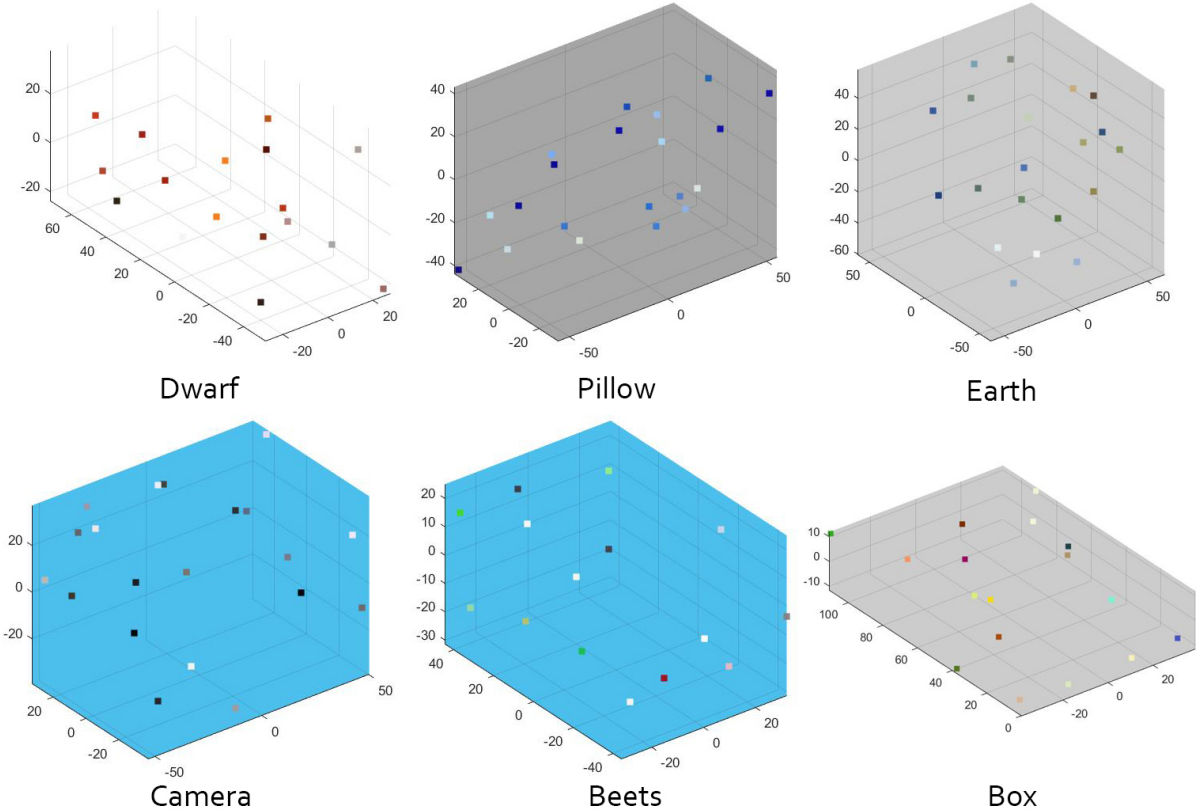


Figure 5.6: selected marker sets for different objects of the dataset

Chapter 6

Marker learning and pose estimation

In this chapter we want to train our selected markers on the image patches. We first back-project the markers and gather our training and testing dataset. Then we feed the points to a deep convolutional neural network which we propose in the second section. Finally we explain how we can use the trained network to classify markers and reproduce a 3D point cloud which can be used for object 6-DOF pose estimation.

6.1 Marker dataset creation

At this stage we create a dataset to train the markers. This procedure runs automatically after the keypoint selection procedure discussed in chapter 5. It is worth to note that in order to include all different keypoints for different objects and different patch sizes, we need to have an adaptive dataset creation and marker training.

We reproject the keypoints using the same principles as introduced in section 5.1. Here we need to extract a patch relative to the object size in the projections, since the object's depth is varying. so we use the keypoint reprojections to estimate the rough the object size. The window for the frame in each view is calculate as follows:

$$\begin{aligned} size_x &= \max(keypoint_x(i)) - \min(keypoint_x(i)) \\ size_y &= \max(keypoint_y(i)) - \min(keypoint_y(i)) \\ window\ size &= \frac{size_x + size_y}{6}. \end{aligned}$$

For the marker dataset we will resize all the patches to 40×40 pixels. We also augment the training data with a patch rotation of $[0 - 360]^\circ$, a movement shift of $[-5, 5]$ 0 pixels in x and y direction, and a scale coefficient of $[0.5, 2]$. These transformations are applied to the patch dataset creation, so that we can increase the amount of our training and test data while boosting the robustness. We apply the random scale, shift and rotation to the target patch and then resize the patch to the size of 40×40 pixels.

For a total of n markers, we need to create $n + 1$ classes. One class remains for non-marker patches and n classes for the patches that contain the marker $1, \dots, n$. For the marker patches we use the patch around the marker with a small shifting as described and for the marker-less (negative) class we generate a set of random regions that are not markers, 70% from outside the region of the object and 30% inside the region of the object, but not closer than 10 pixels to a marker patch. This will form a negative class, that is more accurate and can fine-tune the training implicitly. Figure 6.1 shows some

samples of dataset patches for some classes of the Dwarf object.

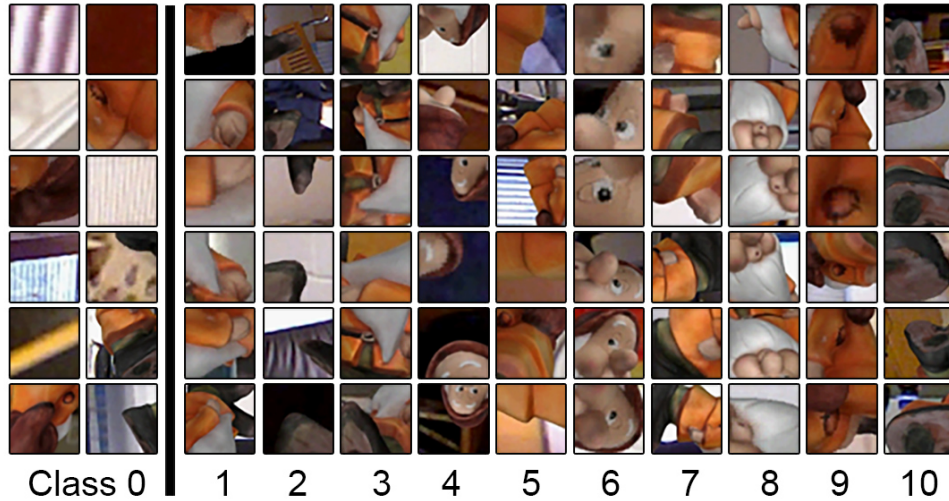


Figure 6.1: Some sample patches for classes 0, ..., 10 of the Dwarf object

All the patches alongside with the labels are written to the disk, and a mapping file including the file path, file names and their class labels is created. The estimated number of images for a 15 class marker is around 200K images in total. We then randomly assign 80% of the dataset to training and the rest to a test set.

We also create an evaluation set which is needed specially for the next section. We use the last 50 frames of a video sequence with their known pose. All patches are cut in equal size and written to the disk with an interval of 8 pixels. This creates 22,816 patches per frame of the video sequence with resolution 1024×768 .

In contrast to many pre-processing techniques for machine learning especially in computer vision, we do not need a color normalization for our RGB images. This is because patches are a small part of the object and their color is a distinctive feature in comparison with the other patches. However, besides transnational augmentations, we augment the dataset with illumination changes, such as brightness, contrast and saturation in a radius of $[-0.2, 0.2]$ of their original value. This is provided by Microsoft CNTK for the process of training images and is performed to all the stored images on the disk.

6.2 CNN architecture

In this section we introduce some basic concepts used in convolutional Neural Networks and introduce how a CNN can learn features through weights. We also give some insights on some widely used CNN architectures and their features to introduce our proposed architecture afterwards.

6.2.1 Introduction to CNN

Convolutional Neural Networks (CNN) are biologically inspired from the visual cortex, in which cells are sensitive to small sub-regions of the visual field called receptive fields [74].

The convolution of x and h written as $x * h$, is defined as the sum of the product of the two functions after one is reversed and shifted:

$$y[n] = x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k].h[n - l]$$

The convolution formula can be described as a weighted average of the function $x[n]$ at k where the weighting is given by $B(-k)$ simply shifted by amount n . Changes in k , emphasizes the weighting function on different parts of the input function [75].

A CNN was first suggested by Yann LeCun in 1998 for detecting hand written digits. Figure 6.2 is the CNN architecture proposed by LeCun in [76]. Some of the most important neuronally-inspired models in CNNs are the NeoCognitron [77], [78], HMAX [79], [80], and the LeNet-5 [76].

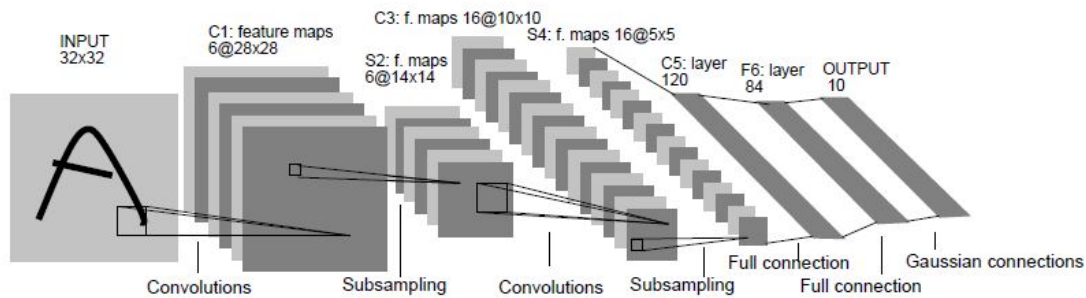


Figure 6.2: Architecture of the LeNet-5: A CNN used for digit recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical, image taken from [76].

Weights in a CNN are filters that are learned while training. They extract specific

features from input images using convolution based on the output given to a CNN. That is a filter studies successively every pixel of the image. For each pixel, it multiplies the value the pixel and values of the 8 surrounding pixels by the filter's corresponding value. Then it adds the results, and the initial pixel is set to this final result value. In a CNN features corresponding to the desired output such as edges, corners or even more complex structures are learned to be extracted by filters and correspondingly activated in the output image (the result of convolving an input image with a filter).

The activation function is usually an abstraction corresponding to the rate of action potential firing in the cell in biologically inspired neural networks [81]. Having a non-linear activation function, a two-layer neural network can be a universal function approximator.[2] The identity activation function however does not follow this statement. Having multiple layers with the identity activation function, the whole network can be shown as a single-layer model.

In a convolutional neural network units within a hidden layer are segmented into feature maps where the units within a feature map share the weight matrix, or in simple terms look for the same feature. To illustrate, the feature map is the output of one filter applied to the previous layer. A given filter is drawn across the entire previous layer, moved one pixel at a time. Each position results in an activation of the neuron and the output is collected in the feature map.

CNNs can employ three architectural ideas [76] which provide them with some extent of shift, scale and distortion invariance: local connectivity, parameter sharing and pooling or sub-sampling. To explain in more details, local connectivity is based on the idea of receptive fields. The thoughts behind it is the idea that images are stationary and there are patches of the image repeating along the image. Parameter sharing refers to feature maps in which the units share the same parameters and lead to a reduction in the number of parameters. There are two types of layers in a CNN: convolutional layers and sub-sampling or pooling layers. The former have a different number of filters for extracting desired distinctive simple features, such as edges. These filters are applied across the image and the result is a feature map that has the same size as the number of filters. Each feature map extracts the same feature regardless of its location in the image. Subsequently, a non-linear down-sampling is applied to the feature maps in the next layer.

There are different approaches for this non-linear down-sampling, where max pooling [76] and average pooling [82] are the most popular ones. Firstly, the image (each feature map) is partitioned into a set of non-overlapping rectangles. Then, if max pooling is the technique applied, in each sub-region the maximum value is selected as a representative of the whole rectangle. Otherwise, for average pooling, the average of all the pixels in

each rectangle is selected. Pooling provides translation invariance as small translations occurring in the same pooling region (rectangle) are neglected. The result of a CNN is an easy to train network benefiting from fewer parameters. Eventually, the network can be followed by a fully connected layer for classification of outputs. Figure 6.3 depicts the first layer of a convolutional neural network including extraction of feature maps and the pooling mechanism.

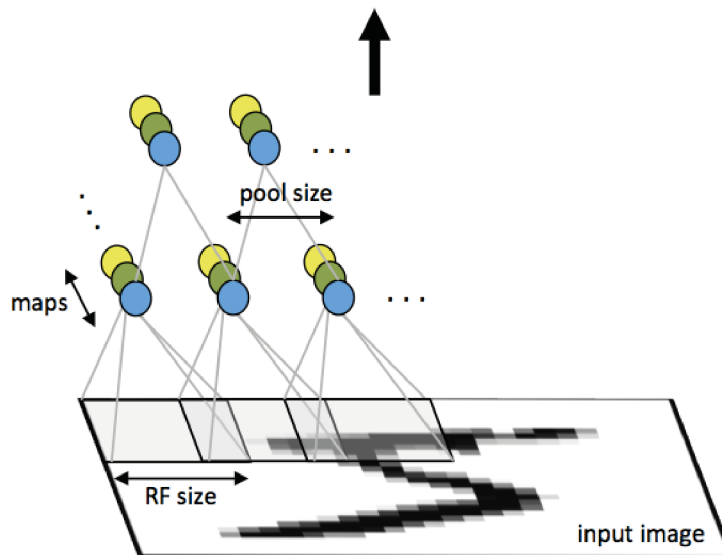


Figure 6.3: First layer of a convolutional neural network with pooling. Units with the same colour share weights, image taken from [83]

For training of a neural network, the concept of back-propagation is usually used. Back-propagation is an invaluable training technique for neural networks, in which the errors are propagated backwards through the last layer of the network (output layer). A common pitfall in most of machine learning implementations is over-fitting. Over-fitting is learning complexities or noise in the dataset rather than the general abstract model.

In order to prevent overfitting, dropout [84] can be employed as a helpful regularization technique. There could be several layers for convolution and pooling in the CNN. The last layer should be a fully connected layer and a pretrained neural network or any other kind of classifier (e.g. Softmax regression) could be used. We use Softmax as according to the empirical evaluation for CNNs [85], [86], [87], [88].

Softmax regression, is a linear classifier that uses log-probability distribution and cross entropy, which works well with CNNs. Another reason for using softmax is that the output of softmax provides the probability of each class and facilitates examining keypoint likelihoods for using the network. It is worth mentioning that the depth of the

network can be an important factor for better performance, since a deeper network is able to extract more abstract features. However, there is a trade-off among the level of abstractness, performance, training time and computations.

6.2.2 Proposed Architecture

In our pipeline we do not want to extract very abstract features as we have small patches of object markers which are mostly not extremely complex. However, patches have high deformation and rotation variance in different viewing angles, so a 5-layer CNN similar to LeNet-5 performs well. As Figure 6.4 depicts, in the first layer we use 15 feature maps of size 7×7 , then we have a pooling of size 2×2 . This is followed by a convolutional layer of size 12×12 and 5×5 with pooling layers of size 2×2 . The third pooling layer is connected to a fully connected layer with 2,048 nodes and then a softmax layer which is connected to the output nodes. Output nodes represent the marker class using softmax regression. The number of nodes in this layer is changing from object to object.

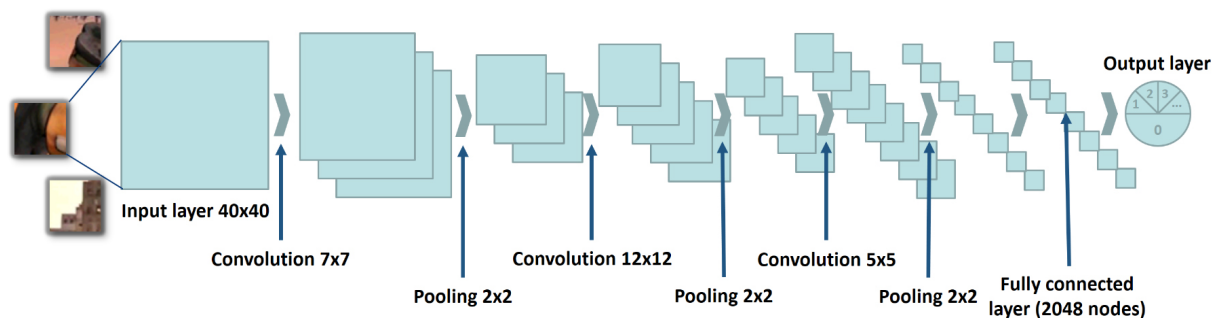


Figure 6.4: The architecture of our CNN, employing 3 convolutional layers followed by 3 pooling layers, and connecting to the output layer through a fully connected layer.

6.2.3 Training

For training deep neural networks, different methods based on gradient descent are used. Batch gradient descent (BGD) is a variant of gradient descent which uses whole dataset samples to compute the gradient and to update the nodes [89]. Since for large datasets, this is a very expensive method, which converges slowly for noisy input, stochastic gradient descent (SGD) can be an alternative [90]. Unlike BGD, SGD uses samples from the dataset one by one. The method updates the gradients and the node weights by a single sample and tends to converge faster for large datasets [91]. It also requires less memory to load the samples individually. On the other side, SGD can be prone to noisy inputs, since a single

image or input can be very noisy and therefore deceptive. To overcome this, mini-batch SGD [92] is proposed which takes the best of SGD and BGD and uses mini-batches of the dataset to implicitly suppress the noise by having a mean gradient and also performs efficiently and fast for large datasets and complex models. Here we also use mini-batch SGD with 10 batch samples and together with a decreasing learning rate. This trick can improve the number of epochs needed for convergence and prevents skipping the optimal solution [91].

For training our CNN we use Microsoft Cognitive Toolkit (CNTK) [93], which is an open-source deep learning toolkit. The training is done on an NVIDIA GeForce GTX 970 and a core i7 6700 CPU at 3.40GHz, an 8 Gigabytes of RAM and Windows 7 OS. For the network architecture, training and evaluation BrainScript code is written and interpreted through the CNTK BrainScript Network builder. The network is created on the GPU and the classes, their likelihoods, and softmax regression outputs, are written to the disk, which are later fetched for evaluation and further processing. The evaluation on error and time is mentioned on section 7.2.

6.3 Pose estimation procedure

So far we have described our marker selection and learning procedure. We build up a network to differentiate marker patches and markerless patches and distinguish different markers as well. In this section we want to investigate how we can use the network to find markers in an image from the output and build the point cloud to perform a pose estimation with a point cloud registration. The pose estimation also helps us evaluating our marker extraction and training procedures for different objects, especially in section 7.3.

Feeding the evaluation set, which are the patches of an image sequence containing the trained object, we will obtain the likelihoods for being a marker or markerless patch. By having n markers we have $n + 1$ classes as $\{0, 1, \dots, n\}$. Class 0 is the negative class or non-marker patch and the other n classes define the likelihood of their corresponding marker patch. For an image of size 1024×768 , we have 124×92 patches. Therefore a $124 \times 92 \times (n + 1)$ likelihood matrix is created for a single image. Figure 6.5 shows likelihood maps of a specific marker for an image pair.

Then for every marker a likelihood map is created for both left and right views and subsequently resized to the full image size. A Gaussian filter is applied to reduce classification errors and to find the location of markers at the pixel level. Therefore, we

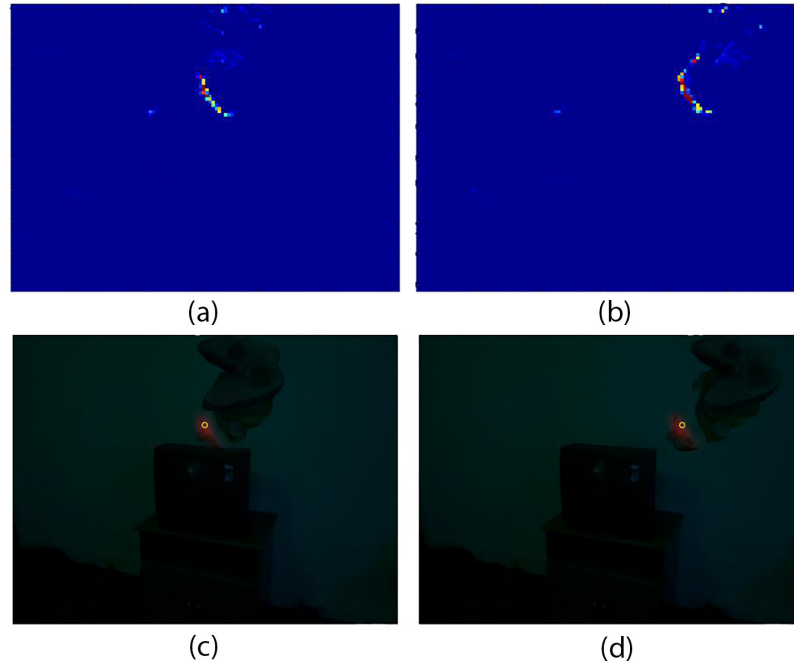


Figure 6.5: likelihood maps of a specific marker (a) and (b) for the left image and the right image, (c) and (d) the filtered heat map of likelihood and their final selection on left and right images

have $\{n \times 2\}$ likelihood maps. Once again stereo vision concepts are used, but here to filter some probable errors in the likelihood map. A correct marker should lay on the same epipolar line in the left and right image for a valid triangulation. Therefore we filter some outliers using epipolar line sweeping. We multiply all the pixel values on an epipolar line with the maximum value of pixels on the same epipolar line in the other image as illustrated in Figure 6.6. With this technique, we diminish the pixels that do not have a good correspondence. This will remove noise and helps us to find the maximum value in the whole image. The maximum values in the whole individual image after this stereo filtering is then calculated and it is checked again whether the two candidates are located on the same epipolar line. If this is true, the markers are accepted, otherwise the markers are rejected and therefore no triangulation is done.

The markers surviving the filtering will be triangulated and a point cloud is created from them. It is worth mentioning that because of the low resolution of the images and a probable error in the classification, some of the markers may be very noisy and therefore useless for the pose estimation. This is a reason why we need to estimate the pose using a combination of points that potentially produce less error. Doing this without any prior additional information is difficult. We use a reference marker point cloud to find the best combination of points to estimate the pose.

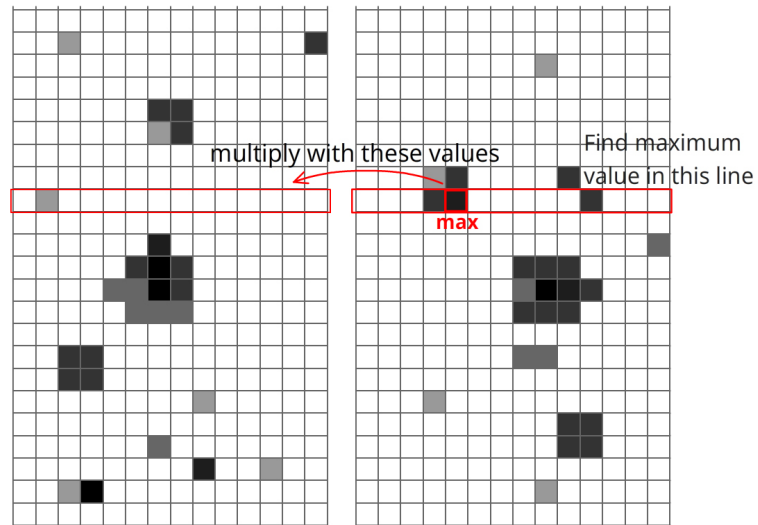


Figure 6.6: Stereo filtering by sweeping on the epipolar lines and multiplying all the values on left line to the maximum value in the right line.

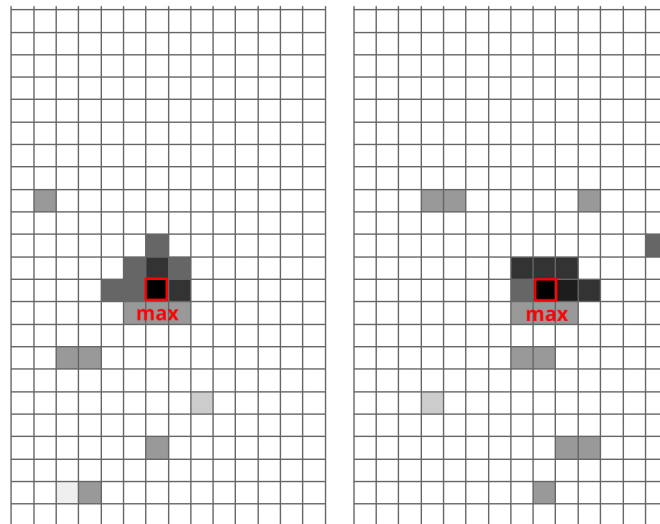


Figure 6.7: The output of the stereo filtering which suppresses noise and highlights the maximum on the whole image.

For this, create a binary number as a counter. The counter counts in the range of $[1, 2^n]$ (n is number of markers) and the binary basis of counter shows which markers are considered in this combination and which ones are not. The digit in binary basis represents its corresponding marker's presence. Another condition is that any combination of markers should consider at least 4 points, which is required for an acceptable point cloud registration. As we form combinations of markers, we can register the reference point cloud (P_{ref}) of the object to them. The translation and rotation of the reference point cloud to the evaluated point cloud (P_{eval}) is considered as the estimated pose, $[R, t]$:

$$P_{ref} = RP_{eval} + t$$

With our classified point cloud, we can also find the correspondent point in two point clouds and see which points are missing. But one major concern in registering the point clouds is finding the object's local axis or pivot point, for which the ground truth pose we have are applied to. Having a subset of the point clouds we can find their centroids to bring the point cloud to a same reference.

$$centroid_{ref} = \frac{1}{n} \sum_{i=1}^n P_{ref,i}$$

$$centroid_{eval} = \frac{1}{m} \sum_{i=1}^m P_{eval,i}$$

Figure 6.8 illustrate centroids for two point clouds with three points. There a few ways to find the optimal rotation for two sets of points. One of the easiest way is using Singular Value Decomposition (SVD) [94] which is a powerful algebraic tool for many numerical solutions. SVD decomposes a matrix into three other matrices such that:

$$[U, S, V] = SVD(E)$$

$$E = USV^T.$$

By the means of SVD, we find the rotation matrix using matrices U and V^T . Here we define a covariance matrix H for accumulating points and their pose. The matrix then is decomposed and the rotation is calculated[94].

$$H = \sum_{i=1}^n (P_{ref}^i - centroid_{ref})(P_{eval}^i - centroid_{eval})$$

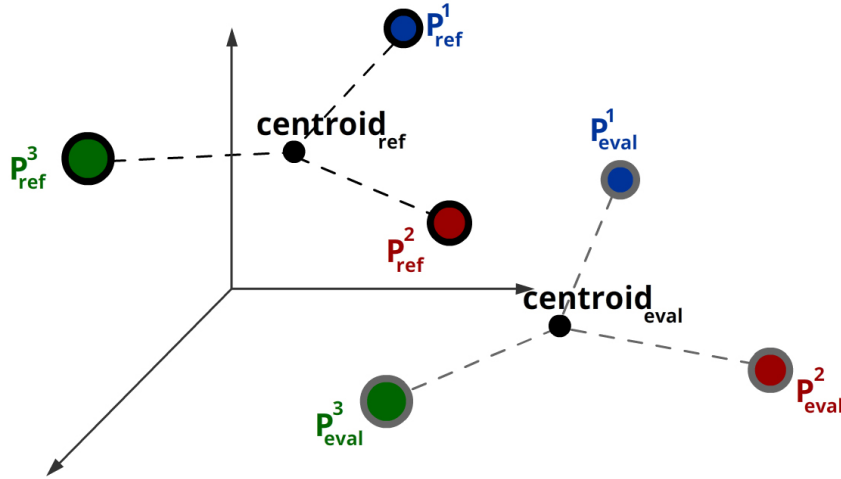


Figure 6.8: Two corresponding point clouds and their calculated centroids

$$[U, S, V] = SVD(H)$$

$$R = VU^T$$

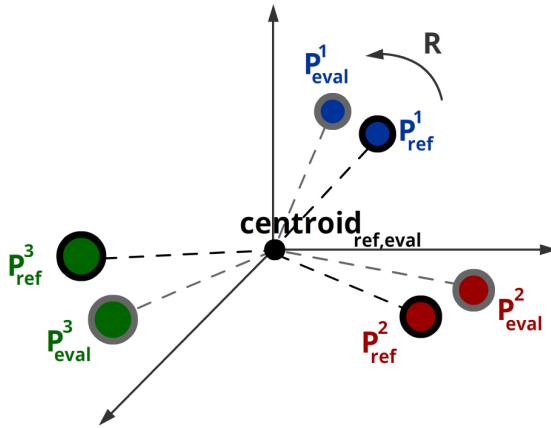


Figure 6.9: Calculating the rotation of point clouds in the same reference of centroids

As Figure 6.9 shows, the rotation R is estimated by bringing both the centroids in the same reference coordinates. After having R calculated, we can derive the value for t , the translation of the reference point cloud to the evaluated point cloud as follows:

$$t = -R \times centroid_{ref} + centroid_{eval}.$$

Furthermore, the pose RMS error err is calculated as follows:

$$err = \sum_{i=1}^n \|RP_{ref}^i + t - P_{eval}^i\|^2$$

We calculate the pose and the error for all the different combinations of markers in evaluation point cloud and choose the pose with the least error as our point cloud registration. More advanced searching methods such as Hungarian method [95] or RANSAC approaches [96] can be applied to find the combination and speed-up the pose estimation with the least error, which is beyond the scope of thesis and is considered as a future work.

The result of pose and the performance on different objects is to be discussed in section 7.3.

Chapter 7

Evaluation and Conclusion

In this chapter, we evaluate our pipeline from keypoint extraction to pose estimation. Firstly, we evaluate our keypoint extraction and selection method, and compare the outputs of different objects. Afterwards we look at training and test results for marker learning. Finally, we study the pose estimation results and errors for different objects to see the quality of the learning in an application. The chapter ends with a conclusion and an outlook for future work, where we discuss the potential and flaws of the proposed approach.

7.1 Marker extraction

Evaluating our marker quality is difficult, as there is no available marker or keypoint extraction benchmark in the literature for binocular images or the use of pose training sequences. This makes it even more difficult when we designed our marker selection procedure optimized for 6D pose estimation. Available techniques either use monocular images which demand relatively few information [2], or use 3D CAD models or depth maps which use more information [1]. Therefore, it would not be fair to compare our marker selection or training with them directly.







Accordingly, we adopt an empirical quantitative approach to evaluate our keypoints instead of a qualitative per keypoint one. Therefore, we test our algorithm by checking how robust the marker extraction and selection is. This is done by running the algorithm on different object video sequences. We have a sequence with 1000 image pairs which is divided in two sections with 500 images each. We run our marker extraction and filters with the same parameters and come up with a final selected batch of markers. The two obtained marker point clouds are then compared consecutively.

The number of markers in the first point clouds is seen in Table 7.1 where we set the value for the maximum number of markers allowed to 20. Two highly textured objects, Pillow and Camera get the maximum of 20 markers. On the other hand, the Earth object has 14 markers which can be a consequence of similarities of its texture. The Beets object also could not find enough markers, which is likely because of the plain texture on the caps and some areas around it. This object is considered as a relatively difficult one because of both the geometry and the number of markers. Finally, the objects Box and Dwarf have 17 markers, which places them in the middle range for the difficulty or quality in our dataset.

The table also shows the mean number of visible keypoints per frame. This term is a similar indication to our repetitiveness score before marker selection and shows the

average of keypoints counted in one frame. This term increases with the number of markers non-linearly. The bigger this average is for a particular object, the easier is the task of pose calculation using marker registration for this object. This factor does not change significantly between different objects, which indicates that the selected markers have similar repetitiveness score averages. With 14-20 markers per object we can expect to observe 6-9 markers in every stereo image pair.

Table 7.1: Quantitative marker evaluation for objects in the dataset.

Object	Number of markers	Mean # of visible keypoints per frame	RMS error [mm]	Outliers
 Dwarf	17	6.86	2.4354	2
 Pillow	20	7.34	2.3171	4
 Earth	14	7.84	3.2647	1
 Camera	20	8.86	2.8927	3
 Beets	15	6.82	3.1193	1
 Box	17	8.78	2.7622	2
Average	17.16	7.75	2.7985	2.16

To investigate the reproducibility of our algorithm we run the quantitative evaluation by registering the two markers point clouds we calculate. We do this by the means of ICP[18] method. The output of the registration shows how well these point clouds match with each other. The RMS error is a standard error metric for ICP registration as shown in Table 7.1 in millimeter for different objects. The resulting errors show a strong robustness for the marker extraction and selection procedure with an average of 2.7985 mm, a standard deviation of 0.365 and a median of 2.89 mm. Again the Pillow object shows the best quality of keypoints, which makes the object the best input for the marker training. Moreover, we also count the number of outliers by looking at the registration of the point clouds to check which points do not have a match in a 5 mm neighborhood. The outcome shows an average of 87% of the markers have a match in the other point cloud within this range.

The quantitative evaluation of markers shows similar errors for all the object, which point out the ability of the algorithm to generate for the extraction of keypoints for different objects. Although this is to some extent subject to how much salient keypoints and texture the object presents, how similar the keypoints are and how the distribution of the keypoints is, which implies low marker scores during our marker selection phase.

7.2 Marker training

In this section we evaluate the marker training and testing quality by the output of convolutional neural network. In section 6.1 we created a dataset of markers and in section 6.2 we proposed our architecture for marker patch classification. Now we want to study the error of marker training which can be an indication of the marker classes, their robustness and the algorithm's capability to accept distinct natural marker for an object.

As already mentioned we train our markers using mini-batch SGD for 60 epochs. The learning rate decreases by the epoch number to first train in coarse level and eventually converge finer to the optimal solution. The training time needed for an epoch is around 120 seconds on an NVIDIA GeForce GTX 970 GPU, Intel CORE-i7 6700 CPU and 16 gigabytes of RAM. The Figure 7.1 shows the learning error for the Dwarf object as an example. By entering the 10th epoch, the learning rates drops to 0.1 of the previous value so the learning converges faster and after the 30th epoch the learning error is changing very fine. This can be an over-fitting problem as well. The last 10 epochs do not show any improvement of the training error as it oscillates around 1.44%. The final training error for this object after 60 epochs is 1.48%.

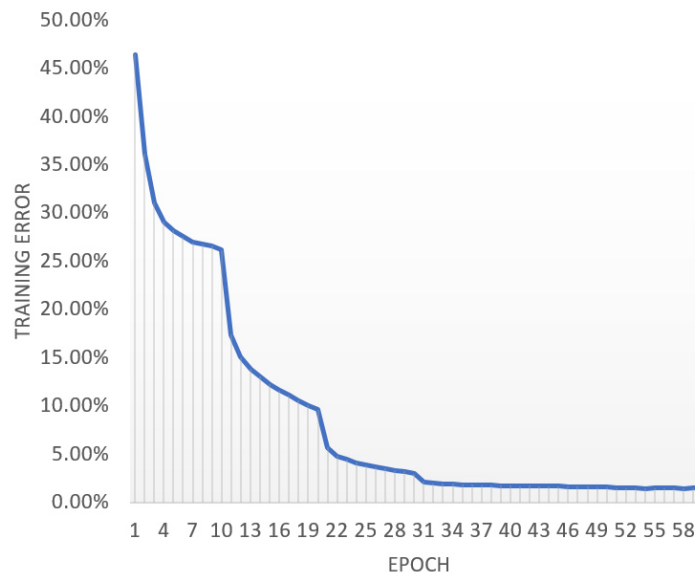








Figure 7.1: Training error diagram for Dwarf object

Table 7.2 shows training and test errors for all the objects of the dataset. The average error for training is 1.83% and 13.28% for testing. The reason for the error difference for these two can be because of the imperfectness of the dataset or over-fitting. The

reprojection error of markers can produce a lot of noise and also the complexity of the textures might affect this. In general it can be said that the test error for objects with a good quality of textures and markers show better results despite higher amount of markers. The Pillow object scores the best, as expected, with 11.11% and 20 markers. The Beets object scores the worst with 14.78% test error per 15 markers. Although the Camera object has the worst test error, likely because of mostly gray areas. It can have a relatively better marker evaluation output from a total of 20 markers.

Table 7.2: Marker training and test results for objects in dataset.

Object	Training error	Test error	Cross entropy
 Dwarf	1.48%	12.76%	0.0845
 Pillow	0.79%	11.11%	0.0729
 Earth	0.88%	11.88%	0.0769
 Camera	1.41%	15.13%	0.0981
 Beets	5.81%	14.78%	0.2109
 Box	0.64%	14.03%	0.0657
Average	1.83%	13.28%	0.1015

The table also includes the cross entropy for different objects, which is significant for training using softmax regression. This output indicates how well the marker patches can be differentiated from one another. The cross entropies for the Beets or the Camera object show that the majority of markers are from areas of the same color which makes also the training more difficult. On the other side, the Box object with the best cross entropy results possesses different textures with varying colors. This makes the markers well distinctive and therefore easy to be learned. The cross entropy is formulated as follows:







$$C = -\frac{1}{n} \sum x[y \ln a + (1 - y) \ln(1 - a)]$$

7.3 Pose estimation

Finally we evaluate our pipeline on the application of pose estimation. As mentioned in section 6.1, we have 30 stereo frame pairs with known pose which we feed them to the network. We estimate the pose using an exhaustive search including all possible combination of markers and finding the pose using variance of the data and a singular value decomposition. This process is apparently very slow and takes approximately 30 seconds

per frame on an Intel CORE-i7 6700 CPU. from marker detection to pose estimation. Finally, the calculated pose is compared to the ground truth pose. The distance error is found using the Euclidean distance while the angular error is calculated as the angle difference of the rotations. The average error for 30 frames can be found in Table 7.3.

Table 7.3: Pose estimation errors for objects in dataset.

Object	Angle error [degree]	Distance error [mm]	Rejection rate
 Dwarf	5.4862	7.0438	33.3%
 Pillow	3.8803	5.0497	10.0%
 Earth	4.3718	3.9942	50.0%
 Camera	6.2057	4.9467	23.3%
 Beets	8.2237	5.9485	20.0%
 Box	4.2041	5.9312	46.6%
Average	5.3953	5.4854	30.5%

Once again we see the quality of markers effecting the pose error. The Pillow object has 3.99° angle error which was already proven to have the best quality of markers. The Earth also has the least distance error which might be because of its very simple geometry. Other objects have acceptable angular error of average 5.39° with 3.6 standard deviation and 4.16° median and 5.65 mm distance error with 4.66 standard deviation and 3.98 mm median. The pose quality for the Beets object is relatively bad and as discussed in the previous section can be because of deficiency of good keypoints or training difficulties.

In the table, the rejection rate for different objects is also noted. The rejection rate shows the frames that did not provide a reliable marker registration. This is counted using the accumulated errors of pose in different combinations within the evaluated point cloud, which may also reject some fine poses. Once again, the Pillow get the best acceptance rate. The adjustment of the rejection rate is a subject of number of keypoints and their training quality and therefore having a constant rejection standard makes the rejection rates very variable. Nevertheless, the average rejection rate is around 30.5% with standard deviation of 15.68 and median of 26.9% which potentially indicates that there is almost one third of frames without enough accurate markers detected.

7.4 Conclusion

Marker extraction is a very challenging task when applied to different object with different textures and geometries. In this thesis we designed a pipeline to flexibly do the whole process of marker extraction and training. We use a sequence of stereo images of an object in different poses. We created a pipeline which finds some markers or keypoints, given a pair of images. These markers can be useful for 6D pose estimation using marker point cloud registration.

In order to achieve this we detect handcrafted features from 2D images and use stereo triangulation and epipolar geometry to triangulate the keypoints in 3D. Having a point cloud for each frame and its ground truth pose, we can use the full sequence to build up a point cloud in reference coordinates. Then we filter the noisy point cloud with thousands of keypoints. With subsequent mean-shift clustering and filtering we provide a robust algorithm to detect salient keypoints and textures. Within the clusters, we then locate a representative from which we form a candidate point cloud. Scoring and ranking strategies are used to have inclusiveness on different objects. The scoring criteria consider view angle range, distinctiveness of the keypoints and supports our need for a minimum marker quantity for the task of pose estimation as well.

The algorithm includes a learning scheme to train the markers and markerless patches on 2D reprojections. This makes the utilization of the trained network for marker detection easier and direct. We use a well-known CNN architecture to train markers and markerless patches as well as classifying the markers. Marker classification - although it increases the complexity of the training- will help us to be more confident in filtering the markers in case of classification errors. We also use a simple pose estimation technique and find an efficient way to evaluate a stereo frame by using a small amount of patches together with epipolar constraints to find the markers. The resulting marker training and evaluation for pose estimation shows that the proposed markers are robust and can be reused for different tasks. We evaluate our markers in a quantitative fashion and prove that their extraction is robust and reproducible with different image sequences. We also investigate the training and test errors for different objects, which is quite satisfying for the number of markers trained. The errors in different evaluations are various from object to object but keep a general standard.

The thesis also provides a discussion of this evaluation for different objects, which is an essential part if the proposed technique wants to be reused. The algorithm is inclusive in the marker extraction and scoring phase but the result of training is determinative on how well the markers could be distinguished. The results of the evaluation show the

pitfalls of marker extraction and learning: The objects that lack changing textures or colors do not get high quality and quantity of markers selected or trained.

7.5 Future work

As mentioned earlier in this section and throughout the thesis, there are some parts that can be improved or added. First of all, applying the algorithm on the real dataset is an upcoming work that will be carried out as an add-on to the algorithm. Evaluating how well the algorithm works with more noisy images and calibration imperfections and also less pose variations in angle and distance can change the results compared to the synthetic dataset. Some calculation and adaptations need to be applied to the work of the thesis to make the algorithm reusable for multicamera ground truth poses.

Another key improvement is the pose estimation procedure. The part -although not the main scope of thesis- is very slow and inefficient and likely prone to errors. A more sophisticated search method can also be used to try different combinations of markers for registration. Also a fine pose estimation approach such as ICP can be applied on top of the algorithm to improve accuracy. A method that can be used is the dual quaternion using pose iteration as [4]

The other improvement suggested is the use of dense point clouds to render the markers on top of the images. This may improve the imperfections and errors in the reprojection phase and therefore the marker dataset creation. Furthermore, a less problematic dataset can improve the training issues such as over-fitting. An approach that can work for very sparse and textureless areas would need to be considered in this regard.

Another application, where a similar approach can be applied, is finding and training keypoints for deformable objects. Any change of form in the object would influence the marker point cloud and therefore threatens the marker extraction or pose estimation. A strategy that can overcome this can be useful for many real applications such as hand or face pose estimation.

Moreover, the CNN architecture can have possible improvements, since the LeNet-5 is relatively old architecture. A fitting new architecture may also decrease the overfitting rate of the training and therefore improve the marker detection procedure. The markers can also be selected using feature descriptor integrated to our scoring algorithm to improve saliency of the markers. One possible 3D feature descriptor is the work of Salti et al. [38]

that performs descriptor learning.

Finally, as an application for 6D pose estimation, the work of the thesiDs can be applied to industrial AR. As we create a synthetic dataset with 3D models, we can use industrial CAD models to train markers of the industrial objects. The augmentations can then benefit the pose estimation using the markers detected. This procedures can be performed on all the CAD models, rate the quality of objects and then tag the marker information to the object's CAD model.

List of Figures

2.1	Sample object models and their depth maps from different angles in bottom, and their trained descriptor and object space in top-right. Later some images with complicated background are used to test the performance as shown in top-left. Figure from Wohlhart and Lepetit [1]	11
2.2	A network of markers from right is matched to a reference point cloud on the left. The system estimates the pose when enough matching markers are found. The illustration of the algorithm by Busam et al. [4].	13
2.3	A Siamese network architecture for matching corresponding patches, employing two CNNs with identical parameters and treating the CNN outputs as patch descriptors. Image taken from Simo-Serra et al. [3]	13
3.1	3D models of Rigid Pose Dataset, Figure from Pauwels et al. [57]	15
3.2	3D models for our synthetic dataset	16
3.3	Rotation in 3Ds max is applied to pivot point in object's local, Figure from Autodesk knowledge network [60]	17
3.4	Rendered images of 4 different poses in left and right views for the dwarf object.	19
3.5	Framos OTS camera mounted between Microsoft Kinect v2 and two RGB cameras.	20
3.6	The cardboard plane with 20 retro reflective markers attached to it	20
3.7	The set of dataset objects attached to the tracking plane.	21
3.8	A frame quadruple of the Truck object in a pose captured by four cameras, the markers on the filtered OTS images are shown in contrast	22
3.9	Four frames of specific pose for different objects in Kinect and RGB cameras	23
4.1	Pinhole camera model and projection of point x_C on image plane	25
4.2	Epipolar plane of a stereo view restricts the point stereo matching problem for correspondences to an intersecting line.	27
4.3	Schematic (a) and (b) are the left and right images before rectification. (c) and (d) are the left and right rectified images, where corresponding points can be found within a fixed line.	28

4.4	Different poses for Dwarf object on rendered dataset and its keypoint correspondences.	30
4.5	The 3D coordinates of a point with respect to focal length f and disparities d_L and d_R	31
4.6	The stereo cameras, triangulated keypoints and the object pose in reference to the world coordinates	32
4.7	The keypoint cloud for 6 different objects. Colors show the mean color of the extracted point on both the left and the right image	33
4.8	Mean-shift clustering algorithm of 2D point data. The algorithm terminates with the center of the clustering circle as the centroid of the points $Cp(i)$	34
4.9	The keypoint point cloud and their mean-shift cluster sizes in black blocks	35
4.10	The marker candidates point cloud shown with their mean color and number their repetition as their size	36
5.1	Ray-casting for rendering on the intersection of rays with the object surface	40
5.2	Finding the normal vector to a point using neighboring points and PCA.	41
5.3	Normal vectors (black) from the keypoint point clouds(red) in red for six different objects of the dataset	42
5.4	Keypoints accepted and projected to the left images for some samples of the Dwarf object.	45
5.5	Marker candidates point cloud and their scores on the left and the selected best markers on the right.	48
5.6	selected marker sets for different objects of the dataset	49
6.1	Some sample patches for classes 0, ..., 10 of the Dwarf object	52
6.2	Architecture of the LeNet-5: A CNN used for digit recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical, image taken from [76].	53
6.3	First layer of a convolutional neural network with pooling. Units with the same colour share weights, image taken from [83]	55
6.4	The architecture of our CNN, employing 3 convolutional layers followed by 3 pooling layers, and connecting to the output layer through a fully connected layer.	56
6.5	likelihood maps of a specific marker (a) and (b) for the left image and the right image, (c) and (d) the filtered heat map of likelihood and their final selection on left and right images	58
6.6	Stereo filtering by sweeping on the epipolar lines and multiplying all the values on left line to the maximum value in the right line.	59
6.7	The output of the stereo filtering which suppresses noise and highlights the maximum on the whole image.	59
6.8	Two corresponding point clouds and their calculated centroids	61

6.9 Calculating the rotation of point clouds in the same reference of centroids . 61

7.1 Training error diagram for Dwarf object 66

List of Tables

7.1	Quantitative marker evaluation for objects in the dataset.	65
7.2	Marker training and test results for objects in dataset.	67
7.3	Pose estimation errors for objects in dataset.	68

Bibliography

- [1] Paul Wohlhart and Vincent Lepetit. Learning descriptors for object recognition and 3d pose estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3109–3118, 2015.
- [2] Yannick Verdie, Kwang Yi, Pascal Fua, and Vincent Lepetit. Tilde: a temporally invariant learned detector. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5279–5288, 2015.
- [3] Edgar Simo-Serra, Eduard Trulls, Luis Ferraz, Iasonas Kokkinos, Pascal Fua, and Francesc Moreno-Noguer. Discriminative learning of deep convolutional feature point descriptors. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 118–126, 2015.
- [4] Benjamin Busam, Marco Esposito, Simon Che’Rose, Nassir Navab, and Benjamin Frisch. A stereo vision approach for cooperative robotic movement therapy. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 127–135, 2015.
- [5] Jan-Henry Seppenwoolde, Max A Viergever, and Chris J.G. Bakker. Passive tracking exploiting local signal conservation: the white marker phenomenon. *Magnetic resonance in medicine*, 50(4):784–790, 2003.
- [6] Hirokazu Kato and Mark Billinghurst. Marker tracking and hmd calibration for a video-based augmented reality conferencing system. In *2nd IEEE and ACM International Workshop on Augmented Reality, 1999. Proceedings.*, pages 85–94. IEEE, 1999.
- [7] Daniel Wagner and Dieter Schmalstieg. *Artoolkitplus for pose tracking on mobile devices*. Graz Technical University, 2007.
- [8] Iason Oikonomidis, Nikolaos Kyriazis, and Antonis A Argyros. Efficient model-based 3d tracking of hand articulations using kinect. In *BMVC*, volume 1, page 3, 2011.
- [9] Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew

- Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *2011 10th IEEE international symposium on Mixed and augmented reality (ISMAR)*,, pages 127–136. IEEE, 2011.
- [10] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.
- [11] Harald Wuest, Timo Engekle, Folker Wientapper, Florian Schmitt, and Jens Keil. From cad to 3d tracking enhancing & scaling model-based tracking for industrial appliances. In *International Symposium on Mixed and Augmented Reality (ISMAR-Adjunct), 2016 IEEE*, pages 346–347. IEEE, 2016.
- [12] Carl Yuheng Ren, Victor Prisacariu, Olaf Kaehler, Ian Reid, and David Murray. 3d tracking of multiple objects with identical appearance using rgb-d input. In *2014 2nd International Conference on 3D Vision (3DV)*,, volume 1, pages 47–54. IEEE, 2014.
- [13] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.
- [14] Jiangbo Lu, Hongsheng Yang, Dongbo Min, and Minh N Do. Patch match filter: Efficient edge-aware filtering meets randomized search for fast correspondence field estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1854–1861, 2013.
- [15] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.
- [16] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.
- [17] Stefan Leutenegger, Margarita Chli, and Roland Y Siegwart. Brisk: Binary robust invariant scalable keypoints. In *2011 International Conference on Computer Vision (ICCV)*,, pages 2548–2555. IEEE, 2011.
- [18] Paul J Besl and Neil D McKay. Method for registration of 3-d shapes. In *Robotics-DL tentative*, pages 586–606. International Society for Optics and Photonics, 1992.
- [19] Chris Harris and Mike Stephens. A combined corner and edge detector. Citeseer, 1988.
- [20] Paul Viola, Michael J. Jones, and Daniel Snow. Detecting pedestrians using patterns of motion and appearance. *International Journal of Computer Vision*, 63(2):153–161, 2005.

- [21] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2005. CVPR 2005.*, volume 1, pages 886–893. IEEE, 2005.
- [22] Luo Juan and Oubong Gwun. A comparison of sift, pca-sift and surf. *International Journal of Image Processing (IJIP)*, 3(4):143–152, 2009.
- [23] Edward Rosten and Tom Drummond. Fusing points and lines for high performance tracking. In *Tenth IEEE International Conference on Computer Vision, 2005. ICCV 2005.*, volume 2, pages 1508–1515. IEEE, 2005.
- [24] Alexandre Alahi, Raphael Ortiz, and Pierre Vanderghenst. Freak: Fast retina keypoint. In *2012 IEEE conference on Computer vision and pattern recognition (CVPR)*,, pages 510–517. Ieee, 2012.
- [25] Stefan Hinterstoisser, Stefan Holzer, Cedric Cagniart, Slobodan Ilic, Kurt Konolige, Nassir Navab, and Vincent Lepetit. Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes. In *2011 International Conference on Computer Vision (ICCV)*,, pages 858–865. IEEE, 2011.
- [26] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory*, pages 23–37. Springer, 1995.
- [27] Gabriella Csurka, Christopher Dance, Lixin Fan, Jutta Willamowski, and Cédric Bray. Visual categorization with bags of keypoints. 2004.
- [28] Pedro Domingos and Michael Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine learning*, 29(2-3):103–130, 1997.
- [29] Alex M Andrew. An introduction to support vector machines and other kernel-based learning methods by nello christianini and john shawe-taylor, cambridge university press, cambridge, 2000, 189 pp.,, 2000.
- [30] Vincent Lepetit and Pascal Fua. Keypoint recognition using randomized trees. *IEEE transactions on pattern analysis and machine intelligence*, 28(9):1465–1479, 2006.
- [31] Ajmal Mian, Mohammed Bennamoun, and Robyn Owens. On the repeatability and quality of keypoints for local feature-based 3d object retrieval from cluttered scenes. *International Journal of Computer Vision*, 89(2-3):348–361, 2010.
- [32] Mustafa Ozuysal, Michael Calonder, Vincent Lepetit, and Pascal Fua. Fast keypoint recognition using random ferns. *IEEE transactions on pattern analysis and machine intelligence*, 32(3):448–461, 2010.
- [33] Anna Bosch, Andrew Zisserman, and Xavier Munoz. Image classification using

- random forests and ferns. In *11th International Conference on Computer Vision, 2007. ICCV 2007.*, pages 1–8. IEEE, 2007.
- [34] John J Leonard and Hugh F Durrant-Whyte. Simultaneous map building and localization for an autonomous mobile robot. In *International Workshop on Intelligent Robots and Systems' 91. Intelligence for Mechanical Systems, Proceedings IROS.*, pages 1442–1447. Ieee, 1991.
- [35] Tomasz Malisiewicz, Abhinav Gupta, and Alexei A Efros. Ensemble of exemplar-svms for object detection and beyond. In *2011 International Conference on Computer Vision (ICCV)*,, pages 89–96. IEEE, 2011.
- [36] Mathieu Aubry, Daniel Maturana, Alexei A Efros, Bryan C Russell, and Josef Sivic. Seeing 3d chairs: exemplar part-based 2d-3d alignment using a large dataset of cad models. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3762–3769, 2014.
- [37] Andrew Richardson and Edwin Olson. Learning convolutional filters for interest point detection. In *2013 International Conference on Robotics and Automation (ICRA)*,, pages 631–637. IEEE, 2013.
- [38] Samuele Salti, Federico Tombari, Riccardo Spezialetti, and Luigi Di Stefano. Learning a descriptor-specific 3d keypoint detector. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2318–2326, 2015.
- [39] Daniel Wagner, Gerhard Reitmayr, Alessandro Mulloni, Tom Drummond, and Dieter Schmalstieg. Pose tracking from natural features on mobile phones. In *Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*, pages 125–134. IEEE Computer Society, 2008.
- [40] Yunpeng Li, Noah Snavely, Daniel P Huttenlocher, and Pascal Fua. Worldwide pose estimation using 3d point clouds. In *Large-Scale Visual Geo-Localization*, pages 147–163. Springer, 2016.
- [41] Chunhui Gu and Xiaofeng Ren. Discriminative mixture-of-templates for viewpoint classification. In *European Conference on Computer Vision*, pages 408–421. Springer, 2010.
- [42] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the icp algorithm. In *3-D Digital Imaging and Modeling, 2001. Proceedings. Third International Conference on*, pages 145–152. IEEE, 2001.
- [43] Pavlos Mavridis, Anthousis Andreadis, and Georgios Papaioannou. Efficient sparse icp. *Computer Aided Geometric Design*, 35:16–26, 2015.
- [44] Changhyun Choi and Henrik I Christensen. 3d pose estimation of daily objects using

- an rgb-d camera. In *2012 International Conference on Intelligent Robots and Systems (IROS)*,, pages 3342–3349. IEEE, 2012.
- [45] Stefan Hinterstoisser, Vincent Lepetit, Slobodan Ilic, Stefan Holzer, Gary Bradski, Kurt Konolige, and Nassir Navab. Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes. In *Asian conference on computer vision*, pages 548–562. Springer, 2012.
- [46] Andy Zeng, Shuran Song, Matthias Nießner, Matthew Fisher, and Jianxiong Xiao. 3dmatch: Learning the matching of local 3d geometry in range scans. *arXiv preprint arXiv:1603.08182*, 2016.
- [47] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11): 1231–1237, 2013.
- [48] Daniel Scharstein, Heiko Hirschmüller, York Kitajima, Greg Krathwohl, Nera Nešić, Xi Wang, and Porter Westling. High-resolution stereo datasets with subpixel-accurate ground truth. In *German Conference on Pattern Recognition*, pages 31–42. Springer, 2014.
- [49] Yann LeCun, Fu Jie Huang, and Leon Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 2, pages II–104. IEEE, 2004.
- [50] Lorenzo Pollini, Fabio Greco, Roberto Mati, Mario Innocenti, and Alessandro Tortelli. Stereo vision obstacle detection based on scale invariant feature transform algorithm. In *AIAA Guidance, Navigation and Control Conference and Exhibit*, page 6612, 2007.
- [51] Gee-Sern Hsu, Chyi-Yeu Lin, and Jia-Shan Wu. Real-time 3-d object recognition using scale invariant feature transform and stereo vision. In *4th International Conference on Autonomous Robots and Agents, 2009. ICARA 2009.*, pages 239–244. IEEE, 2009.
- [52] Björn Barrois and Christian Wöhler. 3d pose estimation of vehicles using stereo camera. In *Transportation Technologies for Sustainability*, pages 1–24. Springer, 2013.
- [53] Marco Esposito, Benjamin Busam, Christoph Hennersperger, Julia Rackerseder, An Lu, Nassir Navab, and Benjamin Frisch. Cooperative robotic gamma imaging: Enhancing us-guided needle biopsy. In *Proceedings of the 18th International Conference on Medical Image Computing and Computer Assisted Interventions (MICCAI)*, October 2015.

- [54] Scott Helmer, David Meger, Marius Muja, James J Little, and David G. Lowe. Multiple viewpoint recognition and localization. In *Asian Conference on Computer Vision*, pages 464–477. Springer, 2010.
- [55] Stefan Hinterstoisser, Vincent Lepetit, Slobodan Ilic, Pascal Fua, and Nassir Navab. Dominant orientation templates for real-time detection of texture-less objects. In *CVPR*, volume 10, pages 2257–2264, 2010.
- [56] Colin Rennie, Rahul Shome, Kostas E Bekris, and Alberto F De Souza. A dataset for improved rgb-d-based object detection and pose estimation for warehouse pick-and-place. *IEEE Robotics and Automation Letters*, 1(2):1179–1185, 2016.
- [57] Karl Pauwels, Leonardo Rubio, Javier Diaz, and Eduardo Ros. Real-time model-based rigid object pose estimation and tracking combining dense and sparse visual cues. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2347–2354, 2013.
- [58] TurboSquid Online resource. <https://www.turbosquid.com/>. Accessed: 2016-11-21.
- [59] Autodesk, Inc. 3ds max 2016. = <http://www.autodesk.com/products/3ds-max/overview>.
- [60] Autodesk knowledge network. <https://knowledge.autodesk.com/support/3ds-max>. Accessed: 2017-03-12.
- [61] Samuel R Buss and Jay P Fillmore. Spherical averages and applications to spherical splines and interpolation. *ACM Transactions on Graphics (TOG)*, 20(2):95–126, 2001.
- [62] Benjamin Busam, Marco Esposito, Benjamin Frisch, and Nassir Navab. Quaternionic upsampling: Hyperspherical techniques for 6 dof pose tracking. In *3D Vision (3DV), 2016 Fourth International Conference on*, pages 629–638. IEEE, 2016.
- [63] Antonio Torralba, Kevin P. Murphy, and Freeman. Context-based vision system for place and object recognition. In *ICCV*, volume 3, pages 273–280, 2003.
- [64] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence*, 22(11):1330–1334, 2000.
- [65] MATLAB. *version 7.10.0 (R2010a)*. The MathWorks Inc., Natick, Massachusetts, 2010.
- [66] Jiri Matas, Ondrej Chum, Martin Urban, and Tomás Pajdla. Robust wide-baseline stereo from maximally stable extremal regions. *Image and vision computing*, 22(10): 761–767, 2004.
- [67] G. Bradski. Opencv library. *Dr. Dobb's Journal of Software Tools*, 2000.

- [68] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA., 1967.
- [69] Ken Shoemake. Animating rotation with quaternion curves. In *ACM SIGGRAPH computer graphics*, volume 19, pages 245–254. ACM, 1985.
- [70] Andrew S Glassner. *An introduction to ray tracing*. Elsevier, 1989.
- [71] Michael Deering, Stephanie Winner, Bic Schediwy, Chris Duffy, and Neil Hunt. The triangle processor and normal vector shader: a vlsi system for high performance graphics. In *Acm siggraph computer graphics*, volume 22, pages 21–30. ACM, 1988.
- [72] Ian Jolliffe. *Principal component analysis*. Wiley Online Library, 2002.
- [73] Jerome H. Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software (TOMS)*, 3(3):209–226, 1977.
- [74] David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex. *The Journal of physiology*, 160(1): 106–154, 1962.
- [75] Steven W Smith et al. *The scientist and engineer’s guide to digital signal processing*. 1997.
- [76] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278–2324, 1998.
- [77] Kuniyiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.
- [78] Kuniyiko Fukushima, Sei Miyake, and Takayuki Ito. Neocognitron: A neural network model for a mechanism of visual pattern recognition. *IEEE Transactions on Systems, Man and Cybernetics*, (5):826–834, 1983.
- [79] Garrick Orchard, Jacob G. Martin, R. Jacob Vogelstein, and Ralph Etienne-Cummings. Fast neuromimetic object recognition using FPGA outperforms GPU implementations. *IEEE Transactions on Neural Networks and Learning Systems*, 24(8):1239–1252, 2013.
- [80] Thomas Serre, Lior Wolf, Stanley Bileschi, Maximilian Riesenhuber, and Tomaso Poggio. Robust object recognition with cortex-like mechanisms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(3):411–426, 2007.

- [81] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, 1989.
- [82] Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In *International Conference on Artificial Neural Networks*, pages 92–101. Springer, 2010.
- [83] Supervised Convolutional Neural Network. <http://ufldl.stanford.edu/tutorial>. Accessed: 2017-01-30.
- [84] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.
- [85] Li Deng. A tutorial survey of architectures, algorithms, and applications for deep learning. *APSIPA Transactions on Signal and Information Processing*, 3:e2, 2014.
- [86] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
- [87] Tara N. Sainath, Brian Kingsbury, A.R. Mohamed, George E. Dahl, George Saon, Hagen Soltau, Tomas Beran, Aleksandr Y. Aravkin, and Bhuvana Ramabhadran. Improvements to deep convolutional neural networks for LVCSR. In *2013 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pages 315–320. IEEE, 2013.
- [88] Joost van Doorn. Analysis of deep convolutional neural network architectures. 2014.
- [89] Cheng-Tao Chu, Sang Kyun Kim, Yi-An Lin, YuanYuan Yu, Gary Bradski, Andrew Y Ng, and Kunle Olukotun. Map-reduce for machine learning on multicore. In *NIPS*, volume 6, pages 281–288. Vancouver, BC, 2006.
- [90] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT 2010*, pages 177–186. Springer, 2010.
- [91] Sebastian Ruder. An overview of gradient descent optimization algorithms. <http://sebastianruder.com/optimizing-gradient-descent/>. Accessed: 2017-03-07.
- [92] Vasilis Vryniotis. Machine Learning Blog and Software Development News. <http://blog.datumbox.com/tuning-the-learning-rate-in-gradient-descent/>. Accessed: 2017-03-07.
- [93] Frank Seide and Amit Agarwal. Cntk: Microsoft’s open-source deep-learning toolkit.

- In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2135–2135. ACM, 2016.
- [94] Nghia Ho. Finding optimal rotation and translation between corresponding 3D points. <http://nghiaho.com/>. Accessed: 2017-03-08.
- [95] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.
- [96] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.
- [97] Alberto Crivellaro, Mahdi Rad, Yannick Verdie, Kwang Moo Yi, Pascal Fua, and Vincent Lepetit. A novel representation of parts for accurate 3d object detection and tracking in monocular images. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4391–4399, 2015.