Slovak University of Technology in Bratislava

Faculty of informatics and information technologies

FIIT-5208-5669

Bc. Marek Jakab

Visual detection, recognition and tracking of three-dimensional objects

Master thesis

Degree Course: Information systems

Study program: 9.2.6 Information systems

Department: Department of Applied Informatics, FIIT STU, Bratislava

Supervisor: Ing. Vanda Benešová, PhD.

Date: May 2015

Slovenská technická univerzita v Bratislave
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Vizuálna detekcia, rozpoznávanie a sledovanie trojrozmerných objektov

Študijný program: Informačné systémy
Autor: Bc. Marek Jakab
Vedúci bakalárskej práce: Ing. Vanda Benešová, PhD.
2015, Máj

Proces detegovania objektov z obrazu kamery patrí do oblasti počítačového videnia, ktorý sa stále vyvíja. Víziou algoritmov pre rozpoznávanie objektov je rozpoznávanie veľkého množstva objektov v reálnom čase pri vysokom rozlíšení obrazu. S príchodom zariadení pre zaznamenávanie hĺbky je možné rozpoznávanie objektov rozšíriť o ďalšiu dimenziu a rozpoznávať 3D objekty.

Pri procese rozpoznávania objektov sa stretávame s nemalým množstvom problémov. V prvom rade sa snažíme detegovať 3D objekty na 2D scéne kamery s pridanou informáciou o hĺbke. Objekt môže byť natočený pri snímaní kamerou v rôznych uhloch, škálach a otočeniach. Na scénu pôsobia rôzne svetelné podmienky, časť objektu môže byť zakrytá, alebo môžu nastať nežiaduce vplyvy zo strany hardvéru ako napríklad šum a rozmazanie obrazu.

V diplomovej práci sa venujeme bližšie problému rozpoznávania objektov pomocou lokálnych deskriptorov a ich možnostiach rozšírenia rozpoznávania pre 3D objekty na základe RGBD obrazu.

Slovak University of Technology Bratislava

FACULTY OF INFORMATICS AND INFORMATION TECHNOLOGIES

Visual detection, recognition and tracking of three-dimensional objects

Degree Course: Information systems

Author: Bc. Marek Jakab

Supervisor: Ing. Vanda Benešová, PhD.

2015, May

Process of object recognition from the camera image belongs to still evolving study field of computer vision. The vision for object recognition is to be able to recognize large dataset of objects at high resolution images within real time. Using the devices for acquiring depth data we are able to extend the field of object recognition with another dimension and detect 3D objects.

In our research we are still facing with large numbers of issues. Even with depth sensors we still do not possess full 3D information about the object. In addition, object can be viewed from different angle, scales and rotation according to the object reference. Various light conditions can affect the scene as well as partial occlusions of the objects are possible and make recognition harder. Additional undesirable impacts are caused by hardware like noise or image blur.

In the master thesis we focus our research on the problem of object recognition using the methods of local descriptors and their possibilities to be extended for 3D objects based on the RGBD image.

Affidavit

I hereby declare that this master thesis has been written only by undersigned with the aid of Ing. Vanda Benešová, PhD. All sources have been stated in the reference list.

.....

Bc. Marek Jakab

Bratislava, May 2015

Acknowledgement

I would like to express my gratitude to my supervisor Ing. Vanda Benešová, PhD. for the useful comments, remarks and engagement through the learning process of this master thesis which enlarged my knowledge about given topic.

Bc. Marek Jakab

Content

1. Introduction

Visual detection and recognition of objects placed in an image or on video belongs to one of the most challenging tasks in recent past. Big amount of methods and algorithms were presented with the aim of precision and speed, compared to inevitable advance of hardware and software. Field of computer vision, to which part of object recognition belongs to, is nowadays one of most active and developing field in informatics. Thanks to research in this field and new hardware, we are able to extend some well-known methods and come up with new ideas, which can contribute to and move the ladder of the hard task of object recognition.

In this thesis we will discuss the problem of object recognition, compare and get acquainted with object recognition algorithms and image segmentation. We will discuss possibilities of recognition based on bottom-up models. The issue of finding object on an image sequence is quite hard thanks to countless possible views of an image. Not only the angle of view, but distance and light variations are making this task harder to compute and make them possible on real time. Not mentioned the distortion of an image caused by hardware, such as noise which occurs on captured images.

In our work, we will enhance the object recognition with a depth information. Most of currently known bottom-up methods are based on classic RGB (Red, Green, Blue) images where you cannot determine the real distance of object from camera. By using hardware like sensor Kinect, we are not only able to capture RGB image but also depth image. With this sensor we are able to see real surface of object and can partially rebuild the 3-dimensional object. Providing that information we are able not only to enhance current methods of object detection, but we are able to make segmentation of an object based on depth information and therefore reduce area of an image we need to recognize on.

From wide area of object detection methods, we will take closer look at the method of local descriptors. Our research focus on the implementation of real time algorithm which will use depth information from available sensors like Kinect 2. Depth information will be used as another value to enhance the recognition.

2. Analysis

Object detection is one of most important tasks of computer vision and image processing. We usually call objects same name, even when their appearance is completely different. For example we can recognize a car even when there are hundreds of types of vehicles. They have different shape, different color or look. We need to take this into consideration to be able to detect objects. In general, we divide objects into two categories[2]:

- generic objects - class recognition (generally a car)
- specific objects - instance recognition (specific type of a car)

Generic object recognition is based on describing the shapes of an object trained on several instances of same class, where some general features which are most of the time same on all instances are used. We create sort of a statistical model. This refer more to classification issues.

In our work, we focus on specific object recognition algorithms. In general, we need an image sample of the object from which we are going to extract information and store it for later use in our recognition. Next, as we capture image from camera we are going to use that information to get results, if image contains our trained object or not. This sounds like an easy task, but it is not. There are many issues we are encountering during this process.

One of the main issues is that we are trying to represent 3 dimensional objects from real world in 2D image. Objects usually do not look the same from different sides, even when humans are still able to recognize them. Using Kinect sensor we are able to partially recognize object shape, but we are still not able to have full 3D information of it. To take care of this issue, the simplest way is to learn about object from sequence of images taken from different views.

However, 3D object represented on 2D image is not the only one issue we are facing during the process of object detection. There are several more problems regarding the specific camera or depth sensor, time of the day or current view of an object we are trying to recognize. We can identify those issues as below: [2]

- **Light variance** - Considering image as a matrix of RGB values, different light source can make a negative impact on recognition. For example, photograph taken of the same scene in different time represent completely different images if we compare the values of each RGB channels in both images.
- **Scale and distance** - If we capture an object from various distances it will have different size in the result image. Also values from depth image are different. We need to consider it in our result algorithm with values, which will be invariant to the change of distance and therefore also scale.
- **Rotation** - Object will probably not be in the same standing position in every captured image. Rotated object differs from template and we need to make our algorithm rotation invariant if we are going to recognize trained object from various rotated positions.
- **Axial rotation** - Object can be captured from different angles and can look different. In our method we should consider possibility of perspective transformed image regarding to original training image.
- **Noise** - Distortion of an image not present in real environment. It is caused by various quality of sensors and cameras.
- **Occlusion** - In a lot of images, we do not see object in a whole. There are many situations where we are able to see only part of it.
- **Object pose** - Considering non rigid object we are trying to recognize.

The first and very easy conclusion for object recognition was a holistic method. Main idea was to create a histogram of color intensities of the specific object. In such a histogram we can see a similar pattern for the same object detected on an image. We can think of a histogram as high dimensional space and use metrics for distance measurement like Euclidian distance. However such a methods are not quite robust, need to be focused on certain object optimally without any background and still can lead to false positive results.

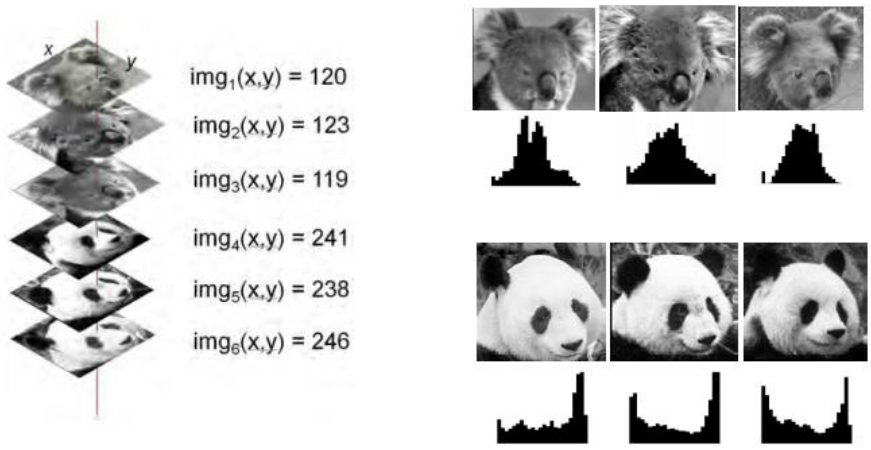


Figure 2.1. Histogram representation of selected images. [2]

Statistical methods as told above was used for further analysis and multiple improvements were made. In 1992, Turk & Pentland [4] in their work with face images were able to reduce such an analysis into less dimensional vector using PCA (Principal Component Analysis) creating an eigenface.

In 1995 this idea was used and improved by Murase & Nayar [5]. They created an algorithm which were capable of recognizing about 100 objects in real time. All mentioned methods had one major issue. For example using face dataset, any change in pose could lead to failed detection using global representation. Positive thing about this method is that the rotation invariance do not need to be take into consideration as intensity histogram does not change after various rotations of the object.

Mentioned methods were based on global representation of an image and objects. The main issue there was with occlusions which were detected on an image, large changes in viewpoints or with deformable objects. Therefore new method was proposed based on local features also known as key-points, and local descriptors.

2.1. Local descriptors

Discovery of local features and descriptors made a huge change in the research of object recognition. Thanks to local descriptors, we were able to develop object recognition methods which were more efficient and also robust under variety of viewing conditions or occlusions. The main task of local descriptors is to recognize whether the various features

extracted from the same object are presented in an image, how are they oriented and where are they located. The main task of the local descriptor methods are as follows: [2]

- Extract local features both on training image and test image
- Match stored features with detected features in test image
- Verify the correspondences of features. Compute geometric configuration

Using local descriptor methods, features can be extracted with both scale and rotation invariant manner. As we extract selected features independently from others, we are able to translate them to default scale and rotation using for example gradient direction or edge orientation based on certain pixels around the features. In comparison to the global representation we are able to save expensive computing time as we do not need to rotate or scale whole image to achieve desired invariance.

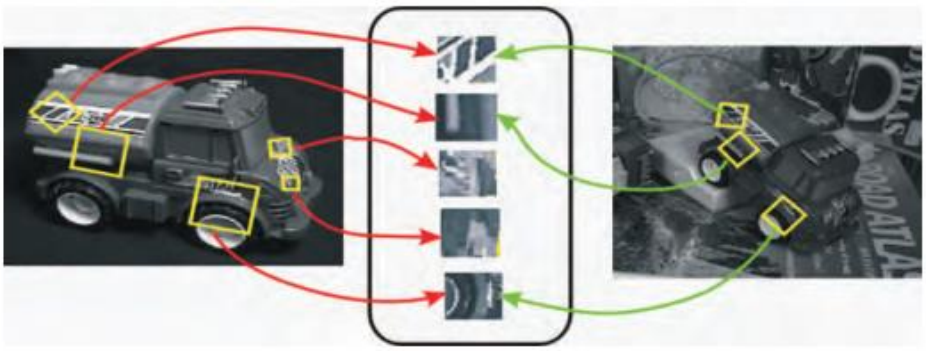


Figure 2.2. Example of object features (key-points). [2]

Using small patterns to form features can still lead to easy mismatch of features and therefore geometric verification need to be done. In general we find out if matched features are able to create convex rectangle with no lines intersection as it could not exist in real world.

2.1.1. Feature extraction

Main goal of successful object recognition is to detect significant local invariant features, with properties as follow [2]:

- Can be found repeatedly in two images showing same object
- Are detected in an image with certain precision
- Are present and can be found in various viewpoints of an image
- Are enough representative for selected object

- Are distinct from another object features
- There are enough features per object so that object can be matched under partial occlusion

To pass to the process of feature description, first a set of distinctive features must be extracted. Feature is presented as key-point localized in an image. Next, we define a region around each detected key-point in a certain manner to achieve scale and rotation invariance. To form local descriptor and therefore describe selected key-point we use information in defined (depends on the certain descriptor) region. But first, the content of the region should be normalized to achieve invariance to varying light conditions. Also the rotation invariance can be achieved simply by finding dominant orientation of selected pattern. According to the orientation found we can rotate selected key-point.

Best adept for key-points are those points in an image where signal changes in two directions. Imagining uniform region or a single line, we are not able to distinguish points from their neighbors. In contrary, corners or non-uniform regions suits our case well. We present a list and a description of feature detectors which can be used in feature extraction.

Hessian detector

The Hessian detector [2] is based on a second derivate matrix called Hessian. It looks for a key-points that are strong enough in two orthogonal directions. The cons of this method is that those operations are sensitive to a noise present in an image. To prevent bad key-point detection we usually smooth the image using Gaussian blur. Besides corners, Hessian detector can detect also responses at places with strong texture.

Harris detector

Harris detector was explicitly designed for geometric stability. It defines key-points to be *"points that have locally maximal self-matching precision under translational least-squares template matching" (Triggs 2004)."* [2] This detector looks for corners and is less responsive on textured areas. Also Harris detector is considered to be more precise than previous mentioned Hessian detector. Nevertheless, both Harris and Hessian detectors are not working well on different image scales and therefore they are not taken as scale invariant detectors.

Laplacian of Gaussian (LoG)

Laplacian of Gaussian [6] detector belongs to the family of scale invariant detectors. First solution to scale invariance was to continuously scale image around detected key-point. However this process is expensive related to speed and computing power. Instead a signature function is evaluated and plotted as a function of the neighborhood scale. To help find corresponding scales and scaling factor we divide two local maxima values. Laplacian of Gaussian detector is a blob-like feature detector based on previous theory that search for a scale space extrema.

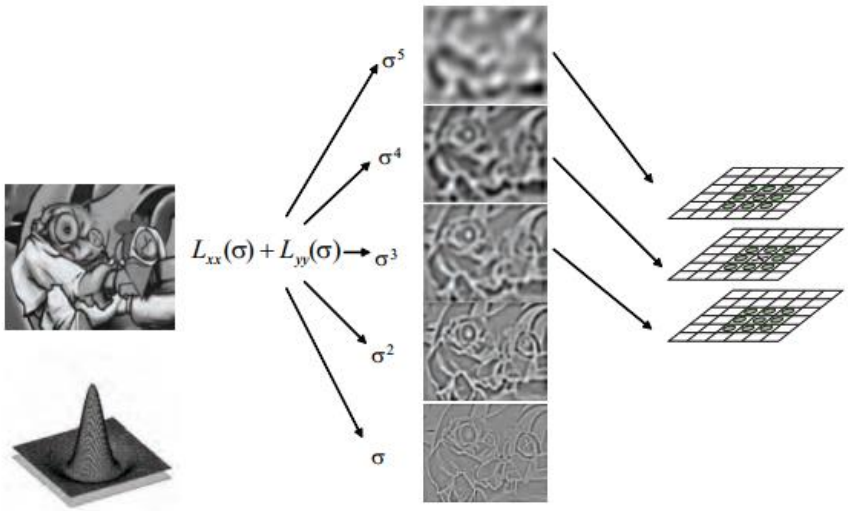


Figure 2.3. Illustration of Laplacian of Gaussian detector. [2]

Difference of Gaussian (DoG)

Difference of Gaussian detector [7] is a good approximation for LoG detector but faster. It searches for 3D scale space extrema of the DoG function. The approximation is based on subtraction of two adjacent scale levels of Gaussian pyramid. Points of local extrema determines the position and size of detected key-points.

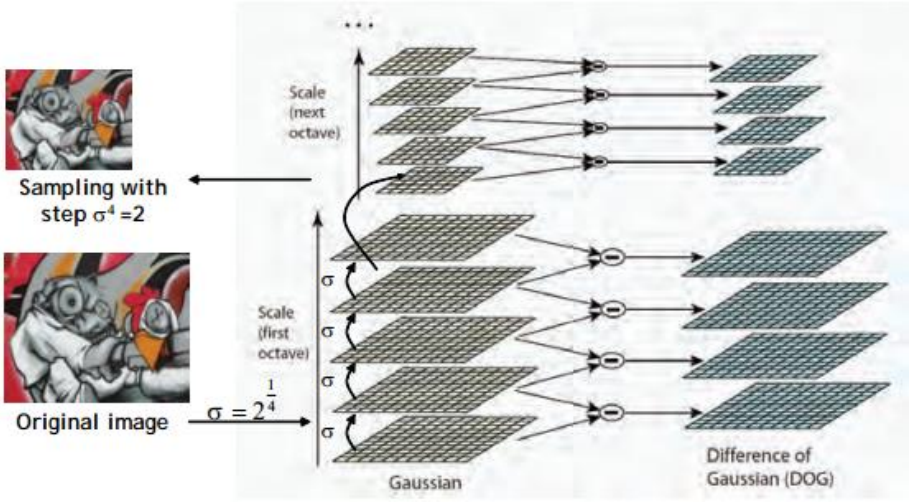


Figure 2.4. Illustration of Difference of Gaussian detector. [2]

The Harris-Laplacian Detector

This detector [8] [9] was originally created for increased power compared to Laplacian or Gaussian detectors. As the name of this detector tell us, it combine Harris detector with added scale invariance from next detectors. Like original Harris detector, this detector looks mostly for corners but is more invariant to scale, image rotation, camera noise and illumination. Harris-Laplacian detector used to however detect lower amount of key-points and therefore partial occlusion of an object could be issue during detection. Because of this issue, detector went through slight changes few years after it was introduced.

Maximally Stable Extremal Regions (MSER)

The detector [10] [11] we are going to describe now brought another improvement to the issue of key-point detection. Previous described detectors suits well in a lot of situations - scale, rotation, illumination invariance or camera noise, however there is still one problem that occurs in the object detection phase - the object viewpoint. Previous detectors proven to be not suited enough for instances, where object was captured from different view and angle, as it was using still the same pattern for detection. In those cases the perspective transformation of an image appeared to be too much expensive.

Next MSER detector was created to be able to detect affine covariant regions. First, the MSER detector applied watershed transformation to the image with the aim to extract homogenous intensity regions. Those regions are stable enough under changes of viewpoints of an image or other image distortion and can be considered as good features to describe.

Features from Acelerated Segment Test (FAST)

Detector FAST [12] [13] was proposed as to be very fast corner detector suitable for real time object detection and tracking. It is not based on first or second derivate of values like LoG or DoG. It simply compares the intensity values of 16 pixels forming a circle around key-point candidate. Corner at point *P* in the figure 5 is declared as a key-point if certain amount of pixels in a circle has bigger or lesser intensity with given threshold as point *P* in the middle of the circle. This number vary depending on the FAST algorithm itself as there are several more variations of this method. In first FAST detector, the number was set to 12 consecutive pixels. FAST detector had to pay for its speed with the lack of rotation invariance.

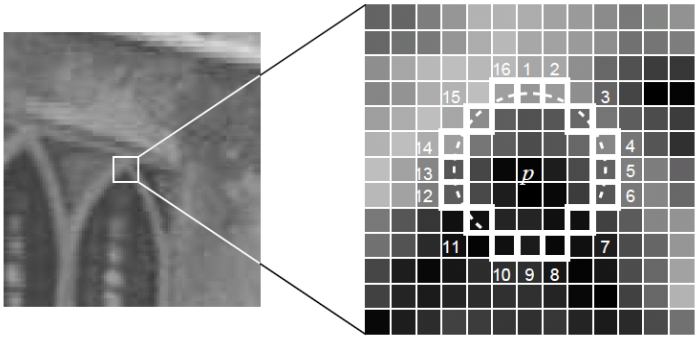


Figure 2.5. FAST key-point detector with circle used for comparison [12]

2.1.2. Feature description

Now that we have detected key-points we need to describe selected features as in next step we are going to compare them. To create such a descriptor we take content around selected key-point and translate pixel values into vector of numbers. Pattern which pixels to take into consideration and the content of vector vary for different descriptors. We divide descriptors into 2 categories: [2]

- numerical descriptors (float type)
- binary descriptors

First descriptor vectors contain float type values in their vectors. Later on, binary descriptors were introduced in aim for better computing speed. We are going to describe few well known descriptors which are promising in our case of real time 3D object detection and tracking.

Scale Invariant Feature Transform (SIFT)

Although SIFT descriptor [14] is old, it is still considered to be as one of the best local descriptors. It is still used in a lot of projects and also for comparison with other methods. SIFT descriptor was introduced by Lowe in 1999 and later 2004. For key-point detection, SIFT came up with his own detector also called SIFT which were based on Difference of Gaussian. It belongs to numerical local descriptors and to match pair of descriptors we can use Euclidean distance.

"The SIFT descriptor aims to achieve robustness to lighting variations and small positional shifts by encoding the image information in a localized set of gradient orientation histograms" [2]

To create SIFT descriptor vector we first create image gradient with magnitude and orientations around detected key-point. The size of the pattern grid is 16 x 16 pixels. Next, we divide the selected pattern into smaller 4 x 4 pixels grids (sum of 16 windows), each representing a gradient orientation histogram with 8 bins created from corresponding 16 x 16 pattern. During creation of descriptor gradient histogram is created after Gaussian weighting function, as we want the pixels closer to the middle of key-point to have bigger impact on the result vector.

Considering 4 x 4 grid, each containing histogram of orientation with 8 bins we create a descriptor vector with the size of 4 x 4 x 8 = 128. After descriptor is created we normalize values to achieve light invariance and unit length.

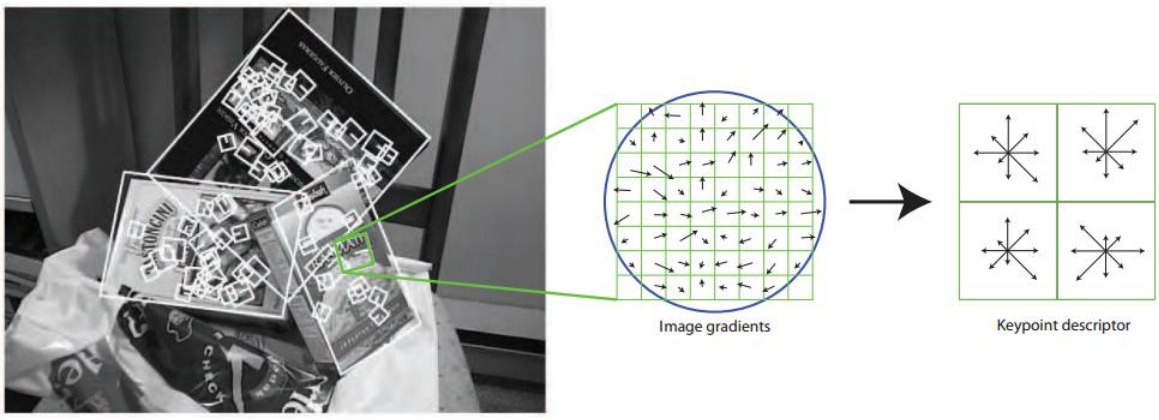


Figure 2.6. Process of forming SIFT descriptor. [2]

Speed Up Robust Features (SURF)

The main reason behind the creation of local descriptor SURF [2] [16] was to provide as good robustness for object detection as SIFT descriptor but in less computational time. Instead of Gaussian derivate, SURF descriptor is based on box filters also known as Haar wavelets. It is an approximation of derivate used in SIFT, but can be easier evaluated by using integral images. Also because of this method, they do not had to use Gaussian pyramid for scale invariance. The common part with SIFT descriptor is that it divides the area around key-point into 4 x 4 grid regions. Instead of making histogram with 8 bins it computes summary statistics, resulting in overall 64 dimensional vector instead of 128 for SIFT. SURF descriptor were also implemented on graphic card using CUDA technology which is known as GPUSURF.

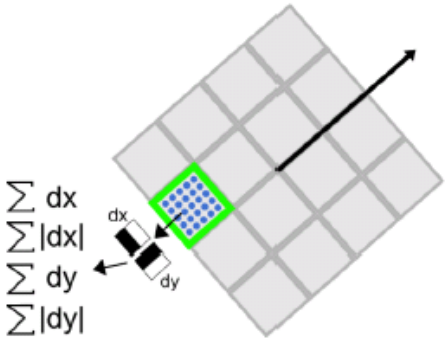


Figure 2.7. SURF descriptor visualization. [16]

Binary Robust Independent Elementary Features (BRIEF)

Numerical (float) descriptor vectors were in general slow to compute and match. Local descriptor BRIEF [17] was intended to change those values into binary strings. In paper were presented that those binary strings can be obtained directly from the image patches. They compute their binary vector in a way of comparison of intensities between two corresponding points. Those points are taken from selected lines they chosen to be their pattern.

Binary descriptors were introduced mostly because of their main advantage over numerical descriptors. To match binary descriptors, lower cost Hamming distance can be computed instead of Euclidean distance. On modern CPU's, Hamming distance can be computed much faster using XOR operations.

To create descriptor vector, authors of BRIEF descriptor have chosen a sample of lines and compare the intensity values of their end points. In next figure we show templates which were tested in the aim of best matching capability.

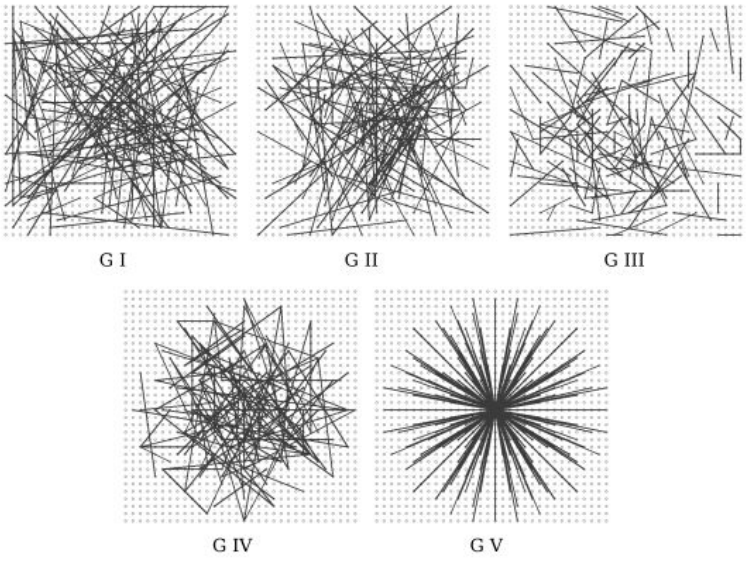


Figure 2.8. Different pattern used for BRIEF evaluation. [17]

For each template recognition rate was computed. The next figure show us the result of their evaluation. The random selected lines in a template have yield better results than generated templates. That is the reason the template pattern for filling the descriptor in BRIEF is generated from random lines.

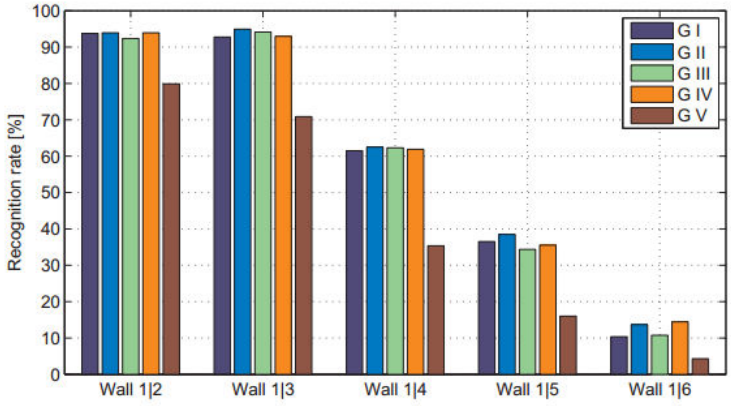


Figure 2.9. Evaluation of recognition for previous selected patterns. [17]

Oriented Brief (ORB)

Rotation invariance was the major issue for the BRIEF descriptor. Not much time has passed since the creation of the ORB [18] descriptor. Authors combined the FAST key-point detector with the method of BRIEF descriptor to create fast binary local descriptor. They upgraded FAST detector with key-point orientation and according to key-point orientation steered the BRIEF descriptor itself.

Fast Retina Key-point (FREAK)

FREAK descriptor [19] was inspired by the human visual system. For key-point detection FREAK descriptor use the same approach as were presented in BRISK [20] descriptor.

Descriptor is consisted of binary strings created by comparing different intensities of the image over the selected pattern. Pattern which is used for descriptor filling is similar to retinal ganglion cells. We show the pattern image, where each circle represents receptive field. The larger circle, the more is image smoothed by using Gaussian convolution.

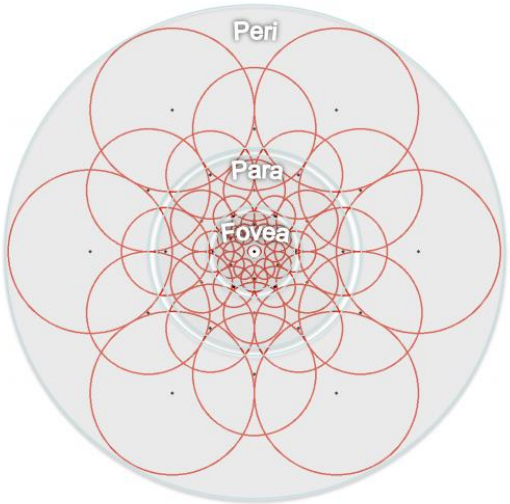


Figure 2.10. Pattern used in FREAK description. [19]

Histogram of Intensity Patches (HIPS)

From our previous work [21] on bachelor thesis we have chosen HIPS descriptor [22] to complete the list. This descriptor was introduced as a fast descriptor capable of real time matching even on low cost CPU units. We have successfully implemented HIPS descriptor to

be able to match in real time even on low price mobile device at the cost of full rotation invariance, making HIPS descriptor powerful to use for weaker devices.

For key-point localization HIPS descriptor use the well-known FAST algorithm. As the FAST detector does not solve the scale and rotation invariance we need to add it to our solution. To be able to match object from different angles and rotations we create a training set of images also called as viewpoints from which we want our object to be detected. Next, to increase robustness of our algorithm all viewpoints are slightly transformed in the manner of affine and perspective transformations. Now we can start a training phase for HIPS descriptor.

From each of the viewpoints images we detect key-points using FAST key-point detector. We select top 50 to 100 features which occurs in the viewpoint most of the time. As we know for each image how it was transformed according to its reference viewpoint image, we can transfer each key-point coordination's back to their original place. It can be easily done by making a list of features with counter for each feature. Those features appear to be strongest for selected viewpoint.

After we have selected top key-points from each viewpoint we create a sample grid of 8 x 8 pixels around them to form the descriptor vector. Each of the value in the grid is normalized to achieve light invariance. Histogram with 5 bins is then created from the intensity values of the pixels at the same position across the same key-points in viewpoint images. According to the position of pixel and value in the histogram we fill the HIPS descriptor in a way that 1 is filled to the descriptor vector when the selected intensity was under selected threshold. Otherwise 0 was filled. Next figure shows the process of filling the descriptor.

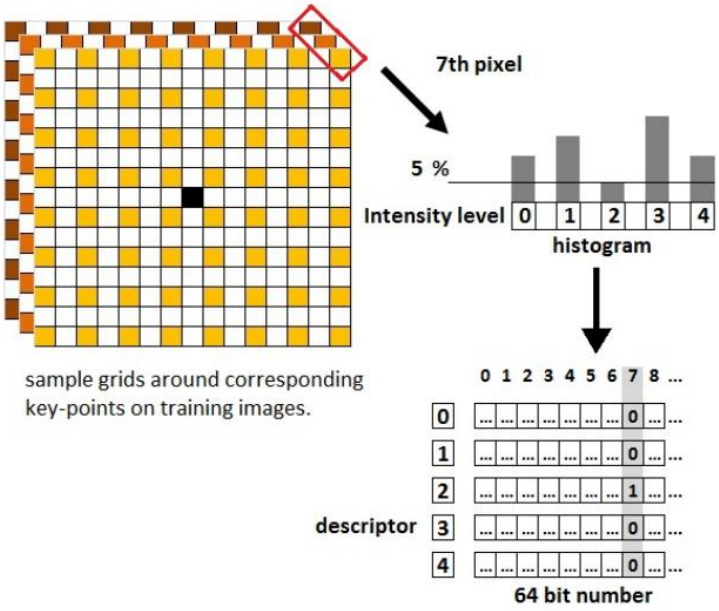


Figure 2.11. Process of forming HIPS descriptor. [21]

Same procedure applies on the image from camera at the recognition time but instead of binary number 1, 0 is filled when the intensity was under selected threshold.

By this we can easily compute the dissimilarity score of two created descriptors using binary AND operation. Exactly the same descriptor vectors after binary AND will lead to the result of vector filled with all zeros. The more the two descriptors are different, the more ones will be in the result vector. To simply match the pair of descriptors we calculate dissimilarity score. It is done by counting the one numbers in the binary vector after AND operation.

2.1.3. Descriptor matching

Considering the fact that each descriptor vector created went through the complex process starting from detecting key-points, achieving invariance to different transformations, normalizing the values end to filling the descriptor, there is high chance that even the same key-points detected on almost the same image can be described in slightly different manner. Hence to find corresponding pairs of descriptors we are not looking for the exact the same pair. We perform actions more likely to be called as finding nearest neighbor or similarity search.

The simplest way to get pair of corresponding descriptors is to make a brute force scan through all of stored descriptors and declare pair of descriptors as a good pair under selected

threshold. This solution is good for smaller number of trained objects, however if we want to store more objects and still want to be able to match them in real time, we need to consider more efficient solution like KD trees or hash functions. [2]

Distance measurement

In previous chapters, we mentioned two types of local descriptors. The numerical and the binary ones. Each of the selected type of local descriptor has their own method for matching the good pair of descriptors. We present basic methods for matching pairs of descriptors for both types.

Euclidean distance

Euclidean distance [2] is used to measure distance of two numerical values. Smaller the result of the Euclidean distance is, the more similar pair of descriptor vectors are. It is given by the Pythagorean formula. In next formula, d_1^i represents *i-th* dimension value of the first pair of descriptor.

$$distance = \sqrt{\sum_{i=1}^{n} (d_1^i - d_2^i)^2}$$

Hamming distance

The purpose of binary local descriptors was to make computing the distance between pair of descriptors easier and faster for current CPU's. The descriptor vector of binary numbers is also known as binary string. The result of Hamming distance [2] is the number of positions at which the binary strings were different. To find out the similarity of pair of descriptors, we perform binary operation XOR between the binary strings. By definition of XOR, the result is 1 when the values at same position of two vectors were different. To calculate the distance, we simply count the occurrence of ones in the result vector.

A	B	XOR
1	*1*	*0*
1	*0*	*1*
0	*1*	*1*
0	*0*	*0*

Table 2.1. Exclusive OR operator

Homography

The next step after finding the pairs of descriptor is to compute the homography [2] matrix. In other words we are trying to get geometrical representation or transformation of the object found in the result image according to the stored object. This process can also serve as a verification phase if the selected object occurs in the result image as we know that some geometrical shapes cannot exist in real world. Local descriptor method can return also false positive pairs which will result in unpredictable shapes after we retrieve homography matrix.

The process of finding homography helps us also to select true positive matches also known as inliers and separate them from true negative matches (outliers) which were selected as good matches from previous distance measurement.

RANSAC

RANSAC [2] is well known algorithm for finding homography. First, it takes four random pairs of descriptors to compute the homography matrix. In next step from it compares all pairs of descriptors left with previous created homoraphy matrix. When more than 50% of descriptor pairs fit into the homography matrix, it claim selected homography matrix as a result.

2.2. Segmentation

Segmentation is quite an old and quite complex task of the computer vision [1]. The idea of segmentation is to be able to distinguish the desired object or group of pixels from the rest of the image. In easier problems we are trying to merge group of pixels with common attributes like color together. More complex issues are focused on object segmentation from the background. There is no universal method for image segmentation and different task would most likely require different approach of segmentation. Nowadays there are several methods for image segmentation. We describe most common segmentation methods and our simplified approach for the depth image.

2.2.1. Active contours

Algorithm based on the active contours method [1] focus on finding the curves to determine boundaries for segmentation. Active contours are also called as snakes. It is an iterative process. We can imagine it as a contour line in the image which is trying to minimize

the energy at current contour. Algorithms which belongs to this category are: *snakes, dynamic snakes, scissors or level sets*.

2.2.2. Split and Merge

Next approach for image segmentation is to split large image into smaller areas. [1] This approach generally splits the area into 4 smaller areas of the same size recursively. In next step, those 4 area textures are compared if they are likely to be the same. If so, they are merged together. As we were splitting the image in a certain pattern we may accidentally split uniform region into 2 areas. Therefore the next step called merge proceeds to compare neighbor areas and merge them together if they have the same texture.

2.2.3. Mean shift and mode finding techniques

Mean shift method [1] can be translated also as "per pixel" method. Each pixel is represented in a feature vector as a sample of probability function. The possible data to create such a vector could be color and position. We can consider this as a classification problem. Algorithms such as *k-means clustering, mean shift or Gaussian mixture models* belongs to this category.

2.2.4. Growing regions

Another segmentation method which can be considered as region or pixel based is called growing regions. It is an iterative process. At the very beginning, we need to pick up the seed point which refer to a start point for segmentation. In later steps we look at seed point neighbor pixels and determine if they belong to the region or not. The process is done for all pixels which were added to the region. The cons of this method is that we need to choose suitable pixel as a seed point. This can be tricky part of the work for the color image. Also selecting the threshold for pixel comparison is essential in order to get good results.

2.2.5. Segmentation in depth image

To be able to distinguish desired object from the background in the depth image, we can use much easier methods which are not time and memory consuming. Depth sensors can give us data which represent the distance of the selected pixel from the sensor. Segmentation task can be easily done just by comparing the distances over pixels. In 2D images we could only determine boundaries by color information. In 3D depth image objects are represented

by continuous surface of pixels with similar distance. The object boundaries are then determined by large change in the distance. Growing region method for segmentation can be used there as we can easily select seed point pixel just by selecting the closest pixel in the depth image.

2.3. Kinect sensor

Kinect is one of the devices [3] which is capable of acquiring images with depth information. Device was created by Microsoft and it consists of the standard RGB color camera, depth sensor and an array of microphones. Kinect device also have tilt motor which is capable of rotating the head of the device.

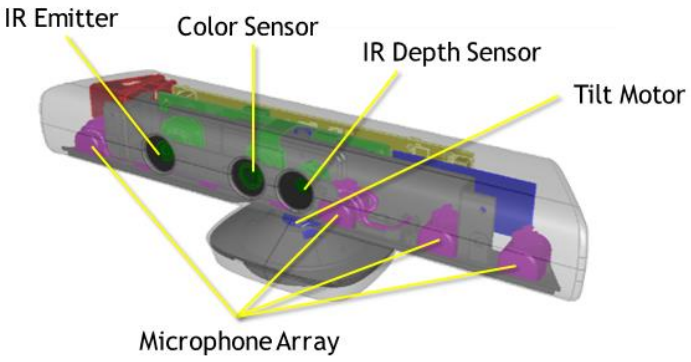


Figure 2.12. Hardware components of Kinect device. [3]

From the Microsoft website, the specifications of the device are as follow:

Kinect	**Array Specifications**
Viewing angle	43° vertical by 57° horizontal field of view
Vertical tilt range	±27°
Frame rate (depth and color stream)	30 frames per second (FPS)
Audio format	16-kHz, 24-bit mono pulse code modulation (PCM)
Audio input characteristics	A four-microphone array with 24-bit analog-to-digital converter (ADC) and Kinect-resident signal processing including acoustic echo cancellation and noise suppression
Accelerometer characteristics	A 2G/4G/8G accelerometer configured for the 2G range, with a 1° accuracy upper limit.

Table 2.2. Kinect sensor device specifications. [3]

The purpose of the device was originally to be able to play games on the XBOX 360 device, but it has many advantages in the research field of computer vision for its price and availability on the market. Device with the Microsoft software kit is able to: [3]

- Capture RGB data from camera
- Capture depth image from depth sensor
- Perform skeleton tracking
- Recognize speech gestures

Along with the Kinect sensor Microsoft created their own SDK (Software Developer Kit) to help developers make use of the Kinect device for research purposes. The interaction between the Kinect sensor and application is allowed by NUI API (Native User Interface) through which can we obtain desired color, depth or audio stream.

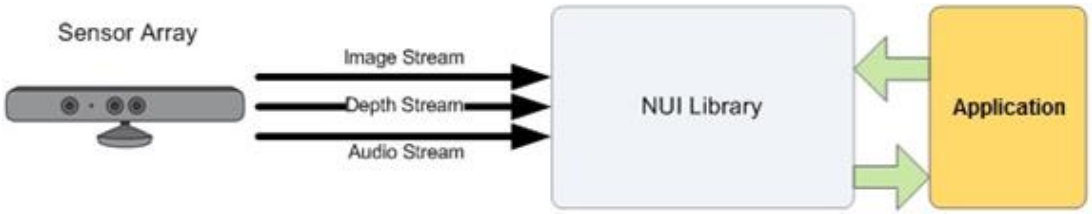


Figure 2.13. Interaction between software and device. [3]

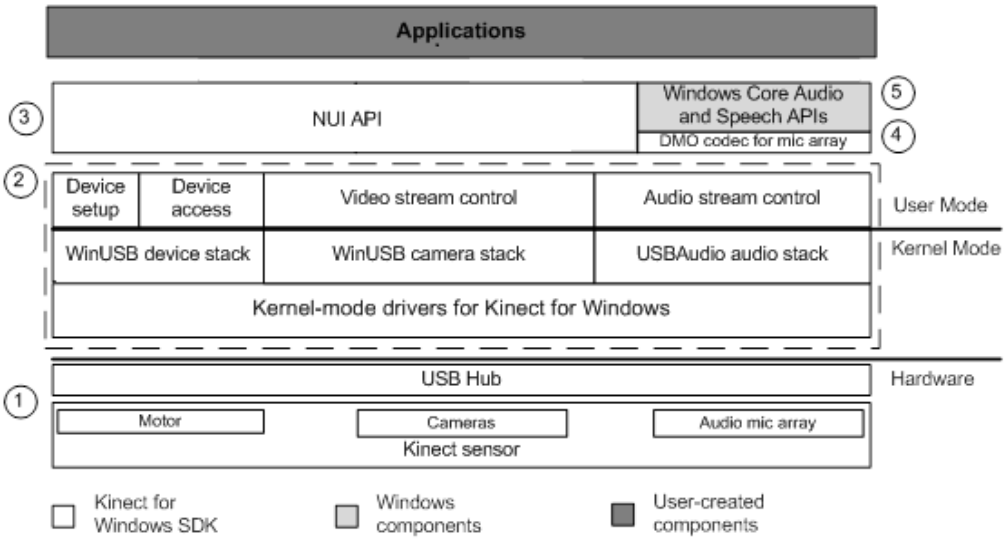


Figure 2.14. Kinect SDK architecture. [3]

2.4. Kinect for Windows v2

During the July 2014 new Kinect sensor v2 were introduced by Microsoft. Providing fidelity of the depth image three times better than in Kinect v1 sensor, it could take 3D object recognition to the new level where the depth description and matching could be done without merging it with color descriptors.

Similar to the Kinect v1 device, new sensor have both RGB and Depth sensor. The new depth sensor is based on the time of flight method, providing more accurate results for depth mapping along with the better resolution. In next figure, we show differences between v1 and v2 sensor.

Feature	Kinect for Windows 1	Kinect for Windows 2
Color Camera	640 x 480 @30 fps	1920 x 1080 @30 fps
Depth Camera	320 x 240	512 x 424
Max Depth Distance	~4.5 M	~4.5 M
Min Depth Distance	40 cm in near mode	50 cm
Horizontal Field of View	57 degrees	70 degrees
Vertical Field of View	43 degrees	60 degrees
Tilt Motor	yes	no
Skeleton Joints Defined	20 joints	26 joints
Full Skeletons Tracked	2	6
USB Standard	2.0	3.0
Supported OS	Win 7, Win 8	Win 8
Price	$299	TBD

Figure 2.15. Comparison of Kinect v1 and v2.

The new Kinect device comes with the new Kinect SDK v2.0. Few changes has been done from the previous version. First, the new Kinect device works only with the new Windows 8 operating system and need USB 3.0 port. The SDK is supported for different languages like C++, C#, JavaScript and more. New SDK contains also additional features like integration with the Unity using a plugin, which is used for the game development.

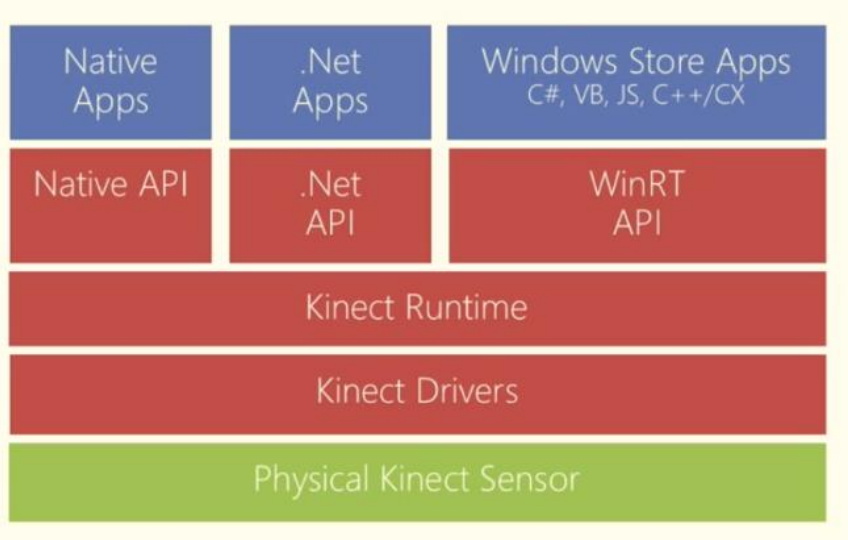


Figure 2.16. High level architecture of new Kinect v2 SDK. [3]

3. Related work

We made a research about given task and present our conclusions to related work. Most interesting points from other author's work will be presented, with the aim of providing useful information for our research.

3.1. Computer visual object detection

The previous work on this topic field have been done by diploma thesis [23] we are continue working at. In previous work, there were three modules implemented. First module was called *KinectX* which provides high level interface for the Kinect sensor to manipulate with. It have been done in the purpose of better code management for the main implementation of object detection.

For the second part of the thesis, two ways of detecting objects using the Kinect sensor were implemented. The first algorithm is based on local histogram of intensities comparison. The second one is the algorithm based on the SIFT descriptor with added depth information. The pipeline for the recognition algorithm is as follow:

- Acquire the color and depth image from Kinect sensor
- Align the depth image according to the color image
- Remove artifacts caused by depth sensor in depth image
- Compute the segmentation mask of the object
- Detect features on the segmented part of the image
- Feature extraction from depth and color image
- Descriptors matching

Kinect device which were used for acquiring the depth information have 2 sensors which are not at the same position. Also, the depth sensor support max resolution of 640 to 480 pixels, where camera supports maximum of 1024 to 768 pixels. The consequence of this set up is that artifacts can be detected as the depth image cannot and in almost every case will not be fully aligned to the color image. In the thesis, artifacts were removed by using filters such as median.

For the segmentation method depth image was used for its efficiency. We can simply select the continuous surface of closest object to create segmentation mask. This part of the algorithm will save us a lot of computation time as we do not need to detect and extract features all over the image.

As for the last step of detecting and extracting features (key-points) we focus on the SIFT implementation. SIFT detector and descriptor was used over a color image. In addition, the result 128 dimensional descriptor vector was expanded with another 2 dimensions. For each detected key-point, we look at the depth image and acquire standard deviation and maximal difference from the depth data at selected surface around the key-point. Those are two additional attributes which were added to the result SIFT descriptor.

3.2. NARF: 3D Range Image Features for Object Recognition

Authors of the paper [24] [25] are interested in feature extraction and description from the 3D image data. They presented interest point extraction method called NARF (Normal Aligned Radial Feature) together with descriptor. For the extraction of interest points or so called key-points, they want to achieve two main rules:

- Detected key-points need to be located in the stable surface region. The main reason is to achieve robustness of the algorithm, as in next process they want to extract normal vector at the selected point from surface around the key-point. Unstable surface may result in not reliable results and errors in matching.
- They want to use object borders which represent shapes of the object which can be seen at current depth frame. Considering the algorithm is used on the hardware which can capture partial view of the 3D scene (laser based scanners, stereo cameras or Kinect device) the shape of the object will be different from other views. Those shapes are rather unique for the selected objects and can result in increased robustness if they are used for the description.

3.2.1. Feature detection

The process of detecting key-points is most important part for the descriptor to be good at matching. For the NARF descriptor, key-points need to be able to recover information

regarding the borders and the surface. Detector need to detect those points which can be also detected from different perspective view of an image. The algorithm for finding the key-points is as follows:

1. Find borders in the range image. Border is found as the non-continuous traversal from the foreground to the background of the image where they look for distance increases between two neighbor points in the image.
2. For every image point, look at the local neighborhood of the image. Compute the change in the surface and dominant orientation for the change.
3. Based on the dominant orientations of the surrounding image points calculate a value which will represent the difference between the orientations in the area and the change in the surface area (how much stable it is).
4. Smooth the interesting values.
5. Perform non-maximal suppression to be able to detect the final key-points.

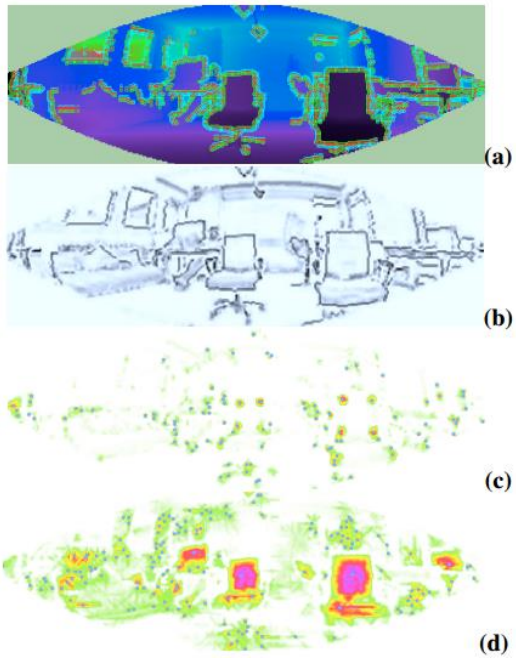


Figure 3.1. Interest point extraction procedure. [24]

3.2.2. Feature extraction

Rotation invariance for the NARF descriptor is achieved by calculating the orientation around the normal which is similar to selecting dominant orientation for 2D descriptors like SIFT, except the 3D space. With the 3D information we are able to determine the

transformation for all the 6DOF (degrees of freedom). To create NARF descriptor around the selected key-point we need to calculate normal range value patch. Star pattern is then used for the selected patch to compute the descriptor. Each line of the star pattern will get its part in the filling of the descriptor in a way it represent how much of the pixels under the line change. Last step is to find dominant orientation of the descriptor and rotate it to default position to achieve rotation invariance.

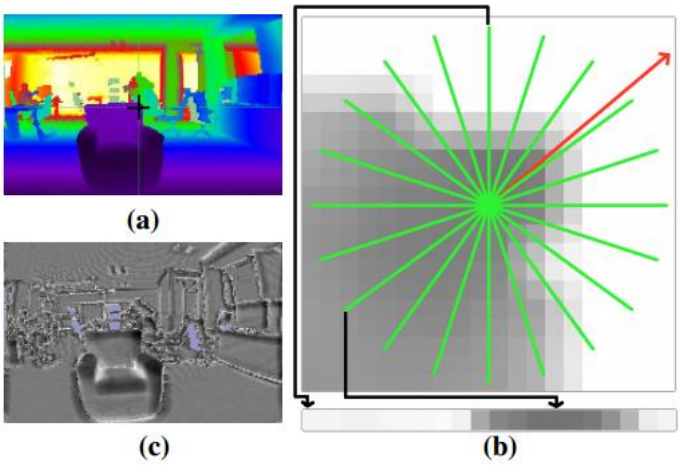


Figure 3.2. Pattern used to fill NARF descriptor. [25]

3.3. A combined texture-shape descriptor for enhanced 3D feature matching

Authors of the paper proposed a novel descriptor for 3D feature matching containing both shape and texture information. They proposed descriptor called CSHOT [26] (Color SHOT) which should improve accuracy of the recognition in environment where clutter and occlusion is present.

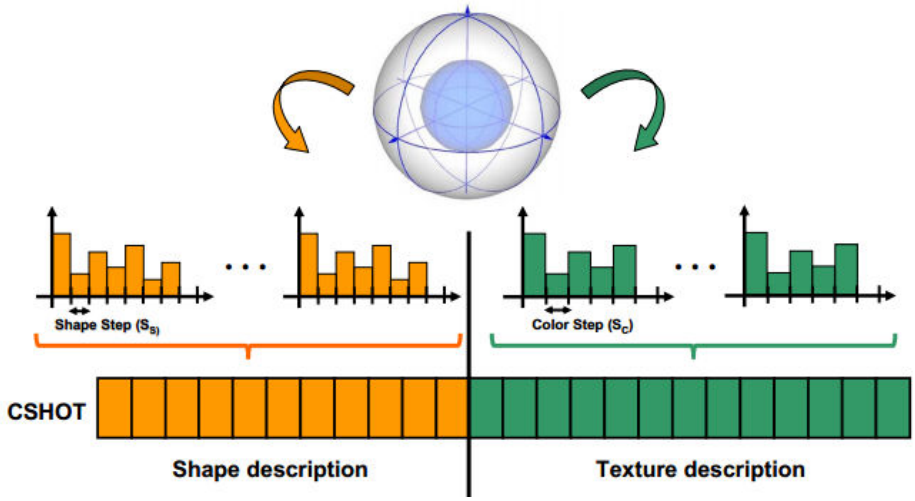


Figure 3.3. Structure of CSHOT descriptor [26]

CSHOT descriptor extended from the SHOT descriptor [27]. Descriptor is based on eigenvalue decomposition of a scatter matrix around selected point. In the paper also known as definition of repeatable local reference frame. To encode spatially information (signature structure) about the point, isotropic spherical grid is defined based on the local reference frame. The result descriptor is then formed in a way that histogram of normal vectors is defined for each sector of the grid and stored.

For the design generalization let's say $SH_{G,f}(P)$ refers to genetic signature of histograms computed over spherical support around feature point P. The signature histogram relies on the G = *vector-valued point-wise property of a vertex* and f = metric used for comparison of two point-wise properties. Next, to compute the signature histogram we apply the f metric over all pairs (G_P, G_Q) where Q represent generic vertex around feature point P.

In order to build up the descriptor at selected feature point, we compute m signatures of histograms from different pairs and merge them together.

$$D(P) = \bigcup_{i=1}^{m} SH^{i}_{(G,f)}(P)$$

As for the texture based part of the descriptor, the authors of paper hand in hand with comparison of RGB intensities associated to each vertex have also chosen an alternative metric based on the L_p norm between two triplets. *L1* norm was implemented as the sum of absolute differences between triplets.

$$l(R_P, R_Q) = \sum_{i=1}^{3} |R_P(i) - R_Q(i)|$$

In addition, CIELab color space was used for the testing purposes as it is well known of being more perceptually uniform than RGB color space. For CIELab color space, two metrics were deployed also known as CIE94 and CIE2000.

3.4. Surface feature detection and description with applications to mesh matching

In presented paper, authors propose 3D feature detector (MeshDOG) along with 3D feature descriptor (MeshHOG) for triangulated meshes [28]. Next to be described descriptor is invariant to changes in rotation, translation and scale. Photometric information available with 2D images with geometric information from 3D sensors are handled hand to hand in a consistent and simultaneous manner. Photometric information from 3D models can be viewed as scalar functions and represent generalization of planar to non-planar domains.

As both photometric information and surface geometry are taken into consideration, discrete convolution and discrete gradient are defined on surfaces (meshes). Based on these functions, MeshDOG and MeshHOG detector and descriptor are presented.

3.4.1. Feature Detection (MeshDOG)

MeshDOG detector is a generalization of DoG operator. Detector seeks for the extrema of a scale-space representation of scalar functions defined over a discrete manifold. The MeshDOG detector performs in 3 main steps:

1. We find the extrema using difference of gaussian method across scales.
2. Apply threshold at detected extrema.
3. We eliminate unstable extrema and keep those locations of meshes which appears to be corners.

Figure 3.4. Feature detection based on the previous steps. [28]

To eliminate more unstable responses, at the third phase of the detection Hessian operator is used.

3.4.2. Feature Descriptor (MeshHOG)

MeshHOG is based on the histogram of oriented gradient descriptor (HOG). To compute descriptor at specific vertex we use support region with defined neighborhood ring size. We compute gradient information from each vertex in the neighborhood and translate it according to dominant orientation to achieve rotation invariance. Next we compute histogram of gradient where each gradient vector is 3 dimensional.

For increased robustness to scaling and different spatial samplings the number of rings for the support region is based on a global measure chosen adaptively. Instead of creating histogram with full 3D information, gradient vectors are projected to 3 orthonormal planes which describes the local coordinate system and provide more compact representation. Histogram with 2 levels is then computed for each of the plane after it is divided into 4 polar slices. At the end we compute orientation histogram with 8 bins for each slice.

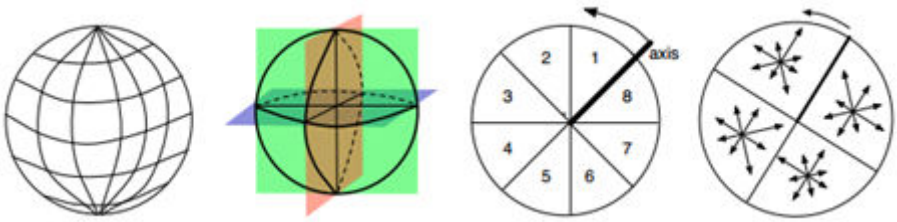


Figure 3.5. Process of creating MeshHOG descriptor. [28]

4. Proposed method

In the following part we are going to further describe the base idea of the depth descriptor (DD). We will discuss the information depth map can provide us along with its application for depth description and object recognition.

4.1. Specification

In previous solution [23], the original SIFT descriptor were enhanced with additional depth statistical information. The depth information was taken from the Kinect v1 device and there was no point of considering standalone depth descriptor because of the sensor fidelity. As the new Kinect v2 device was introduced with better precision we have decided to create depth descriptor capable of object recognition. Although it's precision will unlikely match the SIFT descriptor, it can be well used for object pre-selection. Considering less number of objects will pass to the second round of color matching with 128 dimensional SIFT descriptor we can decrease recognition time over larger databases.

In the next figure we show the proposed method with depth descriptor.

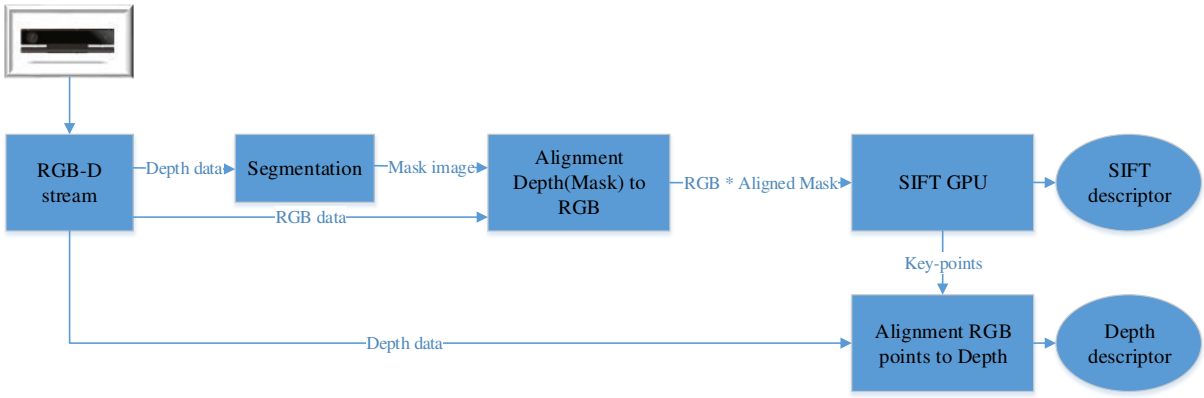


Figure 4.1. Pipeline of the proposed algorithm.

4.2. Components design

Application will be consisted of the following components:

- **Kinect2X library** – Like in previous version, library will provide us with the high-level management of the Kinect v2 device. Starting up with the acquiring basic color

and depth stream, library will also provide basic algorithms working with OpenCV library data types like cv::Mat resulting in better usability. There will be also implemented methods for depth and color image alignment of the streams needed for object segmentation and key-point localization.

- **Descriptor Library** – Library will be used for extracting the depth descriptor. The main idea behind the library is in re-usability of the code isolating the main algorithm from the GUI part so the code could be simply used for different applications.
- **GUI application** – Main application providing us with the model-view-controller design used for depth and color stream visualization from the Kinect2X library as well as communication with the user.

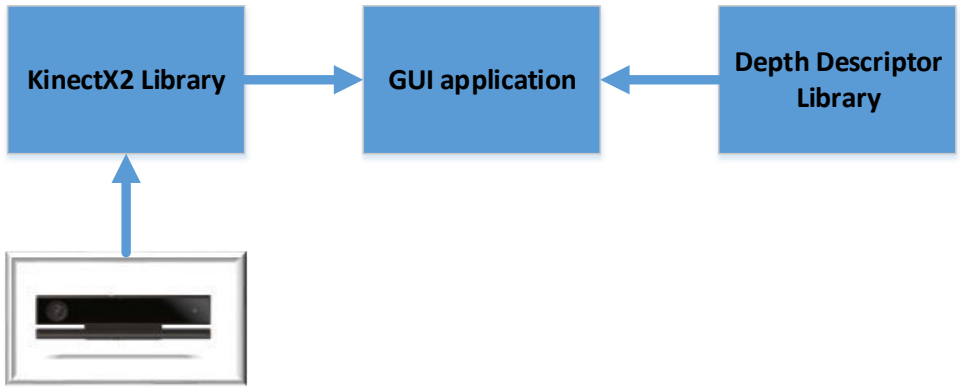


Figure 4.2. Components design.

4.3. Cascade recognition

From the previous work, SIFT descriptor enhanced with depth information was implemented. Because of the increase in dimensions of the descriptor from 128 to 130, the time needed to process frame will rise. Also for the proper matching we will need to re-normalize all the values in the descriptor vector. As we use knn classifier for the matching phase, it could lead to the curse of dimensionality issue without re-normalization of the data. Resulting descriptor will be able to use both depth and color stream for the recognition, but the depth information will have only small overall influence over the recognition.

Thanks to the new Kinect v2 device with higher resolution and time of flight technology we are able to make recognition based only on the depth data. Taking idea from the cascade matching which is now used for example in face detection can lead to improved performance and accurate results.

We will divide the recognition part into the three steps. In the first part, we will consider object recognition based on the global information of the object. As the global information for the object will be stored in only one vector of few values, we expect to remove only objects with large surface difference from the possible matches. The result of this recognition, or better called pre-selection will be the sorted indices of the objects from the best to the worst match. Ordered objects can be later used for the depth descriptor we are going to create and describe later. As we have ordered the objects and will start matching the descriptors from the best match to the worst, we can easily filter out the rest of the objects as we come to the first object which did not pass the threshold value. For the last descriptor matching, we will use original SIFT descriptor which is robust and will give us the best match from the remaining objects.

In the next figures, we show the pipeline used for matching in original solution compared to the new solution with depth descriptor.

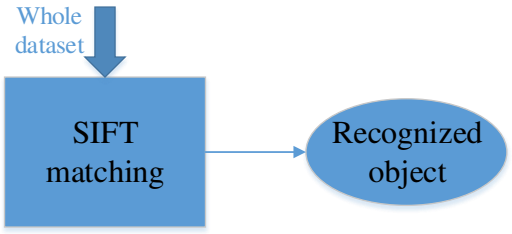


Figure 4.3. Matching process using the default settings.

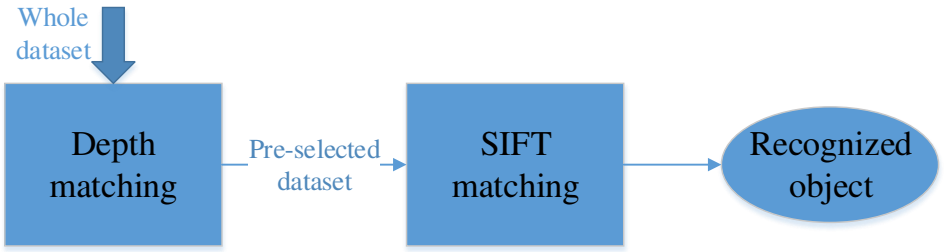


Figure 4.4. Matching process using our proposed method.

4.4. Image stream pre-processing

Resolution of the depth and color data obtained from the Kinect v2 device is different for both streams as well as field of view for both of the sensors, therefore as we detect key-points in color image, their coordinates will not point to the same position in the depth image. To be able to know the coordinates of key-points in the depth image, we need to make an

alignment of the color image to the depth image. This process can be done using the Kinect SDK and will be part of the Kinect2X library.

In addition we are going to make alignment of the whole depth image according to the color image because of the object segmentation. Segmentation of the object within the depth image will provide us with the mask with the same resolution as the depth data and need to be aligned to the color map. Using simple multiplication of the mask image with color image we are able to get the object texture.

4.5. Object segmentation

Segmentation methods used for color images can suite us well for the depth image. Considering the depth image as one channel image (for example grayscale) we can make use of the image just like if we use it for the color processing. In our approach we are going to use the method of growing regions with the seed point set to the coordinates of the lowest depth value.

Thanks to the object segmentation we will be able to remove large number of key-points detected on the scene unrelated to our object (image background). It will greatly decrease the recognition time needed for object matching.

The growing regions method is suitable for this situation, as it select only points which are within the certain threshold from the seed point. The segmentation can be also enhanced in a way that the threshold value will be considered dynamically for each pixel neighborhood. This enhancement will be able to select also objects where the surface depth change more rapidly but still remove the background.

Figure 4.5. Object segmentation

4.6. Depth descriptor design

To create depth descriptor we have chosen to create four dimensional vector filled with statistical information of the object. All the information are based on the normal vector created from the local surface around the key-point. During the descriptor creation we aim to maintain the invariance of the descriptor at least to the invariance level of the color descriptor as the depth descriptor will be used first in line for the recognition.

4.6.1. Key-point detection

Best key-points for depth matching are those with different shapes of the objects with stable but descriptive information of the local surface. It means that the best features would be where the depth changes within the object, excluding the borders of the objects where the depth could change rapidly - for example when the depth from the scene behind the object will be captured.

Because of the four dimensions in the descriptor vector filled with statistical information we will not be able to recognize object with certain precision. Therefore we are going to make a tradeoff of finding the best key-points for depth description within the object and save processing time. Instead we use key-points already detected by detector for color matching which will still provide us with reliable information regarding the depth. As the depth descriptor should not be used for object matching alone we will use it for pre-selection before color matching, speeding up the overall recognition process. The use of the key-points detected from the color stream for the purpose of depth recognition will later be evaluated.

4.6.2. Descriptor pattern

All the information we are going to use in the description is related to the normal vector. Computing the normal vector and determining the surface can be easily done using three points forming the triangle. Overall four triangle patterns are used forming the star pattern over the key-points. The radius size within each triangle is formed changes according to the current depth of the key-point and is set to certain value.

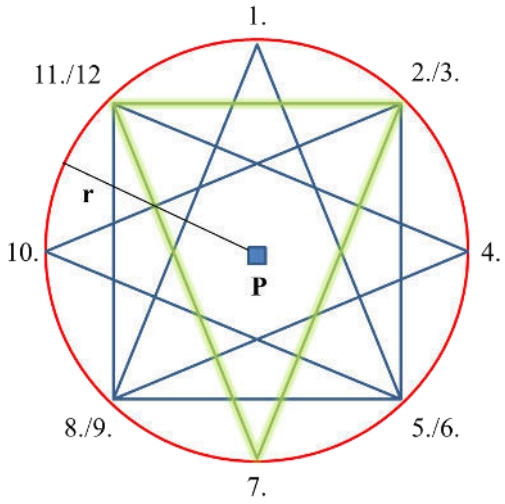


Figure 4.6. Pattern created out of triangles within the radius *r* around the key-point *P*.

Because we know the real world distance of the key-points we are able to compute the pixel size of the pattern within the depth image. This will ensure that the same surface size is chosen for the same key-point captured from different distance.

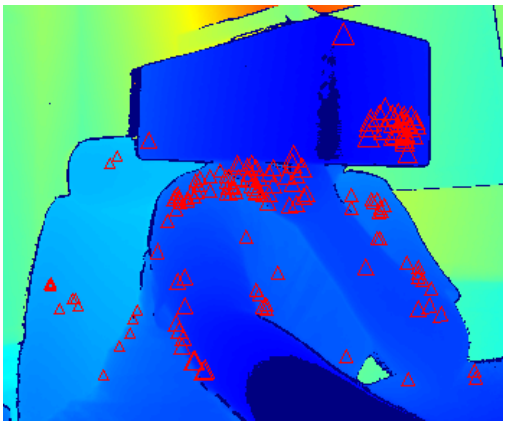


Figure 4.7. Visualization of change in size according to the depth.

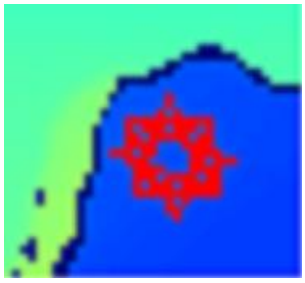


Figure 4.8. Descriptor pattern visualization

4.6.3. Depth description (Descriptor vector)

Four features, which are derived by statistical evaluation, have been taken for the definition of the depth feature vector. Three of them are based on the local surface description and the fourth feature take into account also the global information of the object. In addition, we filter out those triangles, which cannot provide the depth value in at least one point. At the end all of the values are normalized.

Average angle

First feature in the depth feature vector is an average angle of the normal vectors given by all triangles in the key-point. Therefore we need to compute four normal vectors - one for each of the four triangles given by the pattern. Then we calculate the mean value of this vectors. In the next step, we compute the difference angle between each of the normal vectors and the mean normal vector. Now we can fill the first value of our descriptor by the feature value F_avgAngle.

$$F\_avgAngle = \frac{1}{N}\sum_{i=1}^{N} \arccos\left(\frac{u_{i1}.v_{m1} + u_{i2}.v_{m2} + u_{i3}.v_{m3}}{\sqrt{u_{i1}^2 + u_{i2}^2 + u_{i3}^2}.\sqrt{v_{m1}^2 + v_{m2}^2 + v_{m3}^2}} \right)$$

Where N is the number normal vectors/triangles.

$\vec{v}_m$ is the average normal vector.

$\vec{u}_i$ is the normal vector given by triangle.

Standard deviation

Second value in the depth descriptor is the standard deviation of all depth values of the surfaces in the positions given by the triangles.

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

Where N is the number of all points in the descriptor pattern.

x_i is the distance of the point from average surface.

μ is the average distance of all points in pattern from average surface.

Difference of maximal and minimal depth

Difference between the maximum and minimum value define the next feature value in the feature vector. Maximum and minimum can be evaluated during the process of computing standard deviation in the same cycle. We store the maximal and minimal value and return their difference.

Global angle

For the fourth value we have chosen the angle of the average normal vector at given key-point and the normal vector of the whole object. The object normal vector is based on the average vector value out of all normal vectors from all key-points. Comparing the results of this value through all descriptors can provide us with the information relative to the surface alignment of the object.

$$F\_globalAngle = \arccos\left(\frac{g_1.v_1 + g_2.v_2 + g_3.v_3}{\sqrt{g_1^2 + g_2^2 + g_3^2}.\sqrt{v_1^2 + v_2^2 + v_3^2}}\right)$$

Where $\vec{g}$ is the average normal vector out of all normal vectors through all key-points.

$\vec{v}$ is the average the normal vector at given key-point.

$$g = \frac{1}{N}\sum_{i=1}^{N} v_i$$

In next figure we show the visualization of the global angle. Bottom bar show us the possible values starting from the lowest left to the right. There is also average value of all angles shown as black dot within the bar.

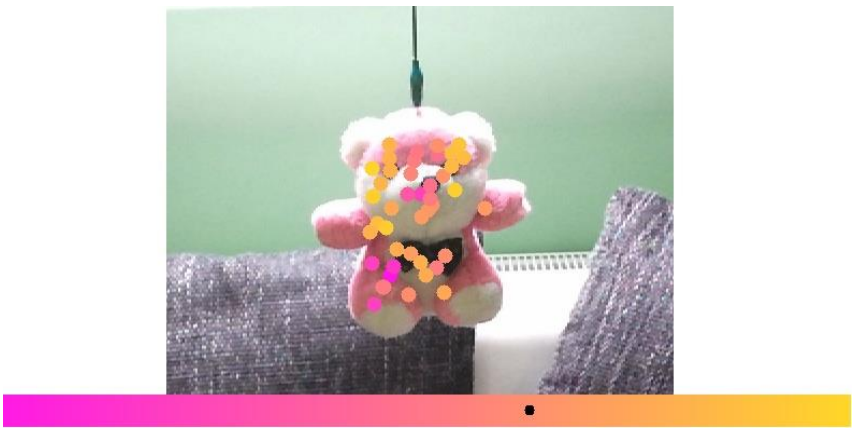


Figure 4.9. Visualization of angle at the given key-point with global normal vector.

4.6.4. Descriptor invariance

The original SIFT descriptor is invariant to different scale, rotation and small perspective transformation of the object. As we are going to make a depth descriptor which object need to pass we need to achieve at least the same invariance to the depth descriptor

Scale invariance

Different scale of the object can be translated into the distance for the 3D scene. Making our descriptor invariant to the object distance is the matter of determining the surface used for creating the descriptor. As we know the distance of the selected key-point from the depth map, we can easily determine the surface around the key-point according to the real world distance, instead of the normal length in pixels which is used in other descriptors.

Rotation & perspective invariance

To achieve invariance to different rotation of the object we have chosen to fill the descriptor with the statistical information which are independent to the rotation and angle of the object. Therefore the values should not change as we capture the object from different view.

4.6.5. Descriptor matching

We will use the created depth descriptor in the same way as the SIFT descriptor is used during the matching phase. All key-points found on the object using SIFT detector will be taken to create new depth descriptor and then used for descriptor matching. Knn classifier with k=2 will be used to obtain best pairs of descriptor using Euclidean distance metric. In addition, nearest pair will be chosen only if their distance is within certain ratio threshold to the second nearest pair. This method was proposed by Lowe with a goal to remove those matches which could be labeled as false positive.

5. Implementation

In this part we are going to take closer look at implemented algorithms for the RGB-D images as well as used software or libraries. We will describe the individual parts of our depth descriptor through the key-points we use for descriptor extraction, the extraction itself up to the matching phase.

5.1. Technology used

Application of object recognition is implemented in C++ using the Microsoft Visual Studio 2013 under Windows 8.1 platform. As for the first evaluation, we have decided to continue work on the previous created GUI application. Following software and libraries were used in the implemented system:

- Object Recognizer [23]
- Qt framework[1]
- Kinect SDK v2.0[2]
- OpenCV library v2.4.8[3]
- SIFT GPU [29]

We have decided to remove the KinectX library from previous solution as it was developed for first version of Kinect. Next, we removed the PCL library. It was used for statistical evaluation over depth data which we replaced with our own implementation.

5.1.1. Object Recognizer

Previous solution of Object Recognizer [23] was modified to fulfill the new requirements. Most classes related to the previous version of Kinect were removed along with the classes used for the recognition. These classes were re-implemented and extended in the standalone Descriptor library which will be explained later.

[1] Qt Framework [online]. 2014. [Accessed December 2014]. Available from: https://www.qt.io/
[2] Kinect SDK [online]. 2014. [Accessed August 2014]. Available from: http://www.microsoft.com/en-us/download/details.aspx?id=44561
[3] OpenCV Library [online]. 2014. [Accessed December 2014]. Available from: http://opencv.org/

Object Recognizer application is used to create graphical interface for the recognition using the Qt framework. We have chosen the Qt framework because of the signal/slot communication, simple threading model and we also use its build methods for loading configuration files where we can adjust the recognition settings.

5.1.2. Kinect SDK

Kinect for Windows Software Development Kit allow the user to create applications which use the voice, depth, color or other streams from the new Kinect v2 device. The kit contains also high level methods which we can use for the gesture, face or voice recognition, image alignment and other.

The requirements to use the device with the SDK however are strictly bond to the new Windows 8, 64bit operating system with USB 3.0 present and DirectX 11 capable graphic adapter.

Within the SDK we can find Kinect Studio application which can be used to simulate the connected device. We can easily store the specific streams from the device and later use it in any application without need to rewrite the code.

5.1.3. OpenCV

OpenCV is the well-known library for computer vision and image processing. The library have support for large scale of operating systems like Windows, Android OS, Linux, iOS and Mac OS. The supported languages are C/C++/Python/Java. The OpenCV library algorithms and functions are divided into several modules:

- **Core** – Defines basic structures and functions
- **Imgproc** – Image processing. It contains algorithms for image filtering, transformations and others.
- **Video -** Video analysis, movement estimation or background removal.
- **Calib3d** - Basic geometric algorithms, 3D image reconstruction, camera calibration.
- **Features2d** – Local descriptors module, key-point detection, descriptor matching.
- **Objdetect –** Detection of the predefined objects
- **Highgui** – Basic interfaces, image & video capturing
- **Gpu** - Acceleration of algorithms on the GPU from previous modules.

5.2. Architecture of the solution

In our solution we intend to separate the three basic parts of the object recognition in their own libraries/applications:

- GUI application for interaction with the user (Object Recognizer)
- Library to provide us with the basic connection to the Kinect v2 device, basic image alignment and data transformation which can be used by OpenCV Library (Kinect2X)
- Library for the object recognition (Descriptor Library)

The module parts of the solution can be found in the next figure. We will describe the components of each part in the next sections.

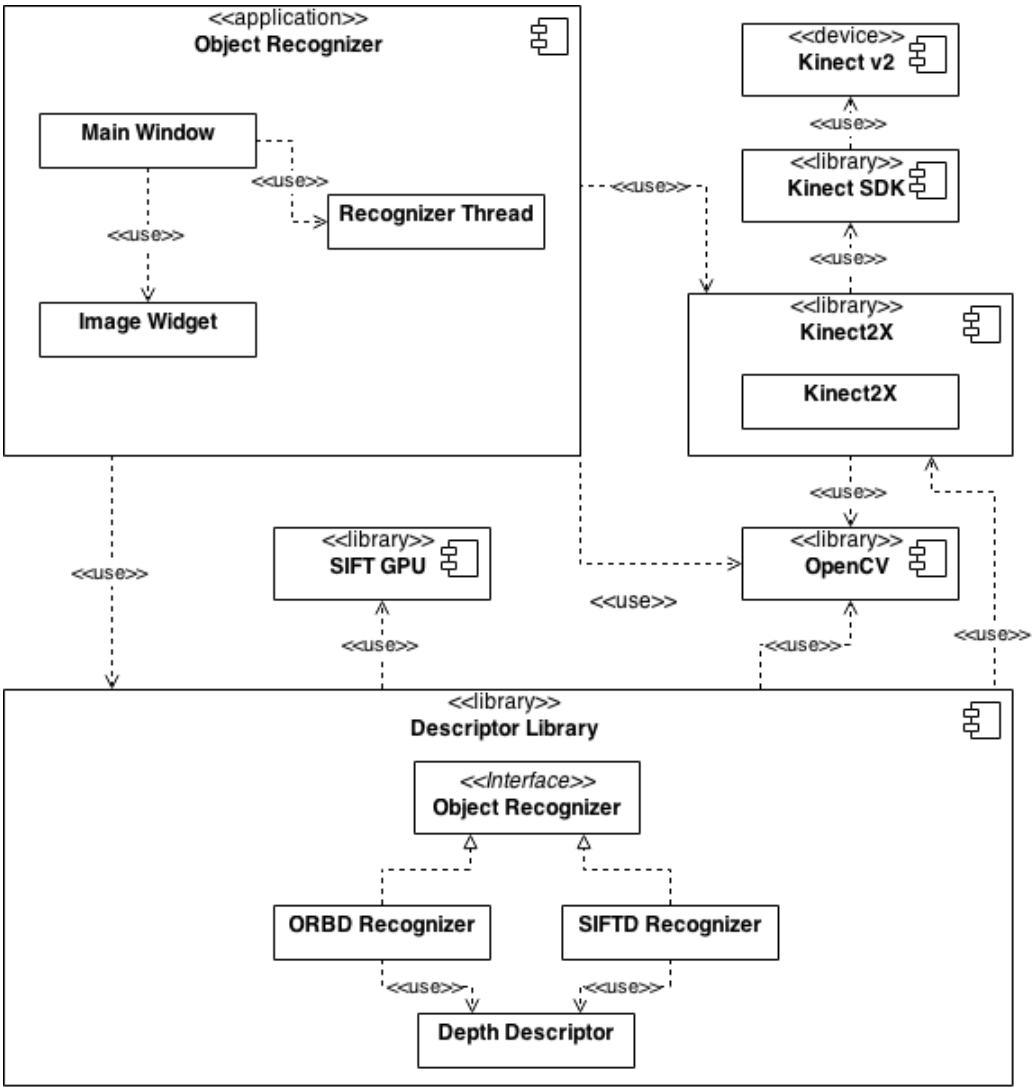


Figure 5.1. Application architecture

5.3. Kinect2X library

Kinect2X library was created with the aim of providing high-level interface for the user who wants to use the Kinect v2 device in their applications. In addition we added OpenCV support to the library. Thanks to it we are able to get the images in *cv::Mat* format for further processing and call some methods from the library with the entry parameters as they are used in OpenCV. The library consists of the following methods:

- Sensor initialization & shutdown
- Opening the streams (both depth & color)
- Image acquisition
    - openCV and KinectSDK matrix datatype
- Converting 16 bit depth map to 8 bit
- Visualization of the depth frame
- Color frame alignment according to depth frame
- Depth frame alignment according to color frame
- Color coordinate (X,Y) alignment to the depth frame
- Depth coordinate (X,Y) alignment to the color frame
- Depth coordinates (X,Y) alignment to the real world coordinates
- Real world coordinates alignment to the Depth coordinates

The image alignment is made per pixel using the buffer as it is used in the Kinect SDK. The results are then stored in the matrix and returned. We include the code for the color alignment only as the rest of the methods are similar. To get the new position of pixel, we created the matrix of indices pointing to the position at the depth or color stream based on the request and the methods for it are already included in the Kinect SDK library.

```
Function alignColorFrame (width, height, frame)
mat_frame ← createEmptyMatrix()
frame_buffer ← createEmptyBuffer(width,height)
for index ← 0 until index < width * height do
        colorInformation ← getCoordinateFromMapper (index)
        frame_buffer (index) ← colorInformation
end
mat_frame ← createMatFromBuffer(frame_buffer)
return mat_frame
```

5.3.1. Kinect2X initialization

The Kinect v2 support only one default sensor connected through the USB 3.0 port to the computer. In our solution we have implemented the singleton instance of the class so we can make access to the same initialized device with set parameters in the Object Recognizer application for image stream acquisition and Descriptor Library for the use of image alignment and depth information extraction.

5.4. Descriptor Library

The strategy design pattern used in the previous version of Object Recognizer was moved to the descriptor library and ORB implementation on the GPU was added to it. The pattern implements virtual methods which are overridden with the desired method from the class specified at the configuration file.

Our solution support two methods for object recognition. First implementation is the GPU implementation of SIFT descriptor [29] as a standalone library and the second one is the ORB GPU implementation from the OpenCV library, build on the CUDA version 6.5.

Based on the configuration file, both of the mentioned descriptors can be used separately as standalone descriptors for the segmented object or as a part of the cascade of descriptors where the first level of recognition is done by our implementation of Depth Descriptor.

5.5. Depth Descriptor

In this part we are going to describe the process of creating the Depth descriptor vector. Important parts of the algorithm will be provided with pseudo-codes. In addition we will make an experimental evaluation of the extracted descriptor for the purpose of implementing additional part of recognition based on decision, whether the present object has flat surface or not.

5.5.1. Extraction of Depth Descriptor features

Let's assume we have already managed to get key-points from the SIFT detector. To create descriptor vector for each key-point, we need to loop through them and extract desired information. In the depth descriptor extraction we will work with the depth data, therefore we

need to align the color coordinates which were detected by SIFT detector to the depth coordinates.

```
Function computeDescriptor (keyPoints)
depthDescriptorVector ← createEmptyDescriptorVector()
foreach keyPoint ∈ keyPoints do
        depthPosition ← getDepthFromColorCoord (keyPoint)
        triangles ← getTriangles(depthPosition, radius)
        averageNormal ← getAverageNormal(triangles)
        averageAngle ← getAverageAngle(averageNormal, triangles)
        std ← getStd(averageNormal, triangles)
        maxmin ← getMaxMin(averageNormal, triangles)
        globalNormal ← globalNormal + averageNormal
        depthDescriptor ← storeValues(averageAngle, std, maxmin)
end
```

Using the pseudo code above, we can store three out of four values for the descriptor. For the last value, we need first to compute global normal vector of the object representing average normal vector out of all normal vectors from the triangles. Hence, the function continues with another loop.

```
Function computeDescriptor (keyPoints)
...
globalNormal ← globalNormal / size(keyPoints)
foreach keyPoint ∈ keyPoints do
        averageNormal ← getAverageNormal(triangles)
        globalAngle ← getAngle(averageNormal, globalNormal)
        depthDescriptor ← storeValues(globalAngle)
end
```

Based on the depth at each key-point we need to determine the size radius for the triangles. As we need to compute the size in real world coordinates, we need to make an alignment of the key-point depth coordinates to the real world coordinates and after extracting the triangle points, transform them back.

```
Function getTriangles(depthPosition, radius)
realPosition ← getRealFromDepthCoord(depthPosition)
for i=1 to 4 do
        firstPoint ← getFirstPoint(realPosition, radius)
        secondPoint ← getSecondPoint(realPosition, radius)
```

```
        thirdPoint ← getThridPoint(realPosition, radius)
        trianglePoints ← getDepthFromRealCoord(firstPoint, secondPoint, thirdPoint)
        triangles ← add(trianglePoints)
    end
    return triangles
```

The formulas of extracting the specific descriptor values have been mentioned in the design chapter and do not be described with the pseudo code as the steps for their extraction are straight-forward.

5.5.2. Key-point detection based on depth image

At the design part of the work we have talked about the key-point detection based on the depth image. During evaluation part we measured the time each step of the process takes to recognize the object. We come to the conclusion that the part, where we need to remove large number of key-points detected at the border of the image (because of the segmentation mask) takes approximately ~ 100 milliseconds, resulting in significant drop of frame rates.

Same process would apply to key-point detection based on the depth image which will drop the frame rate even lower. Instead, we extracted the descriptor vector from the key-points which were detected using the SIFT detector (or FAST detector for the ORB descriptor) and evaluated the computed values. We were looking for the statistical information related the values of descriptor vector and their comparison for flat and non-flat objects.

5.5.3. Evaluating the surface based on DD values

We have taken the extracted Depth Descriptor values to make scatter plot for each feature. In the next figures we compared the objects with flat and non-flat surface. For the increased robustness of the experiment, each of the objects were taken from different views and added to the plot.

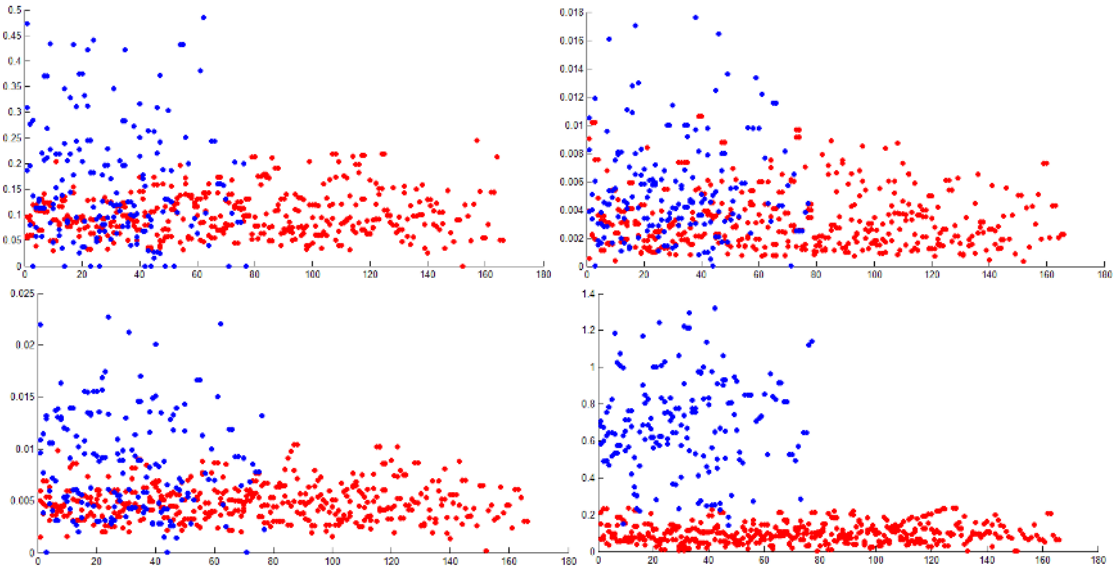


Figure 5.2. Scatterplot images of four depth descriptor features.

Figure 5.3. Example of objects used for evaluation.

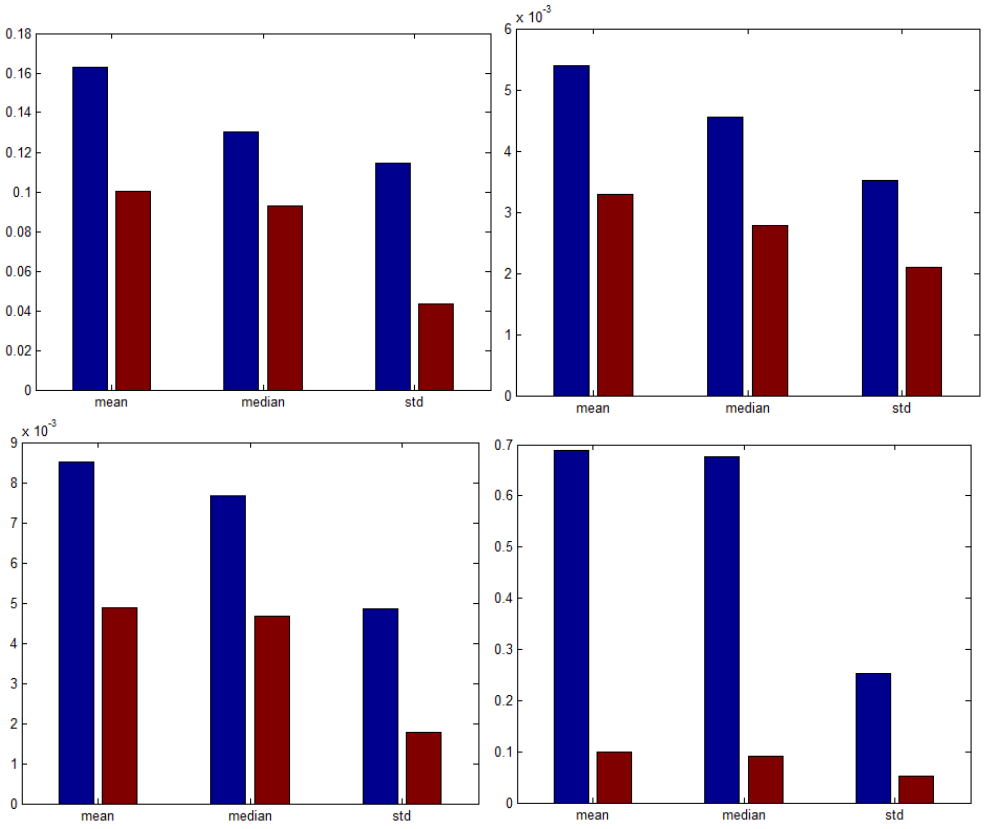


Figure 5.4. Bar plots of mean, median and standard deviation values of all features for non-flat (blue) and flat (red) objects.

Based on the results provided we have decided to make the surface prediction based on the fourth value, which represents the global angle feature. Even when the difference between the objects are significant, during the experiments we came across some false predictions related the surface. Upon extracting the values for such objects we have found that in some cases, the outliers have become the issue.

5.5.4. Outlier removal

To remove outliers from the depth description we detect upper and lower quartile from the descriptors vector values. We subtracted those values and multiplied it by the number 1.5 which represents the outer fence value. If the actual value of the descriptor were under/above the quartile plus the outer fence value it was considered as an outlier and whole descriptor were removed from the descriptor vector.

Function outlierRemoval(*depthDescriptor*)

---

lower_quartile ← getLowerQuartile(*depthDescriptor*)
upper_quartile ← getUpperQuartile(*depthDescriptor*)

```
outer_fence ← (upper_quartile - lower_quartile) * 1.5
foreach depthDescriptor do
        for i=1 to 4 do
                if depthDescriptor[i] < lower_quartile - outer_fence
                or depthDescriptor[i] > upper_quartile + outer_fence
                        removeDescriptor(depthDescriptor,i)
        end
end
return depthDescriptor
```

Removing the outliers could not even improve the surface prediction but also overall recognition based on the depth stream. Now we are able to set the threshold values for the surface prediction.

5.5.5. Estimating the threshold values for flat surface prediction

In our application we have chosen to rely on the fourth value of the descriptor vector for surface prediction as after the several experiments it has proven to have most distinguishable values between flat and non-flat surface. The decision is based on the mean and standard deviation values and could end in three different states:

- The object belongs to the flat surface objects dataset
- The object belongs to the non-flat surface objects dataset
- The object could belong to both of the above mentioned datasets

We added the third class where we cannot decide for sure about the object surface, because this prediction is the first level of the recognition and will affect all recognition processes done after. For example if we want to detect the paper box, which one side is flat object, but the segmentation mask also takes neighbor side of the box, descriptor values will change and it can be interpreted as non-flat object. Therefore the middle class containing flat and non-flat objects at the same time could increase the robustness of the algorithm. The threshold values for the prediction has been set to these values:

- **Flat objects**
    - standard deviation under 0.15
    - mean under 0.15

- **Flat and non-flat objects**
    - Standard deviation above 0.15 and under 0.20
    - Mean above 0.15 and under 0.35
- **Non-flat objects**
    - Standard deviation above 0.20
    - Mean above 0.35

The experiment showing us the accuracy of this prediction along with the threshold setting adjustments can be found at the result section.

5.5.6. Descriptor matching

The depth descriptor values are stored as the real numbers. To compare the results we use the brute force matching approach where we find best pairs of descriptors using the Euclidean distance measurement. We will use the knn (k=2) matching which is already used for SIFT matching in our application where we also consider the ratio between the closest and the second closest descriptor.

As the depth values for the flat objects will have the same values we pass indices of those object right to the last level of recognition without matching phase.

5.6. Color (Intensity) Descriptor

For the next level of recognition from color stream we implemented two well-known descriptors. Both of the descriptors are implemented on the GPU unit and therefore we could process the full HD images in near real time.

SIFT GPU Implementation

The implementation of SIFT on the GPU unit was taken from the University of North Carolina: SiftGPU library [29]. It supports GLSL by default and CUDA for the users with NVidia graphic cards which we also use in our implementation. Key-point detection is provided with library based on Difference of Gaussian method and we also use it in later implementation of depth descriptor.

ORB GPU Implementation

For the ORB GPU descriptor we use the implementation already included in the OpenCV library with CUDA support. For key-point detection ORB descriptor use FAST detector method which can be computed on the GPU as well.

Descriptor matching

To declare the object as one from the trained objects we compare the ratio between good matches and the number of all key-points. For both SIFT and ORB descriptor we use brute force matching. The SIFT descriptor which vector is filled with float datatype numbers is matched using the Euclidean distance and for binary ORB descriptor we use Hamming distance.

6. Results

We have implemented the depth descriptor and now we will evaluate its contribution to the recognition. Several measurements will be executed. In the first measurement we are going to take closer look at the number of objects which passed through pre-selection. The second test will measure the robustness of the depth description pre-selection in comparison to using just the SIFT descriptor in a way, we will compare the number of recognized objects for both cases. Next, we will take a look at how accurate can Depth Descriptor decide, if the object is flat or not. Last experiment will be regard the execution time of the object recognition for the Depth Descriptor followed by the SIFT descriptor. The results will be compared with the object recognition while using the SIFT descriptor alone.

6.1. Dataset

Our tested dataset were created in order to obtain reliable results. The set of toys fulfill very well the requirements of the application and also has a large variability necessary for the testing. Hence, various toys of different size and shape were chosen.

Figure 6.1. Sample of dataset used for the evaluation.

6.2. Hardware

The following hardware was used for the evaluation:

- Laptop with CPU Intel Core i7, 3632QM, 2.2 GHz, GPU NVidia GeForce GT635M and RAM 8GB DDR3 1600Mhz.

6.3. Depth Descriptor pre-selection

First evaluation has been done with the aim of how many objects passed through the depth descriptor to later color recognition. Objects are matched using the knn classifier using the k=2 with respect to the ratio threshold between two closest matches. Changing the ratio threshold could improve the processing time but will also increase the number of possible matches. The ratio threshold for the matching in our experiment has been set to the default value of 0.85.

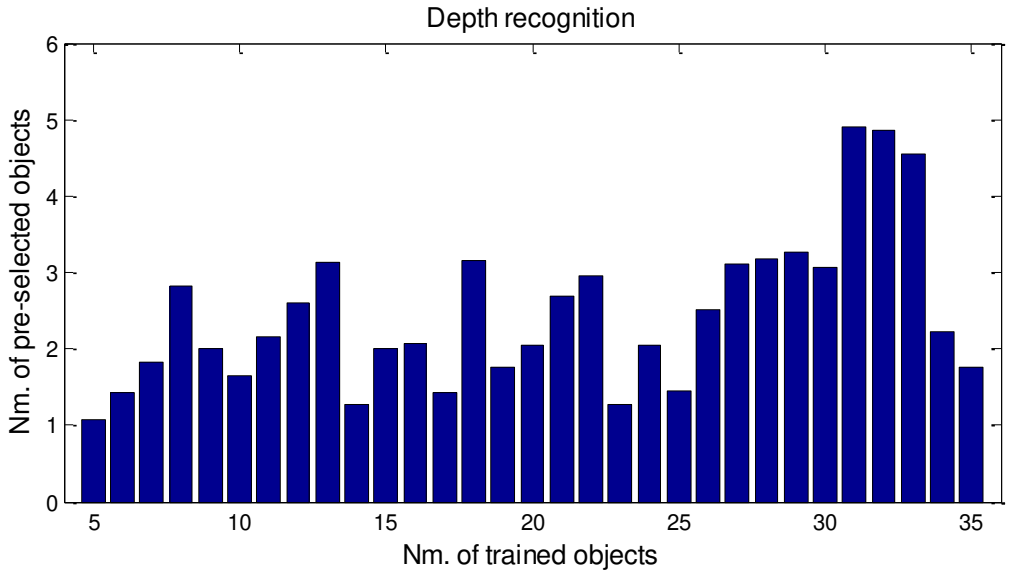


Figure 6.2. Number of objects after pre-selection.

With increasing number of trained objects, our depth descriptor were able to make a pre-selection and filter out most of the undesired objects. We can also see that the number of objects which passed the matching phase is relatively stable and could improve the matching speed for larger databases.

6.4. Evaluation of Depth Descriptor robustness

To evaluate the next experiment, we have acquired 30 images (color image, depth image and mask image) for total of 26 different objects, containing both flat and non-flat objects. In sum we have compared 780 different object views. Images we have acquired and stored were captured from different views and angles which are hard to recognize even for standalone SIFT descriptor. We have done so in order to fully and more precisely evaluate the actual robustness of the depth description. We compared the percentage of recognized objects

for both standalone SIFT and SIFT with Depth Descriptor and computed the difference. The results can be found in the following table.

	Recognized objects	Not-Recognized objects
SIFT descriptor	50.26%	49.74%
Depth + SIFT descriptor	48.59%	51.41%

Table 6.1. Percentage of recognized objects for Depth + SIFT descriptor according to SIFT only.

For the following results we can assume that while using the Depth Descriptor for the pre-selection will worsen the prediction by approximately 3.32% which is acceptable result for the implemented settings. We are able to lower the gap by decreasing the ratio threshold during the matching phase for knn matcher, but that will result in increased number of pre-selected objects and therefore slower recognition.

In this part, we also take closer look at the pattern size. During the implementation phase we have set the size of the triangle pattern to 15 millimeter radius. We have evaluated the recognition accuracy for the additional values of 10 mm, 20mm and 25mm. The results show us that the pattern size of 15mm suits well for the current settings. Table with the results can be seen below.

Pattern size	Recognized objects	Not-Recognized objects
10 mm	47.31%	52.69%
15 mm	48.59%	51.41%
20 mm	48.21%	51.79%
25 mm	46.79%	53.21%

Table 6.2. Percentage of recognized objects for different pattern size.

6.5. Evaluation of surface prediction

In order to evaluate the prediction of the surface with the values described in the implementation part, we divided the experiment into two phases. We will evaluate both mean value and standard deviation value separate and compare the results. For the experiment we used different flat and non-flat objects resulting in 120 different object images captured from

different views. Based on the evaluations we decided to focus more on the surface prediction based on the mean value of descriptor vectors.

Evaluation of mean value

The experimental setup for the mean value was:

- **Flat objects**
  - Mean value under 0.15
- **Flat and non-flat objects**
  - Mean value above 0.15 and under 0.35
- **Non-flat objects**
  - Mean value above 0.35

The results are shown in the following table:

	Flat objects	Flat and non-flat objects	Non-flat objects
Flat objects	83.33%	15%	1.67%
Non-Flat objects	0%	0%	100%

Table 6.3. Surface prediction according to mean value.

The values which will lead to failure on detecting the object are in the bottom left (non-flat objects which were classified as flat objects) and up right (flat objects which were classified as non-flat objects) corners of the table. We can see that the estimated mean values for the selection suits well for the experiments made, with only 1.67% of false surface estimation for flat objects.

Evaluation of standard deviation value

The experimental setup for the standard deviation values was:

- **Flat objects**
  - standard deviation value under 0.15
- **Flat and non-flat objects**
  - Standard deviation value above 0.15 and under 0.20
- **Non-flat objects**
  - Standard deviation value above 0.20

Same procedure applies for the standard deviation value. In the first experiment results show us that the significant amount of objects (20%) were evaluated with false surface.

	Flat objects	Flat and non-flat objects	Non-flat objects
Flat objects	68.33%	11.67%	20%
Non-Flat objects	0%	0%	100%

Table 6.4. Surface prediction according to std value.

We advanced the standard deviation border values to the following settings:

- **Flat objects**
  - standard deviation value under 0.15
- **Flat and non-flat objects**
  - Standard deviation value above 0.15 and under 0.25
- **Non-flat objects**
  - Standard deviation value above 0.25

The number of flat objects recognized as non-flat were reduced, however some non-flat objects which were in previous experiment classified correctly, were moved to the middle class. Still we are able to recognize those objects and therefore overall accuracy will increase, but it can slower the recognition speed.

	Flat objects	Flat and non-flat objects	Non-flat objects
Flat objects	68.33%	15%	16.67%
Non-Flat objects	0	18.33%	81.67%

Table 6.5. Surface prediction according to mean value.

6.6. Comparison of matching time

For the next measurement we have trained same dataset of objects and measured time needed for descriptor matching. The next figure show us the time difference of the matching phase when using the SIFT descriptor only and SIFT descriptor with the pre-selection based on our proposed depth descriptor.

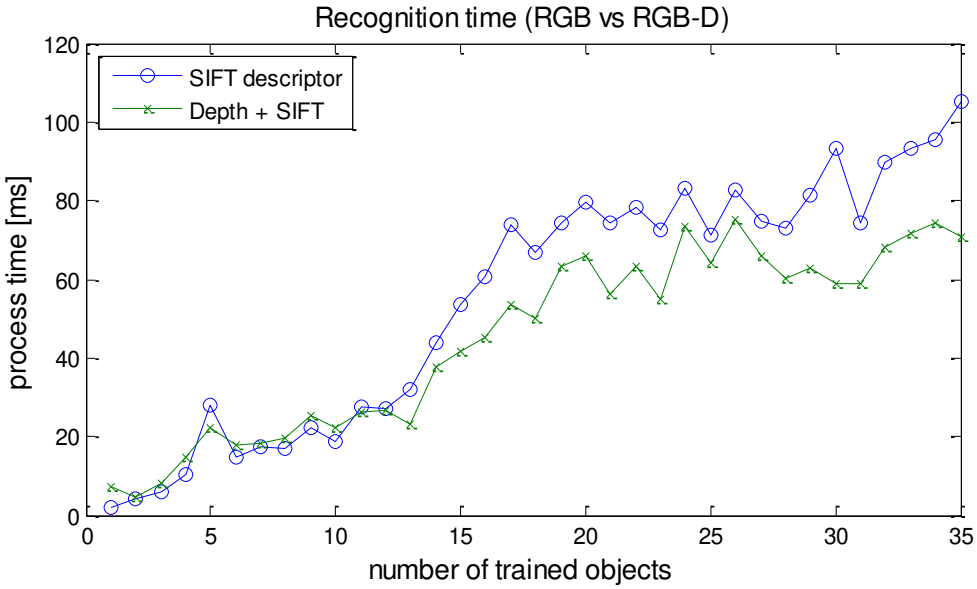


Figure 6.3. Comparison of time needed to match descriptors related to the default solution.

We can see that for the first ten objects the processing time of our proposed method is slightly worse, because the recognition cannot yet fully benefit of pre-selection phase. The change comes with more trained objects. Thanks to the pre-selection of the objects we are able to detect object faster than using only the color-based descriptor.

7. Conclusion

In this work we have analyzed the methods of visual object recognition based on the bottom-up approach of local descriptors. We have talked about the well-known local descriptors based on the color (intensity) image like SIFT, SURF, BRIEF, ORB, FREAK or HIPS as well as about key-point detectors which are used for the mentioned methods. We have also used SIFT and ORB descriptors in our implementation as they represent robust invariant descriptors with in the first case numerical and second binary descriptor vectors.

We took closer look at the local descriptors which use another data for the object recognition: depth frame. Using depth information in object recognition has been taken into consideration and few good descriptors were reviewed. In our research we mentioned NARF descriptor which can be found in the PCL Library, MeshDOG or current state of the art for the color & depth recognition, the CSHOT descriptor.

Because of the new version of Kinect which came to the market, considering the better fidelity and technology which we can use to extract depth information from the scene, we have decided to create a new, standalone local descriptor which will be based only on the depth data acquired from the depth sensor. We are aware that depth information still cannot overcome the robustness and precision of color description, but we will use it for object pre-selection. The overall recognition will be in the form of cascade of descriptors.

We have implemented the depth descriptor based on the statistical information over depth map acquired form the latest Kinect device. Our experiments show us the potential of faster recognition using the “cascade of descriptors” method over larger dataset. In addition we used the values from the descriptor vector to create another level of pre-selection in which we can predict the shape of the surface area and match objects accordingly to the shape.

The speed of the recognition can be improved in the future with additional decision making processes like the color histogram as the background around the object is removed with segmentation method.

8. References

[1] SZELISKI, Richard. Computer vision: algorithms and applications. Springer, 2010.

[2] GRAUMAN, Kristen; LEIBE, Bastian. Visual object recognition. Morgan & Claypool Publishers, 2011.

[3] Microsoft Kinect Website: http://msdn.microsoft.com/en-us/library/hh855347.aspx ; http://www.microsoft.com/en-us/kinectforwindows/default.aspx

[4] PENTLAND, Alex P.; TURK, Matthew. Face recognition system. U.S. Patent No 5,164,992, 1992.

[5] NAYAR, Shree K.; NENE, Sameer A.; MURASE, Hiroshi. Real-time 100 object recognition system. In: Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on. IEEE, 1996. p. 2321-2325.

[6] LINDEBERG, Tony. Feature detection with automatic scale selection. International journal of computer vision, 1998, 30.2: 79-116.

[7] TUYTELAARS, Tinne; MIKOLAJCZYK, Krystian. Local invariant feature detectors: a survey. Foundations and Trends® in Computer Graphics and Vision, 2008, 3.3: 177-280.

[8] MIKOLAJCZYK, Krystian; SCHMID, Cordelia. Scale & affine invariant interest point detectors. International journal of computer vision, 2004, 60.1: 63-86.

[9] MIKOLAJCZYK, Krystian; SCHMID, Cordelia. Indexing based on scale invariant interest points. In: Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on. IEEE, 2001. p. 525-531.

[10] MATAS, Jiri, et al. Robust wide-baseline stereo from maximally stable extremal regions. Image and vision computing, 2004, 22.10: 761-767.

[11] DONOSER, Michael; BISCHOF, Horst. Efficient maximally stable extremal region (MSER) tracking. In: Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on. IEEE, 2006. p. 553-560.

[12] ROSTEN, Edward; DRUMMOND, Tom. Fusing points and lines for high performance tracking. In: Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on. IEEE, 2005. p. 1508-1515.

[13] ROSTEN, Edward; DRUMMOND, Tom. Machine learning for high-speed corner detection. In: Computer Vision–ECCV 2006. Springer Berlin Heidelberg, 2006. p. 430-443.

[14] LOWE, David G. Object recognition from local scale-invariant features. In: Computer vision, 1999. The proceedings of the seventh IEEE international conference on. Ieee, 1999. p. 1150-1157.

[15] MIKOLAJCZYK, Krystian, et al. A comparison of affine region detectors. International journal of computer vision, 2005, 65.1-2: 43-72.

[16] BAY, Herbert; TUYTELAARS, Tinne; VAN GOOL, Luc. Surf: Speeded up robust features. In: Computer Vision–ECCV 2006. Springer Berlin Heidelberg, 2006. p. 404-417.

[17] CALONDER, Michael, et al. Brief: Binary robust independent elementary features. In: Computer Vision–ECCV 2010. Springer Berlin Heidelberg, 2010. p. 778-792.

[18] RUBLEE, Ethan, et al. ORB: an efficient alternative to SIFT or SURF. In: Computer Vision (ICCV), 2011 IEEE International Conference on. IEEE, 2011. p. 2564-2571.

[19] ALAHI, Alexandre; ORTIZ, Raphael; VANDERGHEYNST, Pierre. Freak: Fast retina keypoint. In: Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on. IEEE, 2012. p. 510-517.

[20] LEUTENEGGER, Stefan; CHLI, Margarita; SIEGWART, Roland Yves. BRISK: Binary robust invariant scalable keypoints. In: Computer Vision (ICCV), 2011 IEEE International Conference on. IEEE, 2011. p. 2548-2555.

[21] JAKAB, Marek. Planar object recognition using local descriptor based on histogram of intensity patches. In: Proceedings of the 17th Central European Seminar on Computer Graphics. 2013. p. 139-143.

[22] TAYLOR, Simon; ROSTEN, Edward; DRUMMOND, Tom. Robust feature matching in 2.3µs. In: Computer Vision and Pattern Recognition Workshops, 2009. CVPR Workshops 2009. IEEE Computer Society Conference on. IEEE, 2009. p. 15-22.

[23] RAČEV, Marek. Počítačové vizuálne rozpoznávanie objektov. Master thesis. Slovak University of Technology, Faculty of informatics and information technologies. 2013.

[24] STEDER, Bastian, et al. NARF: 3D range image features for object recognition. In: Workshop on Defining and Solving Realistic Perception Problems in Personal Robotics at the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS). 2010.

[25] STEDER, Bastian, et al. Point feature extraction on 3D range scans taking into account object boundaries. In: Robotics and automation (icra), 2011 ieee international conference on. IEEE, 2011. p. 2601-2608.

[26] TOMBARI, Federico; SALTI, Samuele; DI STEFANO, Luigi. A combined texture-shape descriptor for enhanced 3D feature matching. In: Image Processing (ICIP), 2011 18th IEEE International Conference on. IEEE, 2011. p. 809-812.

[27] TOMBARI, Federico; SALTI, Samuele; DI STEFANO, Luigi. Unique signatures of histograms for local surface description. In: Computer Vision–ECCV 2010. Springer Berlin Heidelberg, 2010. p. 356-369.

[28] ZAHARESCU, Andrei, et al. Surface feature detection and description with applications to mesh matching. In: Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on. IEEE, 2009. p. 373-380.

[29] Changchang, Wu., "SiftGPU: A GPU Implementation of Scale Invariant Feature Transform (SIFT)," < http://cs.unc.edu/~ccwu/siftgpu/>

[30] JAKAB, Marek; BENESOVA, Wanda; RACEV, Marek. 3D object recognition based on local descriptors. In: IS&T/SPIE Electronic Imaging. International Society for Optics and Photonics, 2015. p. 94060L-94060L-11.

Attachment A: Content of electronic media

Documents/MT_Jakab.pdf/.doc

- Master Thesis document in doc and pdf fomat.
- Papers published at SPIE and IITSRC 2015

Documents/Technical Documentation/

- Description of the methods used in the solution. Generated by Doxygen tool.
- Html and latex format

Libraries/sift gpu/

- SIFT GPU library (build under vs 2013 and CUDA 6.5)

Objects/

- Object images used for evaluation phase.

Source Code/ObjectRecognizer

- Implementation of the application.
- Including GUI application, Kinect2X library and Descriptor Library.
- All in one Visual Studio solution (VS2013)

Attachment B: User manual

After running the application you will get a simple windows with few options. The object recognition window contains the button which will start communication with the default Kinect v2 sensor and the next button in the recognition settings will be used to train the object while the connection is established and both depth & color stream from Kinect sensor are open.

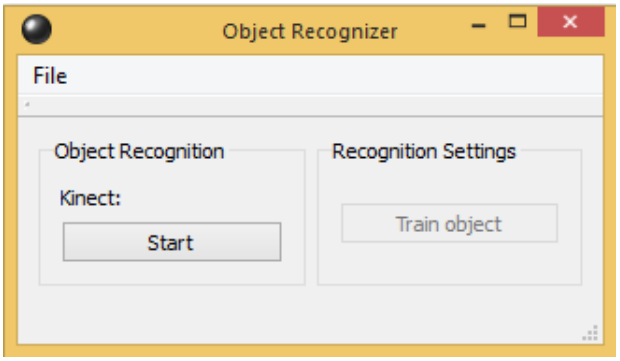


Figure B.1. Main window of application.

In order to start the recognition we first need to load all the settings. We can do so by clicking on the File at the top menu and selecting Open. Dialog window will pop up and ask us to select the *.INI* file with the recognition settings.

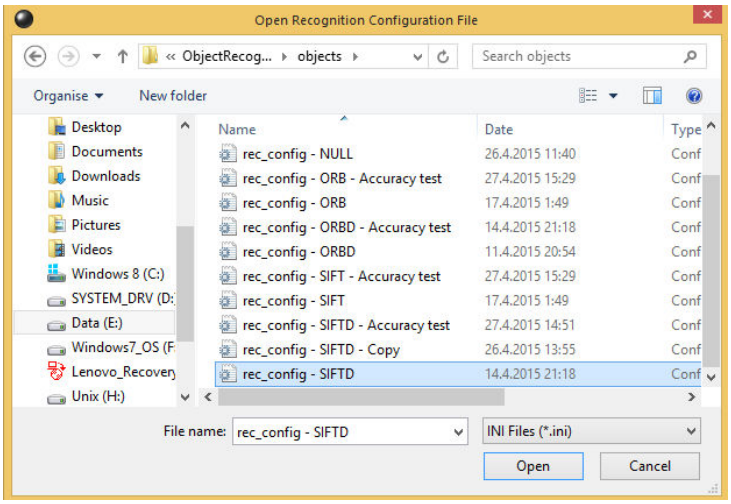


Figure B.2. Dialog window to open settings file.

After we click start with the loaded settings four additional windows will create with for the following streams: *Color Stream, Color Segment, Depth Stream and Aligned Depth Stream.* The Color Segment window is showing us the current object we are trying to

recognize. If we want to save the current object showing in the Color Segment, we can simply click on the train button in the main window. Dialog will popup asking us to specify the object name we want to store.

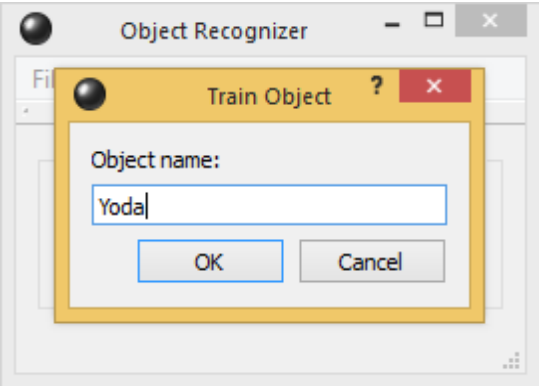


Figure B.3. Adding trained object name.

After a while the object will be saved and we are ready to recognize the stored object. The next figures are showing the output of Color Segment and Depth Stream windows.

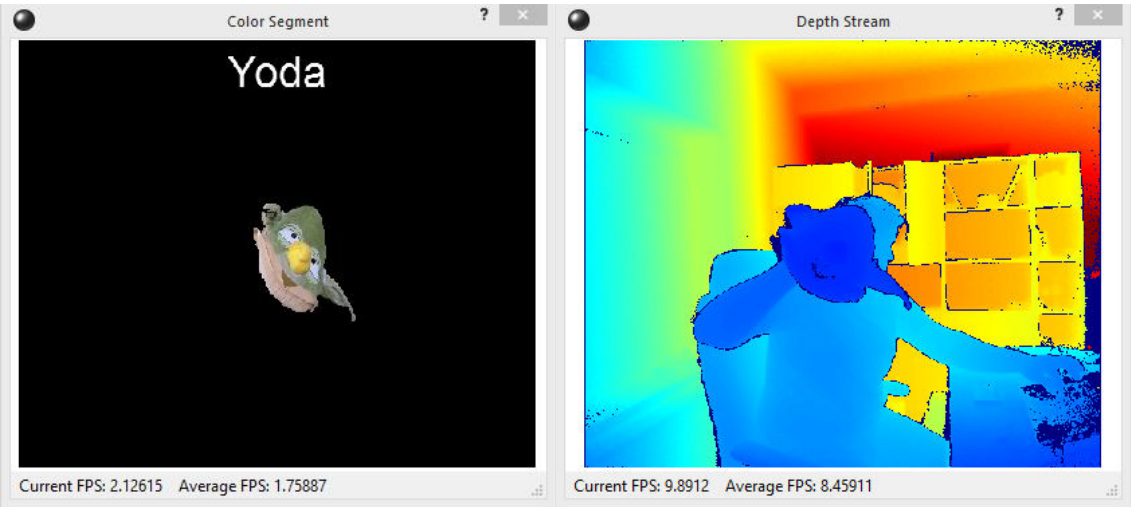


Figure B.4. Color segment and Depth stream windows.

Configuration file

In order to change additional settings we can edit the *.INI* file we use to load before we start the recognition. There are four main groups of settings we are able to change. *SensorConfig* group is used to change the settings for the color & depth stream from the Kinect device. It will resize the images acquired from the device as the default stream

resolution cannot be changed. *DescriptorConfig* group specify the local descriptor we want to use. The options are: SIFT, SIFTD, ORB and ORBD. Next, in the *MatchingConfig* group we can change the values for specific parts of the recognition, like the pattern size of the triangles for the depth descriptor, recognition confidence and so on. Last group named *objects* is used to train the objects from the stored images. The example of the file we use to set up the basic settings for the recognition application can be found in the lines below:

```
[SensorConfig]
sparameters\colorWidth=1920
sparameters\colorHeight=1080
sparameters\depthWidth=512
sparameters\depthHeight=424

[DescriptorConfig]
descriptorName=SIFTD

[MatchingConfig]
mparameters\minKeypointsPerObject=4
mparameters\distanceRatioThreshold=0.8
mparameters\minRecognitionConfidence=0.1
mparameters\minMatchesToFindHomography=4
mparameters\ransacOutliersRemovalEnabled=false
mparameters\patternSize=15

[objects]
1\name=1
1\views\1\color=/data/1_0_color.png
1\views\1\depth=/data/1_0_depth.png
1\views\1\mask=/data/1_0_mask.png
1\views\size=1
size=1
```

Attachment C: Technical documentation

The application is implemented in C++ as a Visual Studio 2013 solution with 3 independent projects. The external libraries used in the project with their versions are as follows:

- OpenCV 2.4.8
- Kinect SDK v2.0 1409
- SIFT GPU v400

Documentation

The description of the individual functions are generated by Doxygen tool and can be found in the attached DVD under *documents/technical documentation* folder in html and latex format.

Use case diagram

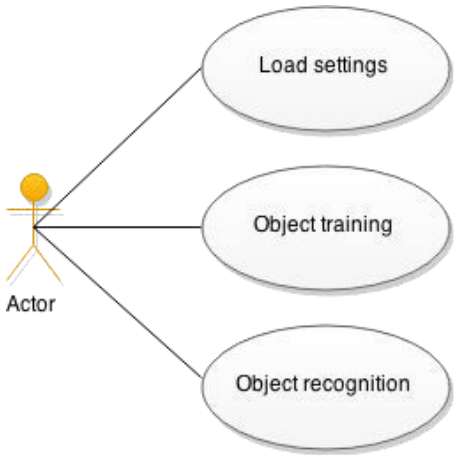


Figure C.1. Use case diagram.

Class Diagram

In the following part we include class diagrams generated in Visual Studio 2013 for the following projects in the solution:

Object Recognizer

Object Recognizer is the GUI application based on Qt framework. There are two main classes. *MainWindow* for managing the graphical interface, user input as well as image output from the application and recognition. *RecognizeThread* for actual recognition in separated thread.

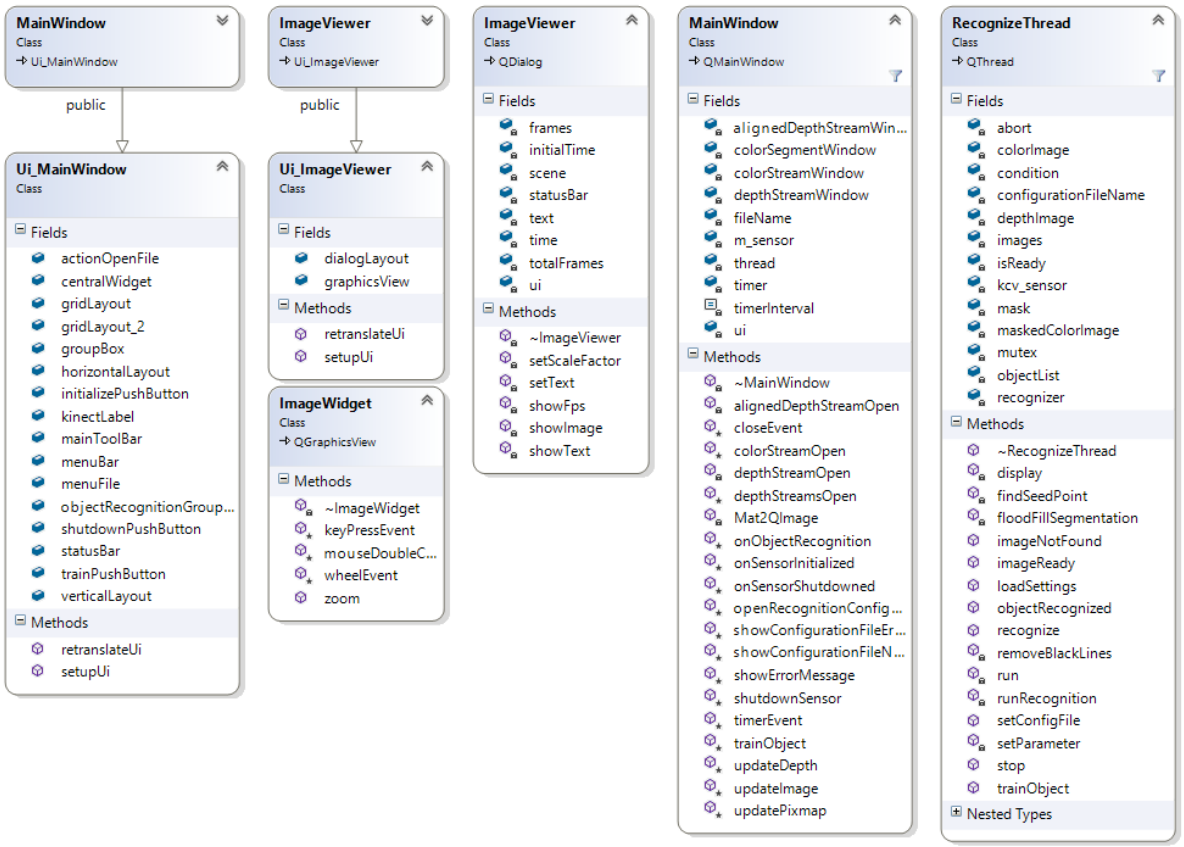


Figure C.2. Object recognizer class diagram.

Kinect2X

Kinect2X is a library used to establish communication with Kinect v2 sensor as well as acquiring color and depth images from it. There are also several functions used during descriptor extraction and supports OpenCV datatype for image.

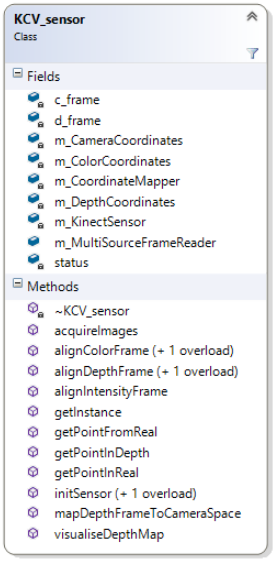


Figure C.3. Kinect2X class diagram.

Descriptor Library

Descriptor Library contains the main methods for descriptor extraction and object recognition. It contains virtual class *ObjectRecognizer* which other classes used for recognition inherits from.

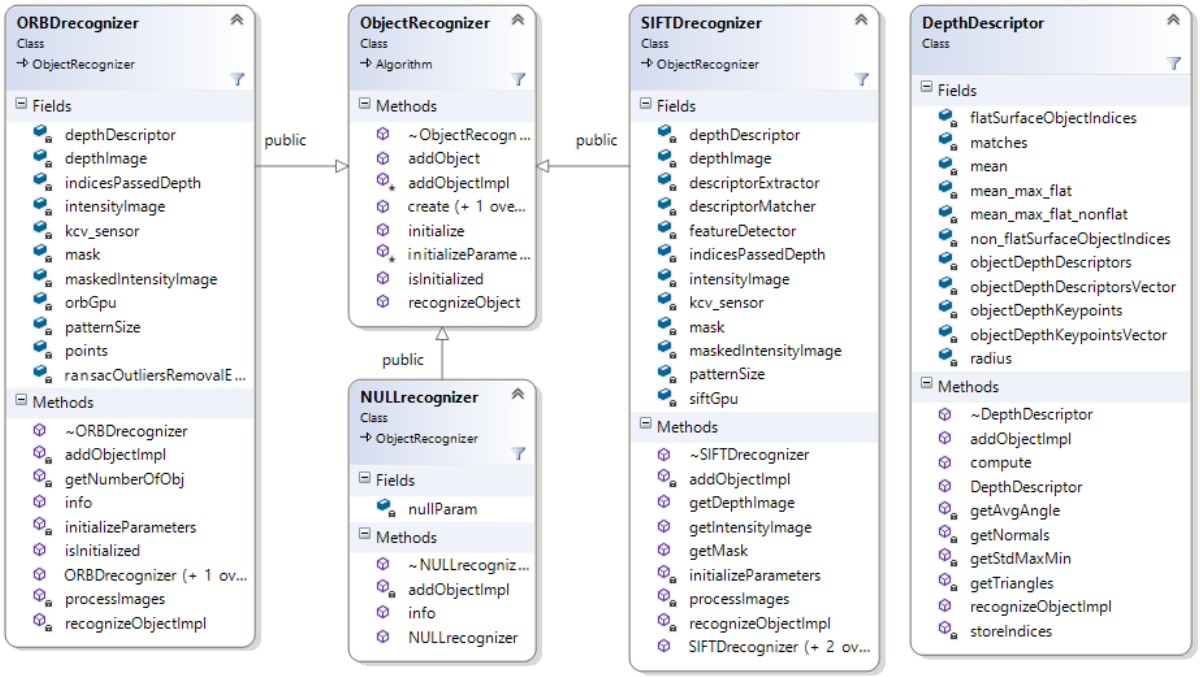


Figure C.4. Descriptor Library class diagram.

Depth Descriptor feature extraction

The main addition to the object recognition comes with the own implementation of depth descriptor. Here, we will show the algorithm for its extraction. The next for cycle show us the extraction of first three descriptor values. First, we need to extract the real word coordinates of the triangle around the key-point. In next steps, we extract the normal vectors and compute the actual values. We store the maximal values for later normalization and also compare if we successfully obtained good values for the descriptor. If any measurement error occurred during computation, we simply remove the descriptor.

```
for (int i = 0; i < objectDepthKeypoints.size(); ++i)
	{
		descriptor = objectDepthDescriptors.ptr<float>(i);
		std::vector<cv::Point> depthPoints;
		std::vector<cv::Point3f> realPoints;
		cv::Point3f realKeypoint;

		// get Triangles and extract points
		getTriangles(cv::Point((int)objectDepthKeypoints[i].pt.x,
(int)objectDepthKeypoints[i].pt.y), realKeypoint, depthPoints, depthImage);
		for (int j = 0; j < depthPoints.size(); ++j)
		{

			cv::Point3f realPoint1, realPoint2, realPoint3;
			// get real world coordinates
			if (kcv->getPointInReal(depthPoints[j], depthImage.cols, depthImage.rows,
realPoint1))
			{
				if (kcv->getPointInReal(depthPoints[j + 1], depthImage.cols,
depthImage.rows, realPoint2))
				{
					if (kcv->getPointInReal(depthPoints[j + 2], depthImage.cols,
depthImage.rows, realPoint3))
					{
						// store real coordinates
						realPoints.push_back(realPoint1);
						realPoints.push_back(realPoint2);
						realPoints.push_back(realPoint3);
					}
				}
			}
			j = j + 2;
		}
		cv::Vec3f avgNormal;
		std::vector<cv::Vec3f> normals;
		float avgAngle;
		// initialization to zero to avoid infinity values
		if (realPoints.size() == 0)
		{
			descriptor[0] = 0.0f;
			descriptor[1] = 0.0f;
			descriptor[2] = 0.0f;
			if (lastBadDescriptorIndex < 0)
			{
				lastBadDescriptorIndex = i;
				continue;
			}
			continue;
		}
		// get average normal and first feature -- average angle
		getNormals(realPoints, avgAngle, avgNormal, normals);
```

```
                float std, maxmin;
                // standard deviation and max-min
                // we translate surface to the keypoint position
                getStdMaxMin(realPoints, realKeypoint, avgNormal, std, maxmin);
                // store values for later normalisation
                if (!(avgAngle == avgAngle) || !(std == std) || !(maxmin == maxmin))
                {
                        descriptor[0] = 0.0f;
                        descriptor[1] = 0.0f;
                        descriptor[2] = 0.0f;
                        if (lastBadDescriptorIndex < 0)
                        {
                                lastBadDescriptorIndex = i;
                        }
                        continue;
                }

                if ((avgNormal[0] == avgNormal[0]) && (avgNormal[1] == avgNormal[1]) && (avgNormal[2]
== avgNormal[2]))
                {
                        good_normals++;
                        object_normal[0] = object_normal[0] + avgNormal[0];
                        object_normal[1] = object_normal[1] + avgNormal[1];
                        object_normal[2] = object_normal[2] + avgNormal[2];
                }

                // clear all
                realPoints.clear();
                depthPoints.clear();
                if (avgAngle > max_angle)
                        max_angle = avgAngle;
                descriptor[0] = avgAngle;
                if (std > max_std)
                        max_std = std;
                descriptor[1] = std;
                if (maxmin > max_maxmin)
                        max_maxmin = maxmin;
                descriptor[2] = maxmin;

                avg_normals.push_back(avgNormal);
                objectDepthKeypointsGood.push_back(objectDepthKeypoints[i]);
                if (lastBadDescriptorIndex < 0)
                        continue;
                std::memcpy(objectDepthDescriptors.ptr<float>(lastBadDescriptorIndex),
objectDepthDescriptors.ptr<float>(i), 4 * sizeof(float));
                ++lastBadDescriptorIndex;
        }
```

Fourth descriptor value

Last value of the descriptor is extracted in a similar way. As it was described, we compute average angle through all key-points and write down the angle between each key-point and average angle.

```
for (int i = 0; i < avg_normals.size(); i++)
	{
		// if average normal is finite & with no error
		descriptor = objectDepthDescriptors.ptr<float>(i);
		if ((avg_normals[i][0] == avg_normals[i][0]) &&
			(avg_normals[i][1] == avg_normals[i][1]) &&
			(avg_normals[i][2] == avg_normals[i][2]))
		{
			float angle;
			// compute global angle
			getAvgAngle(object_normal, avg_normals[i], angle);
			if (!(angle == angle))
			{
				descriptor[3] = 0.0f;
				if (lastBadDescriptorIndex < 0)
				{
					lastBadDescriptorIndex = i;
					continue;
				}
				continue;
			}
			descriptor[3] = angle;
			if (angle > max_object_angle)
				max_object_angle = angle;
		}
		else
		{
			descriptor[3] = 0.0f;
			if (lastBadDescriptorIndex < 0)
			{
				lastBadDescriptorIndex = i;
				continue;
			}
			continue;
		}
		objectDepthKeypointsGood.push_back(objectDepthKeypoints[i]);
		for (int j = 0; j < 4; j++)
			descriptorValues[j].push_back(descriptor[j]);
		if (lastBadDescriptorIndex < 0)
			continue;
		std::memcpy(objectDepthDescriptors.ptr<float>(lastBadDescriptorIndex),
objectDepthDescriptors.ptr<float>(i), 4 * sizeof(float));
		++lastBadDescriptorIndex;
	}
```

Attachment D: Resumé v slovenskom jazyku

Úvod

Vizuálna detekcia objektov patrí v súčasnosti medzi rozvíjajúce sa časti počítačového videnia. Dôraz pri rozpoznávaní sa kladie hlavne na rýchlosť, robustnosť a zabezpečenie toho, aby bolo možné objekty rozpoznávať aj pri rôznych uhloch natočenia a aj v prípadoch, kde nedokážeme zachytiť celý objekt na obraze. V práci sa ďalej venujeme rozpoznávaniu objektov pomocou lokálnych deskriptorov. Diskutujeme o možnostiach využitia hĺbkových dát pre potreby rozpoznávania 3D objektov v podobe vytvorených deskriptorov, ktoré spájajú informáciu o textúre spolu s hĺbkovou informáciou.

Analýza

Rozpoznávanie objektov možno vo všeobecnosti rozdeliť do dvoch kategórií:

- Rozpoznávanie generických objektov
- Rozpoznávanie špecifických objektov

Naša práca sa zaoberá práve rozpoznávaním špecifických objektov pomocou metód lokálnych deskriptorov. Vo všeobecnosti algoritmus pre rozpoznávanie funguje tak, že sa na vstupnom obrázku nájdu kľúčové body záujmu, ktorých okolie sa vhodným spôsobom opíše a uloží do číselného, prípadne binárneho vektora. Takto vytvorené vektory sa potom uložia a predstavujú náš objekt, ktorý chceme neskôr rozpoznať. Rovnaký postup sa následne aplikuje na obraze na ktorom chceme objekt rozpoznať. Rozpoznávanie prebieha na základe porovnávania vektorov deskriptorov a nájdenia korešpondujúcich párov deskriptorov.

Akokoľvek môže postup znieť veľmi jednoducho, pri procese dochádza k viacerým faktorom, ktoré môžu mať negatívny vplyv na rozpoznávanie. V prvom rade sa snažíme rozpoznávať 3D objekt na 2D reprezentácii. To znamená, že samotný objekt môžeme rôzne otáčať a stále bude jeho reprezentácia v podobe obrázku rôzna.

Ďalšie faktory ktoré negatívne vplývajú na rozpoznávanie a je potrebné ich pre algoritmus rozpoznávania brať v úvahu sú:

- **Svetelné podmienky** – Ak sa na obrázok pozeráme ako maticu ktorá obsahuje jednotlivé RGB hodnoty, tak rôzny vplyv svetla na scénu mení tieto hodnoty. Tým pádom napríklad dva rovnaké objekty, ktoré odfotíme z rovnakého pohľadu pri rôznom osvetlení budú predstavovať iné hodnoty v matici obrázka.
- **Škála a vzdialenosť** – V algoritme rozpoznávania je potrebné dbať aj na vzdialenosť objektu, ktorá rovnako vplýva zmenou dát ako i na farebnom obrázku, tak aj na hĺbkovej mape.
- **Rotácia** – Objekt, ktorý plánujeme rozpoznať nemusí byť vždy otočený rovnako. Pri našom riešení žiadame, aby naša implementácia bola voči tomuto vplyvu odolná.
- **Rotácia okolo osi** – Vo voľnom preklade rôzne uhly natočenia objektu. Vo výslednom obrázku predstavujú rôzne formy perspektívnych transformácii.
- **Šum** – Skreslenie obrázku, artefakty, ktoré sa na živej scéne nenachádza. Sú spôsobené hlavne samotným senzorom.
- **Prekrytie** – Vo veľa prípadoch nevidíme celý objekt, ale len jeho časť. Algoritmus rozpoznávania by mal byť schopný rozpoznať objekt aj pri jeho čiastočnej viditeľnosti.

Detekcia kľúčových bodov

Prvým krokom algoritmu lokálnej deskripcie je detekcia kľúčových bodov. Pre samotné rozpoznávanie je vhodná detekcia týchto bodov nesmierne dôležitá, pretože budú ďalej slúžiť pri vytváraní deskriptora. Hlavné vlastnosti, ktoré sa od kľúčových bodov požadujú, sú nasledovné:

- Aby bolo možné nájsť rovnaké body na objekte pri rôznych meraniach
- Aby boli tieto body nachádzané s určitou presnosťou
- Aby sa kľúčové body nachádzali na objekte aj pri rôznych natočeniach
- Aby boli body dostatočne reprezentatívne a zároveň odlišné
- Aby týchto bodov bolo viac a tým pádom by sa objekt mohol rozpoznať aj keď vidíme iba jeho časť

Niektoré deskriptory majú aj vlastné detektory kľúčových bodov. Medzi známe metódy detekcie takýchto bodov ale patria detektory ako DoG, LoG, FAST.

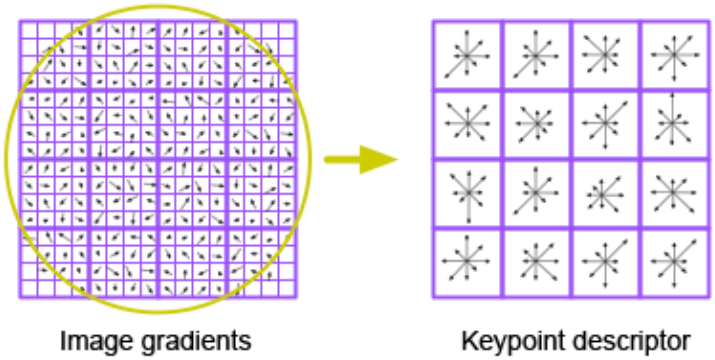
Deskripcia kľúčových bodov

Na základe toho, či sa jedná o deskriptory s reálnymi hodnotami alebo binárne deskriptory, bude prebiehať aj samotná deskripcia. Okolie kľúčových bodov sa zakóduje do vektorov deskriptorov, ktoré sa neskôr budú porovnávať. Medzi doteraz najznámejšie deskriptory patrí napríklad deskriptor SIFT.

SIFT deskriptor

Pre výpočet deskriptora je potrebné najskôr vytvoriť obrázkový gradient s veľkosťou a orientáciou v okolí kľúčového bodu. Veľkosť mriežky, nad ktorou sa táto operácia robí je 16 na 16 pixelov. Táto mriežka sa ďalej delí na menšie časti veľkosti 4 na 4 pixela, pre ktoré sa každej časti vypočíta histogram orientácii pre 8 smerov. Počas vytvárania sa dbá na to, aby jednotlivé pixely, ktoré sa nachádzajú bližšie ku stredu vzoru a tým pádom kľúčového bodu mali väčší vplyv pri tvorbe histogramu.

Vzhľadom na menšie časti s veľkosťou 4 x 4 a vytvorených histogramov s 8 hodnotami bude celkový SIFT deskriptor obsahovať 4 x 4 x 8 = 128 hodnôt. Následne sa normalizujú hodnoty aby sa dosiahla invariantnosť pred rôznymi svetelnými podmienkami.



Obrázok D.1. Deskriptor SIFT

Párovanie deskriptorov

Najjednoduchšia metóda párovania deskriptorov je takzvaná metóda hrubej sily, kde porovnávame každý deskriptor s každým. Vzhľadom na typ deskriptoru je možné použit viacero metrík porovnávania.

Pre číselné (dátový typ float) deskriptory je najčastejšie používaná metrika pre nájdenie podobnosti na základe Euklidovskej vzdialenosti.

Pre binárne deskriptory sa môže napríklad použiť Hammingová vzdialenosť, ktorá sa realizuje pomocou binárnej operácie XOR.

Návrh riešenia

Riešenie nadväzuje na prácu predošlého diplomového projektu, v ktorom bola vytvorená aplikácia rozpoznávania objektov pomocou deskriptora SIFT, ktorého vektor bol rozšírený o ďalšie 2 hodnoty na základe hĺbkovej mapy zo senzora kinect.

Na túto aplikáciu sme v našom riešení nadviazali a rozhodli sa viac venovať potenciálu rozpoznávania objektov pomocou hĺbkovej informácii. Na trh sa dostala nová verzia Kinect založená na metóde „času letu“ (time of flight), ktorá sľubuje presnejšie meranie vzdialeností.

Kaskádové rozpoznávanie

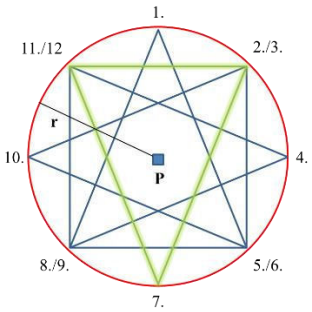
Vďaka lepšiemu hĺbkovému senzoru sme sa v práci rozhodli vytvoriť vlastný hĺbkový deskriptor, ktorý by bol schopný rozpoznávať objekty. Vzhľadom na fakt, že hĺbkové dáta stále nedosahujú dostatočnú kvalitu ako dáta farebnej kamery, rozhodli sme sa daný hĺbkový deskriptor použiť ako prvý vstup pre pre-selekciu objektov z daného naučeného datasetu. Tým pádom môžeme niektoré objekty, ktoré neprešli hĺbkovým deskriptorom vylúčiť, a neskôr vo fáze rozpoznávania objektov pre deskriptory založené na textúre urýchliť celkové rozpoznávanie. Zároveň vďaka hĺbkovej informácii pred samotnú hĺbkovú detekciu vieme pridať rozlíšenie objektov na základe tvaru jeho povrchu. Môžeme tak rozdeliť objekty s rovnou plochou od ostatných objektov.

Detekcia kľúčových bodov

Pre detekciu kľúčových bodov pri vytváraní vlastného hĺbkového deskriptora používame body, ktoré boli detegované pre deskriptory v ďalšej úrovni rozpoznávania. Neskoršie testy ukazujú na vhodnosť využitia týchto bodov a je tak možné ušetriť čas, kde ich pre hĺbkové dáta nemusíme znova detegovať. Zároveň dochádza k zrýchleniu aj vďaka segmentácii, kde vďaka hĺbkovej informácii ktorú máme k dispozícii dokážeme určiť pozadie objektu a odstrániť všetky kľúčové body, ktoré objektu nepatria.

Deskripcia kľúčových bodov

Pri tvorbe deskriptora sme sa rozhodli použiť hviezdicový vzor zo štyroch trojuholníkov vzhľadom na fakt, že tri body trojuholníka môžeme použiť pri určovaní normálového vektora plochy, ktoré tieto body opisujú.



Obrázok D.2. Vzor použitý pri tvorbe hĺbkového deskriptora.

Vektor hĺbkového deskriptora bude obsahovať 4 hodnoty, ktoré budú vypočítané štatistickými metódami. Takto dosiahneme invariantnosť voči rôznym natočeniam. Zároveň je veľkosť daného vzoru prepočítaná na veľkosť 15 milimetrov, čím dosiahneme to, že hĺbkový deskriptor bude možné využiť pri detekcii objektov z rôznych vzdialeností. V nasledujúcej časti si opíšeme jednotlivé hodnoty deskriptora.

Priemerný uhol

Prvá hodnota deskriptora je daná priemerným uhlom na základe všetkých trojuholníkov vo vzore. Z normálových vektorov vypočítame priemernú normálu, od ktorej zisťujeme uhly k jednotlivým normálovým vektorom. Z týchto uhlov je následne zistený priemerný uhol.

$$F\_avgAngle = \frac{1}{N}\sum_{i=1}^{N}\arccos\left(\frac{u_{i1}.v_{m1} + u_{i2}.v_{m2} + u_{i3}.v_{m3}}{\sqrt{u_{i1}^{2} + u_{i2}^{2} + u_{i3}^{2}}.\sqrt{v_{m1}^{2} + v_{m2}^{2} + v_{m3}^{2}}}\right)$$

Kde N je počet normálových vektorov.

$\vec{v}_m$ je priemerný normálový vektor.

$\vec{u}_i$ je normálový vektor pre trojuholník.

Štandardná odchýlka

Druhá hodnota deskriptora je štandardná odchýlka hĺbkových hodnôt vypočítanej z hĺbok v bodoch trojuholníkov.

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

Kde N je počet všetkých bodov vo vzore.

x_i je vzdialenosť bodu od priemernej plochy.

μ je priemerná vzdialenosť bodov od priemernej plochy.

Rozdiel maximálne ja minimálnej hĺbky

Tretia hodnota predstavuje rozdiel maximálnej a minimálnej hodnoty hĺbky v jednotlivých bodoch vo vzore.

Globálny uhol

Ako štvrtú hodnotu sme sa rozhodli pozrieť na objekt globálne. Zistili sme si priemerný normálový vektor pre celý objekt cez všetky vzory. Následne sme si pre každý deskriptor uložili hodnoty uhlu medzi celkovým normálovým vektorom objektu a priemerným normálovým vektorom pre každý kľúčový bod.

$$F\_globalAngle = \arccos\left(\frac{g_1.v_1 + g_2.v_2 + g_3.v_3}{\sqrt{g_1^2 + g_2^2 + g_3^2}.\sqrt{v_1^2 + v_2^2 + v_3^2}}\right)$$

Kde $\vec{g}$ je priemerný normálový vektor celého objektu.

$\vec{v}$ je priemerný normálový vektor kľúčového bodu.

$$g = \frac{1}{N}\sum_{i=1}^{N} v_i$$

Implementácia

Na základe návrhu sme zostrojili hĺbkový deskriptor s danými parametrami. Pri implementácii sme použili jazyk C++ a nasledovné technológie:

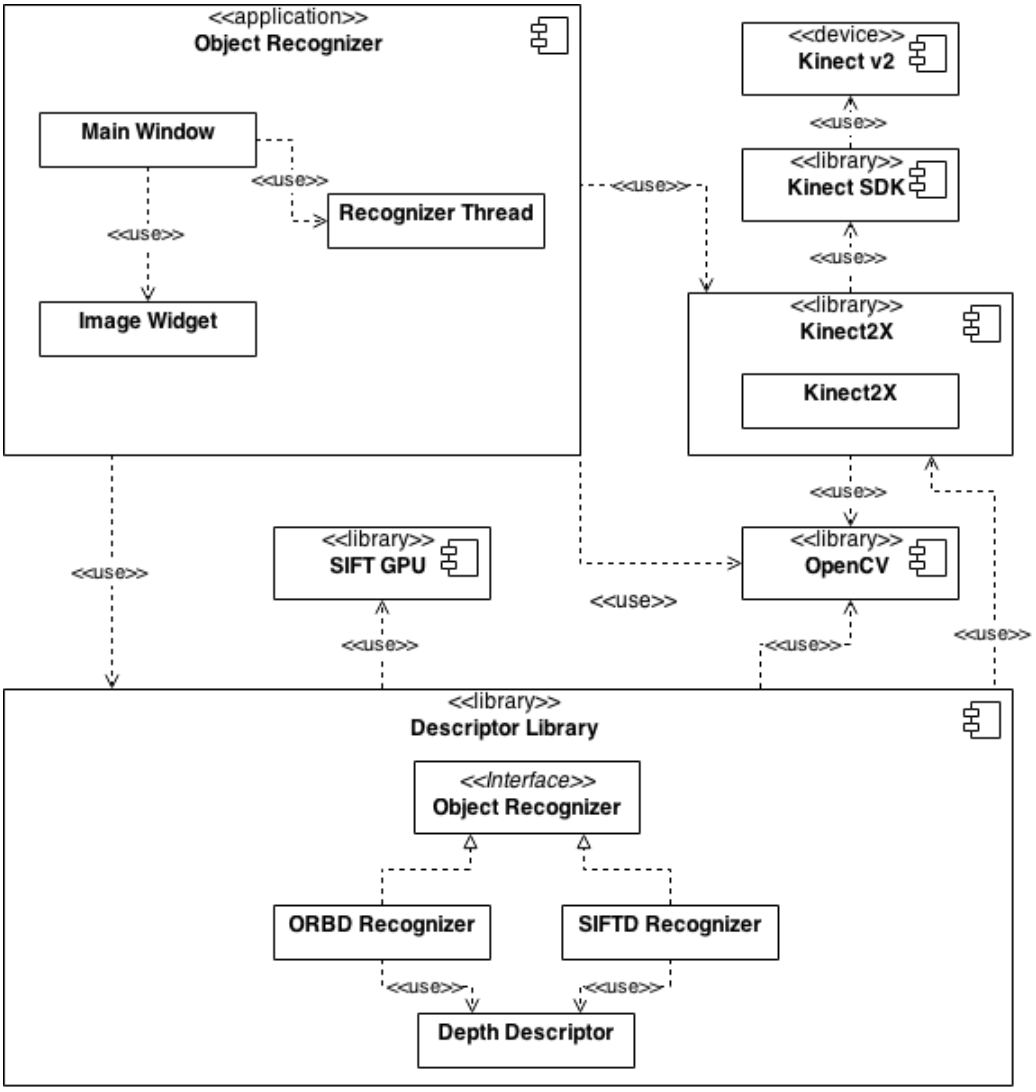
- Object Recognizer[23]
- OpenCV knižnica
- Qt Framework
- Kinect SDK v2.0

Architektúra systému

Pri implementácii sme dbali na rozdelenie jednotlivých modulov. Vytvorili sme knižnicu Kinect2X, ktorá má na starosti komunikáciu so zariadením Kinect ako aj mapovanie dát do formátu použiteľných pre OpenCV knižnicu. Zároveň knižnica ponúka mapovanie pozícií hĺbkových dát na pozície vo farebných dátach a naopak.

Všetky funkcionality deskriptorov sme umiestnili do samostatnej knižnice s názvom Descriptor Library. Nachádza sa tu implementácia deskriptorov SIFT a ORB na grafickej jednotke ako aj naša implementácia hĺbkového deskriptora.

Posledný modul tvorí samotná aplikácia, ktorá sa stará hlavne o používateľské grafické rozhranie a správu vlákien.

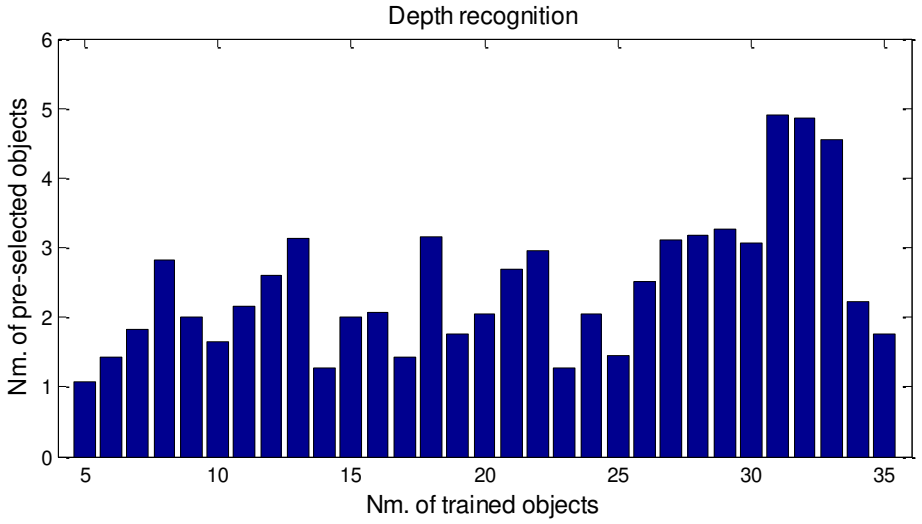


Obrázok D.3. Architektúra systému.

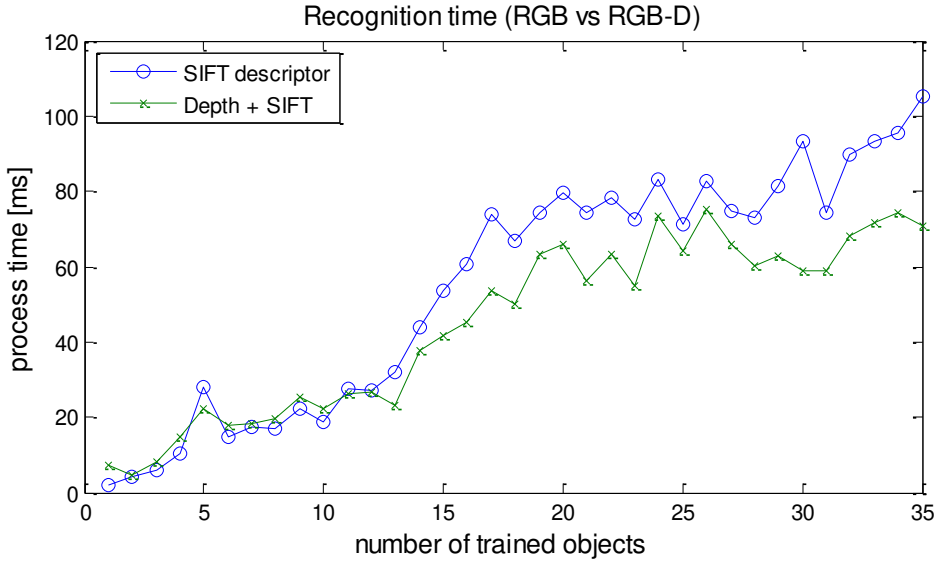
Jednotlivé časti implementácie v podobe pseudo kódov sú popísané v anglickej verzii implementácie.

Výsledky

Pri zhodnocovaní výsledkov sme vykonali viacero experimentov. V prvom rade nás zaujímal počet objektov, ktoré hĺbkový deskriptor posunie do ďalšej úrovne rozpoznávania. Vďaka pre-selekcie môžeme následne porovnať čas vykonávania rozpoznávania, ktorý by mal s rastúcim datasetom klesať. Výsledky jednotlivých testov nájdeme na nasledujúcich obrázkoch.



Obrázok D.4. Počet objektov ktoré prešli pre-selekciou.



Obrázok D.5. Porovnanie času rozpoznávania pri použití deskriptora SIFT s hĺbkovým deskriptorom a samostatného deskriptora SIFT.

Zhodnotenie a záver

V práci sme analyzovali jednotlivé metódy detekcie a zamerali sa hlavne na detekciu pomocou lokálnych deskriptorov. Preskúmali sme oblasť rozpoznávania objektov, kde bola využitá aj hĺbková informácia z dostupných senzorov. Rozhodli sme sa pre vlastnú implementáciu hĺbkového deskriptora, ktorá bola úspešná a zároveň pomáha v redukcii času rozpoznávania so zvyšujúcim sa počtom natrénovaných objektov. Hĺbkový deskriptor

prepustí do ďalšej úrovne rozpoznávania väčšinou vždy aj objekt, ktorý hľadáme a zhoršenie presnosti o 3.32% považujeme za akceptovateľné.

Problematika detekcie je stále ešte nepreskúmaná a možností vylepšenia detekcie neubúda. Medzi ďalšie možné spôsoby rozpoznania objektov napríklad môže patriť porovnanie veľkosti samotného objektu, prípadne porovnanie farebného histogramu.

Attachment E: Paper published at SPIE Electronic & Imaging conference in San Francisco, California

3D object recognition based on local descriptors

Marek Jakab[1], Wanda Benesova[2], Marek Racev[3],

Slovak University of Technology, Faculty of Informatics and Information Technologies, Slovakia

ABSTRACT

In this paper, we propose an enhanced method of 3D object description and recognition based on local descriptors using RGB image and depth information (D) acquired by Kinect sensor. Our main contribution is focused on an extension of the SIFT feature vector by the 3D information derived from the depth map (SIFT-D). We also propose a novel local depth descriptor (DD) that includes a 3D description of the key point neighborhood. Thus defined the 3D descriptor can then enter the decision-making process. Two different approaches have been proposed, tested and evaluated in this paper. First approach deals with the object recognition system using the original SIFT descriptor in combination with our novel proposed 3D descriptor, where the proposed 3D descriptor is responsible for the pre-selection of the objects. Second approach demonstrates the object recognition using an extension of the SIFT feature vector by the local depth description. In this paper, we present the results of two experiments for the evaluation of the proposed depth descriptors. The results show an improvement in accuracy of the recognition system that includes the 3D local description compared with the same system without the 3D local description. Our experimental system of object recognition is working near real-time.

Keywords: local descriptor, depth descriptor, SIFT, segmentation, Kinect v2, 3D object recognition

1. INTRODUCTION

Visual object recognition is still one of the biggest challenges in computer vision. One of the promising ways to approach this challenge seems to be the usage of local descriptors. The bottom-up approach using local descriptors is widespread and has also been the focus of research interest all over the world in recent years. Object recognition methods based on local descriptors are potentially applicable for applications working in near real time and the method also provides invariance to different illumination, scale, angle or rotation of the object, even if limited. Our goal is to provide a novel local descriptor which will extend the local description derived from an RGB image with a description of the neighborhood of a key point in the depth image (D). Therefore we can use the RGB-D sensors providing us with a color (RGB) and also a depth image (D).

Our main contribution is focused on the extension of the object recognition method using a SIFT feature vector by the 3D information derived from the depth mask. In this paper, we propose a novel local depth D-descriptor which represents a 3D description of the key point neighborhood. As so defined, the 3D descriptor can then enter into the decision-making process. Two different approaches could be considered:

- Object recognition using the original SIFT descriptor in combination with our novel proposed 3D descriptor, where the proposed 3D descriptor is responsible for the pre-selection of the objects.
- Object recognition using an extension of the SIFT feature vector by the depth local description as for example: absolute value of the difference between the depth minimum and depth maximum in the local area, standard deviation of the depth value in the local area.

In this paper, we present the results of two experiments for the evaluation of the proposed depth descriptors.

[1] marko.jakab@gmail.com

[2] vanda_benesova@stuba.sk

[3] marek.racev@gmail.com

Object recognition using local descriptors is typically based on the paired matching of all the trained object data with the data derived from an unknown object. The problem of a growing time requirement and a decreasing recognition accuracy becomes serious with the increasing number of objects in a dataset. Our first experiment is based on the idea that depth description will be used in the pre-selection step. We propose the usage of the RGB-D data obtained from the Kinect v2 sensor device for the purpose of a pre-selection of the objects for a subsequent SIFT-matching-based object recognition. That means that only a selected part of the objects from the trained dataset will be accepted by the first recognition part which uses the depth descriptors. Hence, the calculation time needed for the following SIFT matching will not exceed certain limits with growing number of objects in the dataset. For the purpose of the pre-selection, we use the information which provides us with complex and relatively precise measurements of the depth area around each of the key points. This information is statistically evaluated and then used in the descriptor creation.

In addition, object depth information could be used for a more global object description based on depth information, as for example the "global flatness" of an object.

The goal of the pre-selection is to speed up the decision process and also to improve the achieved recognition accuracy. The hardware setup for the presented evaluation has been completed using the Kinect v2 sensor device.

In the second experiment the well-known SIFT descriptor was extended by two values derived from the local depth description in the neighborhood of the key point. In the experiment, a whole recognition system has been evaluated in two ways: including original SIFT descriptor and our proposed extended descriptor SIFT-D.

2. RELATED WORK

The most relevant papers which present the research related to object recognition concerning both RGB and depth image (RGB-D) have been taken into consideration.

Normal Aligned Radial Feature (NARF) [1] [2] descriptor is able to extract and describe features from the 3D image data. The NARF descriptor has been developed with the goal to achieve two important objectives necessary for the object recognition. Firstly, the algorithm needs to select those key points which are located in the stable surface region. The reason is the robustness of the algorithm, since without this constraint, errors can occur in the computing of the normal vector in the point. Hence, these errors can subsequently cause errors in the matching phase. The second rule concerns finding the useful key points. Considering the fact that the 3D data are taken from the devices which are able to take only partial 3D image of the scene (like laser scanners, stereo camera or the Kinect device), shapes of the object will be different for different views of the object. Therefore, unique and high-quality selection of the key points is of high importance. The NARF descriptor uses star pattern for filling the descriptor. This means that way that each line of a star represents how different the pixel value is under the line.

Another depth descriptor is called Color-Signature of Histograms of Orientations (CSHOT) [3] [4] (color signatures of histograms). This descriptor contains both shape and texture information suitable for RGB-D (RGB & Depth) matching. The basic idea of forming this descriptor is the eigenvalue decomposition of the scatter matrix around the key point. A spherical grid is created around the point it could form a histogram of normal vectors defined for each sector of the grid. This descriptor could be an example of merging both color and depth information in one single descriptor.

Nascimento, Erickson R., et al. have proposed a Binary Robust Appearance and Normals Descriptor (BRAND) [5] which combines the appearance and geometric shape information from RGB-D images. In the first step, the scale factor using the depth information from RGB-D image is calculated. The scale factor is then used in the next step (dominant direction estimation) and in the feature analysis in the key point's vicinity. At last, the authors combine both appearance and geometric information to create key point descriptors that are expected to be robust, fast and lightweight. The authors also demonstrate that the descriptor is robust, invariant to rotation and scale, and provides reliable results in a registration task even when a sparsely textured and poorly illuminated scene is used. The main constraint of this method is, that a small irregularities of these surfaces can be confused with noise.

Lowe proposed a Scale Invariant Feature Transform (SIFT) [6]. SIFT combines a scale invariant region detector and a descriptor based on the gradient distribution in the detected regions. The main idea of the descriptor is to compose the local histograms of gradient locations and gradient orientations in one vector, wherein the contribution to the location and orientation in histogram bins is weighted by the gradient magnitude. The quantization of gradient locations and orientations makes the descriptor robust to small geometric distortions and small errors in the region detection. The

descriptor is finally arranged as a 128-dimensional vector of float numbers. The SIFT [6] descriptor is one of the older descriptors, but still robust, scale and rotation invariant and widely used.

3. DEPTH DESCRIPTOR (DD)

3.1 Descriptor pattern

To fill the descriptor vector, we use a star pattern created by four triangles which are rotated around the selected key point. Four triangles determine 12 points and the depth distances value of these 12 points are used in the computing of the depth descriptor vector. Each triangle defines a plane and hence the normal vectors of the planes could be calculated and subsequently used in the descriptor.

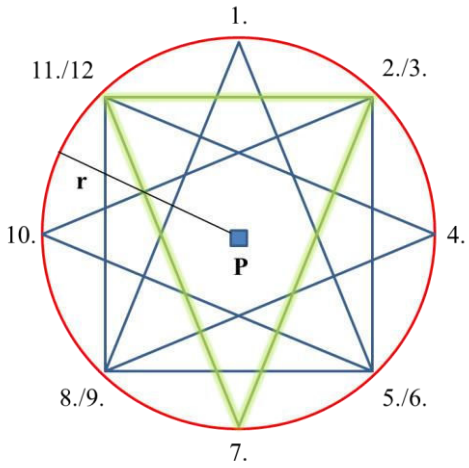


Figure 1. Star pattern around the key point *P*.

The points in the star pattern are defined with the constraint that the maximum distance radius in the real world metric is set to 15mm. To make the algorithm more robust, we can add more star patterns with different size radii.

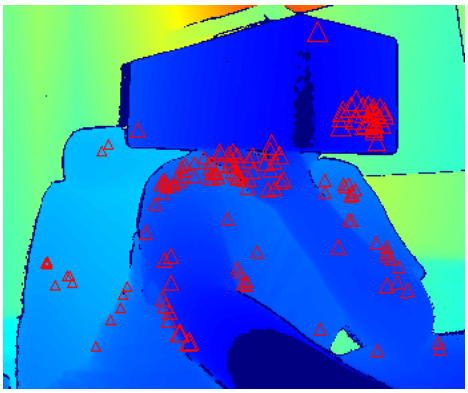


Figure 2. Visualization of dynamic change of the triangle size according to the depth.

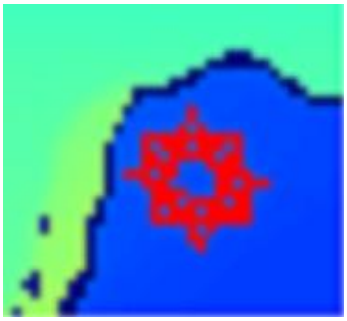


Figure 3. Visualization of descriptor pattern on trained object.

3.2 Descriptor vector

Four features, which are derived by statistical evaluation, have been taken for the definition of the depth feature vector. Three of them are based on the local surface description and the fourth feature also takes into account the global information of the object as will be described later in the chapter. In addition, we filter out those triangles which cannot provide the depth value in at least one point.

1st feature value derived from average angle referenced to a normal vector

The first feature in the depth feature vector is derived from the average angle of the normal vectors given by all triangles in the key point. Therefore we need to compute four normal vectors - one for each of the four triangles given by the pattern. Then the corresponding average vector of this key point will be calculated. This vector will be used as a reference in the next evaluation. In the next step, we can compute the difference angle between each of the normal vectors and the reference normal vector.

The first value of our depth descriptor *F_avgAngle* will be then calculated as follows:

$$F\_avgAngle = \frac{1}{N}\sum_{i=1}^{N}\arccos\left(\frac{u_{i1}.v_{m1} + u_{i2}.v_{m2} + u_{i3}.v_{m3}}{\sqrt{u_{i1}^{2} + u_{i2}^{2} + u_{i3}^{2}}.\sqrt{v_{m1}^{2} + v_{m2}^{2} + v_{m3}^{2}}}\right) \quad (1)$$

Where N is the number normal vectors/triangles.

$\vec{v}_m$ is the average normal vector.

$\vec{u}_i$ is the normal vector given by triangle.

2nd feature value derived from standard deviation of depth referenced to a plane perpendicular to the averaged normal vector

The second feature was proposed with the goal to create an efficient description of the local depth differences. To yield at least some acceptable kind of rotation and distance invariance of this feature, it is necessary to define a reference plane which is invariant to object rotation and to distance from the sensor. For this purpose, we have chosen the plane which is perpendicular to the reference averaged normal vector described in the previous section. The descriptor value is then given by the standard deviation of the depth differences between the actual depth value in a point and the reference plane. All points given by the descriptor pattern (see section 3.2.1.) shall enter into the evaluation.

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2} \quad (2)$$

Where N is the number of all points in the descriptor pattern (see section 3.1).

x_i is the distance of the depth value from the reference plane in the point i.

μ is the mean value of x_i (i=1...n).

3rd feature value derived from the difference between the maximum and minimum depth values referenced to a plane perpendicular to the averaged normal vector

Calculation of the depth value referenced to the reference plane is similar to those described in the previous section. Instead of the previous statistical evaluation by standard deviation, the simple difference between the maximum and minimum value of the recalculated depth value was taken into account as the next feature value.

Calculation of these values can be done very efficiently. The maximum and minimum value can be incorporated into the process of the calculation of the previous feature: standard deviation.

4th feature value derived by using a global angle

The previous three features' values have been designed as a local description without any global reference. The goal of the fourth feature value is to take into consideration also the global orientation of the object and hence a global normal vector was defined for this reason. The global normal vector is calculated as the average vector value out of all local normal vectors across all key points. This feature can provide additional information about the flatness of the object.

$$F\_globalAngle = \arccos\left(\frac{g_1.v_1 + g_2.v_2 + g_3.v_3}{\sqrt{g_1^2 + g_2^2 + g_3^2}.\sqrt{v_1^2 + v_2^2 + v_3^2}}\right) \tag{3}$$

Where $\vec{g}$ is the global average normal vector through all key points.

$\vec{v}$ represents the local average normal vector at a given key point.

$$g = \frac{1}{N}\sum_{i=1}^{N} v_i \tag{4}$$

The next figure shows a visualization of the difference angle between the global normal vector and local reference average normal vector. The bottom bar shows the possible values of color scale normalized into the range 0-1. The mean value of all angles is visualized as a black dot within the bar.

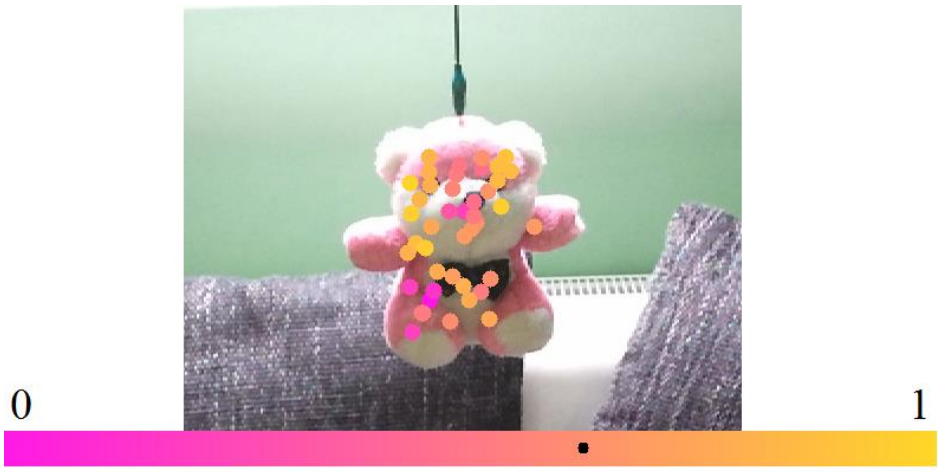


Figure 4. Visualization of angle at the given key point with global normal vector.

3.3 Invariance

The original SIFT descriptor is invariant to different scale, rotation and small perspective transformation of the object. Our goal is to develop a local depth descriptor which will extend the SIFT descriptor and hence we also aim to achieve a comparable rotation and scale invariance.

Scale invariance

The 3D information should be taken into account for the scale definition of the depth descriptor. As we know the distance of the selected key point from the sensor, we can easily determine the corresponding neighborhood size in pixels around the key point according to the real world distance.

Rotation & perspective invariance

To achieve invariance to different rotations of the object, we aim to fill the descriptor with the information which is defined as independent of the rotation and angle of the object. We have already described those values in previous chapters.

4. EXTENSION OF THE SIFT FEATURE VECTOR BY THE LOCAL DEPTH DESCRIPTION

Our idea was to extend the 128 feature values of the SIFT descriptor by additional feature values derived from the depth map which can provide additional local information. For this purpose we need to calculate a normal vector to the local area given by the key point. Then the local area around the key point will be transformed so that this normal vector is in the direction of the view. This transformation results in a new depth map which will be used for the next calculation of the additional features.

4.1 Extended SIFT feature vector (SIFT-D)

The first additional feature used in the extended feature vector is a standard deviation of the depth values in the local area surrounding the key point. The reference plane for the depth calculation is the plane perpendicular to the normal vector. Unlike the depth descriptor (DD) described above, the standard function in the PCL library was used for the calculation of the normal vector in the key point instead of the triangle pattern.

$$\sigma = \sqrt{\frac{1}{I_r \cdot I_c} \sum_{y=0}^{I_r-1} \sum_{x=0}^{I_c-1} \left(I(x,y) - \mu\right)^2} \tag{5}$$

$$\mu = \frac{1}{I_r \cdot I_c} \sum_{y=0}^{I_r-1} \sum_{x=0}^{I_c-1} I(x,y) \tag{6}$$

Where I(x, y) is a depth value at the position with coordinates x, y.

I_r is a number of rows in the segment.

I_c is a number of columns in the segment.

The second additional feature used to extend the feature vector is the difference between the maximum and minimum of the depth values in the described area [mm].

$$\alpha = \max(I) - \min(I) \tag{7}$$

An intuitive interpretation of the presented maximum range feature value is similar to the interpretation presented above: in case of a flat area the feature value is near or equal to zero.

If the corresponding normal vector was not able to be calculated due to the missing values in the depth map, zero values in place of both extending features are used.

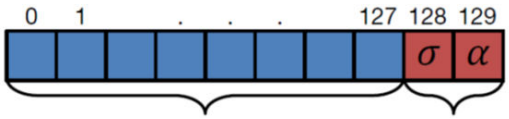


Figure 5. The extended proposed feature vector SIFT-D.

Actually, the development of the extension features was greatly limited by the depth map quality. The next limitation is given by the resolution of the depth sensor in relation to the recognized object size. For example, the NARF descriptor

can describe the area in much more detail, but the NARF-described area in the size of about 20 x 20 cm in real world coordinates is too large and hence not usable for our tested objects.

5. EXPERIMENTAL SETUP

5.1 Dataset

Our tested dataset was related to the final application: a self checkout in a shop. The set of toys fulfill very well the requirements of the application. The set also has a large variability necessary for the testing. Hence, various toys of different size and shape were chosen. The same example of the toys in the dataset are presented in the figure 6.

Figure 6. Objects used for training and measurement.

5.2 Hardware and software implementation

In the experiment for the evaluation of the proposed depth-based pre-selection the following hardware was used:

- Laptop with CPU Intel Core i7, 3632QM, 2.2 GHz, GPU NVidia GeForce GT635M and RAM 8GB DDR3 1600Mhz.

The test program was implemented in C++ with the library OpenCV. In the second experiment the PCL library was also included.

Implementation of the SIFT feature extraction and description was computed on the GPU unit [7]. Thanks to the GPU implementation we are able to use full HD images for color description and still provide the results in near real time.

Kinect v2 provides significantly higher depth fidelity (depth accuracy +- 1mm) and depth resolution (512 x 424) compared with the previous Kinect device.

Resolution of the RGB-D image acquired by Kinect v2 in the experiment with depth descriptor is presented in the next table:

Table 1. Image settings for color and depth processing.

Image	Description
Color image	1080p, 1920x1080 color image from Kinect v2 RGB sensor
Depth image	512x424 pixel resolution image from Kinect v2 Depth sensor

An older version of sensor Kinect (Kinect for Windows) was used in the experiment presented as the extension of SIFT descriptors

The project and the source code are available on the web page:

http://vgg.fiit.stuba.sk/3d-object-recognition/

6. PROPOSED DEPTH DESCRIPTOR (DD) EVALUATION

6.1 Overview of the processing pipeline

The whole pipeline of the experimental setup is presented in the next figure.

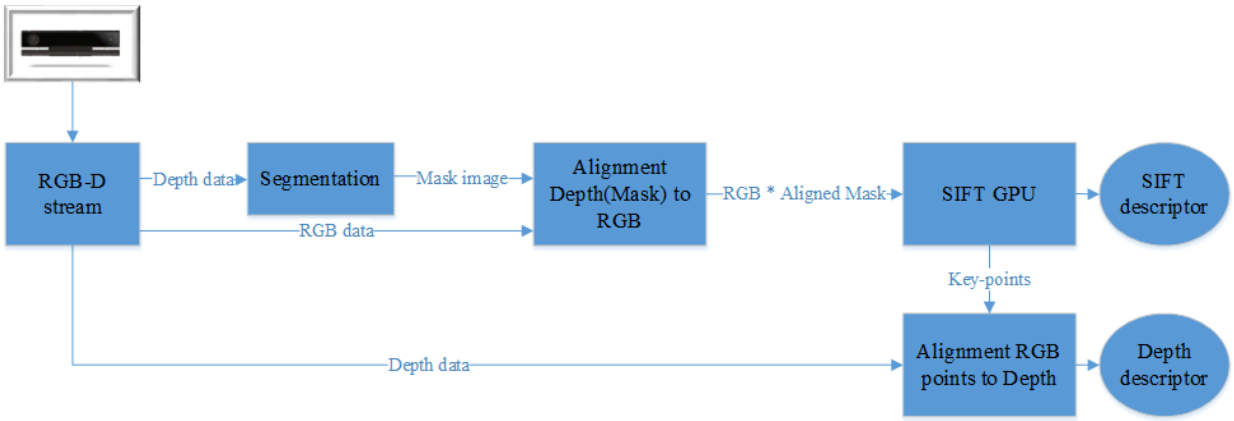


Figure 7. Pipeline of the experimental setup.

6.2 Object segmentation in the D image and geometric alignment of the RGB and D images

The first step in the processing pipeline is the object segmentation. Methods which used the RGB-D sensor and, hence, the depth image for the segmentation of 3D objects can overcome the methods of segmentation using RGB image in lower computational complexity and also in the achieved robustness.

A presumption of the proposed method is the object position in the captured scene, which means that the unknown object is positioned as the nearest object to the Kinect sensor. In our approach we use the method of growing regions, where a seed point of the segmentation is the nearest point to the Kinect device detected in the depth image. The result of the segmentation is a binary mask. The border of the mask can produce misleading key points which are not inside of the object and hence a morphological processing - erosion - of the mask is necessary. The segmentation step is crucial for the recognition as we can easily filter out most of the undesired key points and speed up the matching phase.

Because of the different resolution of the depth sensor and the color camera, we need to geometrically align the both frames RGB and D. Then we can apply the binary segmentation mask on the geometrically aligned RGB image.

Figure 8. Object segmentation based on the depth.

6.3 Key point selection

In the next step, the SIFT detector is used for the calculation of all the key points in the segmented area. As mentioned, our real-time system already uses published implementation of the SIFT detector and SIFT descriptor calculated on the graphical processor unit (GPU).

The information about the SIFT key point position is then used by the depth-based pre-selection step which will be described in chapter 6.4 in more detail. The result of this step is a subset of selected objects which satisfy the depth examination. This subset will then serve as an input for the SIFT brute-force matching recognition.

6.4 Matching

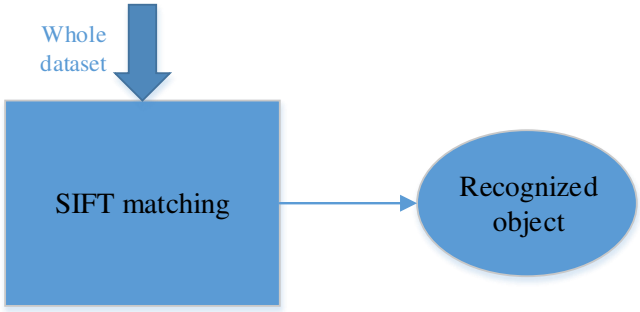


Figure 9. Reference method for matching without any pre-selection.

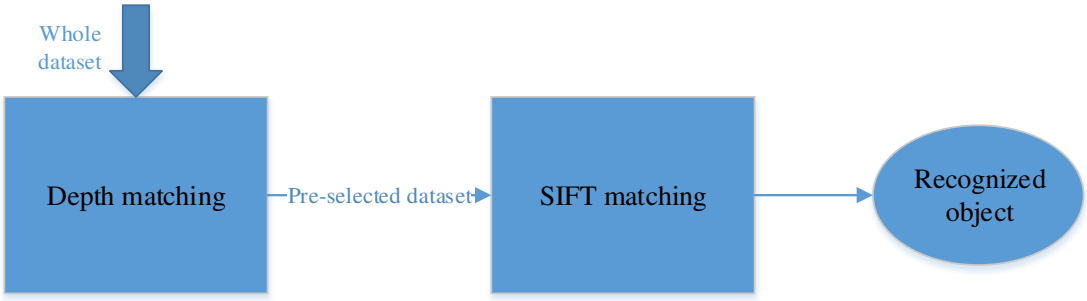


Figure 10. Proposed matching with pre-selection.

For the evaluation of the tested recognition system we have used the standard implementation of the brute-force matching. The Euclidean distance in the SIFT feature for each pair of the features will be calculated and subsequently the two best matches for each query are taken into consideration.

The proposed pre-selection method is compared with matching without any pre-selection which serves as a reference method.

7. RESULTS

7.1 Evaluation of the depth descriptor (DD) based pre-selection

The goal of the experimental setup is to prove the ability of a depth description based pre-selection of objects in a dataset which should be used in the subsequent recognition system. The choice of the recognition system can be free in general, but we use a SIFT brute-force matching in our experimental system and we also exploit synergies in SIFT key point detection.

In the evaluation, we are looking for an answer to the question: how many objects of the whole dataset fulfill the pre-selection step with the expectations that the correct corresponding template is included in the pre-selected subset. The number of objects accepted in the pre-selection step “Depth recognition” in relation to the growing size of the dataset is presented in the next figure:

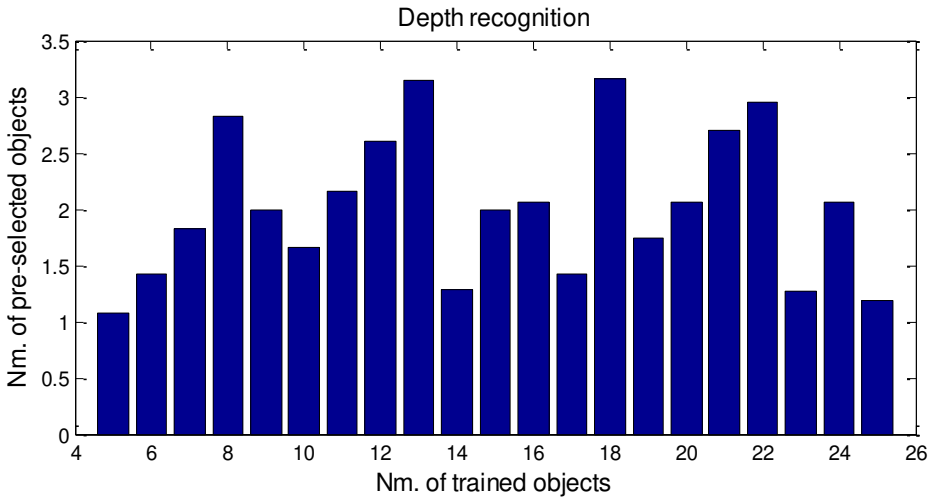


Figure 11. Average number of objects passed depth descriptor matching.

We can see that the depth descriptor is able to remove large numbers of trained objects and hence decrease the time of the color matching. The tested input dataset was growing from 5 to 25 and the number of objects accepted in the pre-selection did not extend the number: 3. The mean value of accepted objects was: 2.03.

7.2 Evaluation of the extended SIFT (SIFT-D)

In the second experiment we tested how an extension of the SIFT descriptor by the depth local description can improve the recognition accuracy. We will address this descriptor as SIFT-D in the following description.

The segmentation part of the processing pipeline and key point detection are similar to those described in the previous section. Then for each of the detected SIFT key points there will be a SIFT and, alternatively, a SIFT-D descriptor calculated.

The recognition has been evaluated in the whole recognition system using SIFT compared with the same recognition system using SIFT-D instead of SIFT.

The matching strategy was proposed by Lowe and is similar to the matching strategy used in our first experiment described in section 3. In this matching strategy the acceptance of a matched pair is given by the relation of the Euclidean distances of two nearest neighbors.

$$M = \{(a_1, b_1) | \frac{d(a, b_1)}{d(a, b_2)} < t, \forall a \in A\} \tag{8}$$

If the relation of distances d(a,b1) and d(a,b2) is < threshold t, the pair (a ,b1) will be accepted. This threshold will be also applied in the described bi-directional matching strategy. The traditional RANSAC algorithm follows this step before the final decision will be achieved.

Whether the template is corresponding to the investigated object or not depends on the ratio of accepted symmetrical correspondences R to the sum of correspondences in one direction A and the opposite direction B.

An object will be recognized if the following condition has been satisfied.

$$c < \frac{2 \cdot |R|}{|A| + |B|} \tag{9}$$

Where c is the final threshold of positive/negative decision. In our evaluation there was c= 0.1.

25 objects (toys) of the dataset were used in the training phase. The recognition system was tested 10 times for each object and each of both descriptors (SIFT and SIFT-D).

The results of this evaluation are presented in the next table.

Table 2. SIFT-D evaluation.

	correct recognized - direct view	correct recognized - variable view
SIFT	99%	94.5%
SIFT-D	100%	97%

8. CONCLUSION AND FUTURE WORK

Two strategies for including the depth information into the local descriptors have been proposed, implemented and tested.

The Depth Descriptor (DD) was used for the reduction of the object from the dataset which will be passed into the next recognition section. The proposed feature is composed of four different depth descriptions in the neighborhood of a key point. Using this descriptor, the number of the pre-selected objects could be significantly reduced.

The SIFT descriptor extended by the local depth information has been also tested and evaluated. The recognition rate was quite high using the SIFT descriptor, but the contribution of the SIFT-D descriptor brought a still better result of successful recognition.

ACKNOWLEDGMENT

This research has been supported by a grant VEGA 1/0625/14.

REFERENCES

[1] Steder, Bastian, Radu Bogdan Rusu, Kurt Konolige, and Wolfram Burgard. "NARF: 3D range image features for object recognition." In Workshop on Defining and Solving Realistic Perception Problems in Personal Robotics at the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), vol. 44. 2010.

[2] Steder, Bastian, Radu Bogdan Rusu, Kurt Konolige, and Wolfram Burgard. “Point Feature Extraction on 3D Range Scans Taking into Account Object Boundaries.” 2011 IEEE International Conference on Robotics and Automation (May 2011). doi:10.1109/icra.2011.5980187.

[3] Tombari, Federico, Samuele Salti, and Luigi Di Stefano. “A Combined Texture-Shape Descriptor for Enhanced 3D Feature Matching.” 2011 18th IEEE International Conference on Image Processing (September 2011). doi:10.1109/icip.2011.6116679.

[4] Tombari, Federico, Samuele Salti, and Luigi Di Stefano. “Unique Signatures of Histograms for Local Surface Description.” Lecture Notes in Computer Science (2010): 356–369. doi:10.1007/978-3-642-15558-1_26.

[5] Nascimento, Erickson R., Gabriel L. Oliveira, Mario F. M. Campos, Antonio W. Vieira, and William Robson Schwartz. “BRAND: A Robust Appearance and Depth Descriptor for RGB-D Images.” 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (October 2012). doi:10.1109/iros.2012.6385693.

[6] Lowe, D.G. “Object Recognition from Local Scale-Invariant Features.” Proceedings of the Seventh IEEE International Conference on Computer Vision (1999). doi:10.1109/iccv.1999.790410.

[7] Changchang, Wu., “SiftGPU: A GPU Implementation of Scale Invariant Feature Transform (SIFT),” < http://cs.unc.edu/~ccwu/siftgpu/>

Attachment F: Paper published at IITSRC 2015 conference

3D Object Recognition Based on Local Descriptors

Marek JAKAB*

Slovak University of Technology in Bratislava
Faculty of Informatics and Information Technologies
Ilkovičova 2, 842 16 Bratislava, Slovakia
marko.jakab@gmail.com

Abstract. In this paper, we propose an enhanced method of 3D object description and recognition based on local descriptors using RGB image and depth information (D) acquired by Kinect v2 sensor. Our main contribution is focused on a novel local depth descriptor (DD) that includes a 3D description of the key point neighborhood. Thus defined the 3D descriptor can then enter the decision-making process. New approach has been proposed, tested and evaluated in this paper that deals with the object recognition system using the original SIFT descriptor in combination with our novel proposed 3D descriptor, where the proposed 3D descriptor is responsible for the pre-selection of the objects. The results show an improvement in speed of the recognition system.

1 Introduction

Visual object recognition is still one of the biggest challenges in computer vision. One of the promising ways to approach this challenge seems to be the usage of local descriptors. Object recognition methods based on local descriptors are potentially applicable for applications working in near real time and the method also provides invariance to different illumination, scale, angle or rotation of the object, even if limited. Our goal is to provide a novel local descriptor which will extend the local description derived from an RGB image with a description of the neighborhood of a key point in the depth image (D).

In this paper, we propose a novel local depth D-descriptor which represents a 3D description of the key point neighborhood. As so defined, the 3D descriptor can then enter into the decision-making process where the proposed 3D descriptor is responsible for the pre-selection of the objects.

Object recognition using local descriptors is typically based on the paired matching of all the trained object data with the data derived from an unknown object. The problem of a growing time requirement and a decreasing recognition accuracy becomes serious with the increasing number of objects in a dataset. We propose the usage of the RGB-D data obtained from the Kinect v2 sensor device for the purpose of a pre-selection of the objects for a subsequent SIFT-matching-based object recognition. That means that only a selected part of the objects from the trained dataset will

* Master degree study programme in field: Applied Informatics
Supervisor: Dr. Vanda Benešová, Institute of Applied Informatics, Faculty of Informatics and Information Technologies STU in Bratislava

IIT.SRC 2015, Bratislava, April 23, 2015, pp. 231–236.

be accepted by the first recognition part which uses the depth descriptors. Hence, the calculation time needed for the following SIFT matching will not exceed certain limits with growing number of objects in the dataset.

2 Related Work

The most relevant papers which present the research related to object recognition concerning both RGB and depth image (RGB-D) have been taken into consideration.

Example of combined texture & shape descriptor is Color-Signature of Histograms of Orientations (CSHOT) [1] [2]. This descriptor contains both shape and texture information suitable for RGB-D (RGB & Depth) matching. The basic idea of forming this descriptor is the eigenvalue decomposition of the scatter matrix around the key point. A spherical grid is created around the point it could form a histogram of normal vectors defined for each sector of the grid.

Binary Robust Appearance and Normals Descriptor (BRAND) [3] is the descriptor which combines the appearance and geometric shape information from RGB-D images. In the first step, the scale factor using the depth information from RGB-D image is calculated. The scale factor is then used in the next step (dominant direction estimation) and in the feature analysis in the key point's vicinity. The authors combine both appearance and geometric information to create key point descriptors that are expected to be robust, fast and lightweight. The authors also demonstrate that the descriptor is robust, invariant to rotation and scale, and provides reliable results even for sparsely textured and poorly illuminated scene. The main constraint of this method is, that a small irregularities of these surfaces can be confused with noise.

Lowe proposed a Scale Invariant Feature Transform (SIFT) [4]. SIFT combines a scale invariant region detector and a descriptor based on the gradient distribution in the detected regions. The main idea of the descriptor is to compose the local histograms of gradient locations and gradient orientations in one vector, wherein the contribution to the location and orientation in histogram bins is weighted by the gradient magnitude. The quantization of gradient locations and orientations makes the descriptor robust to small geometric distortions and small errors in the region detection. The descriptor is finally arranged as a 128-dimensional vector of float numbers. The SIFT descriptor is one of the older descriptors, but still robust, scale and rotation invariant and widely used.

3 Depth Descriptor (DD)

3.1 Descriptor Pattern

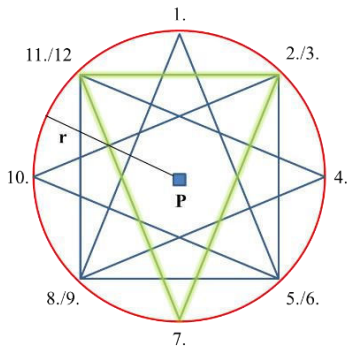


Figure 1. Star pattern around the key point P.

To fill the descriptor vector, we use a star pattern created by four triangles which are rotated around the selected key point. Four triangles determine 12 points and the depth distances value of these 12 points are used in the computing of the depth descriptor vector. Each triangle defines a plane and hence the normal vectors of the planes could be calculated and subsequently used in the descriptor. The points in the star pattern are defined with the 15mm constraint radius.

3.2 Descriptor vector

Four features, which are derived by statistical evaluation, have been taken for the definition of the depth feature vector. Three of them are based on the local surface description and the fourth feature also takes into account the global information of the object.

1st feature value derived from average angle referenced to a normal vector

The first feature in the depth feature vector is derived from the average angle of the normal vectors given by all triangles in the key point. Therefore we need to compute four normal vectors - one for each of the four triangles given by the pattern. Then the corresponding average vector of this key point will be calculated. This vector will be used as a reference in the next evaluation. In the next step, we can compute the difference angle between each of the normal vectors and the reference normal vector.

$$F\_avgAngle = \frac{1}{N}\sum_{i=1}^{N} \arccos\left(\frac{u_{i1}.v_{m1} + u_{i2}.v_{m2} + u_{i3}.v_{m3}}{\sqrt{u_{i1}^{2} + u_{i2}^{2} + u_{i3}^{2}}.\sqrt{v_{m1}^{2} + v_{m2}^{2} + v_{m3}^{2}}} \right) \tag{1}$$

Where N is the number normal vectors/triangles.

$\vec{v}_m$ is the average normal vector.

$\vec{u}_i$ is the normal vector given by triangle.

2nd feature value derived from standard deviation of depth referenced to a plane perpendicular to the averaged normal vector

The second feature was proposed with the goal to create an efficient description of the local depth differences. The descriptor value is then given by the standard deviation of the depth differences between the actual depth value in a point and the reference plane which is computed from average normal vector.

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2} \tag{2}$$

Where N is the number of all points in the descriptor pattern (see section 3.1).

x_i is the distance of the depth value from the reference plane in the point i.

μ is the mean value of x_i (i=1...n).

3rd feature value derived from the difference between the maximum and minimum depth values referenced to a plane perpendicular to the averaged normal vector

Calculation of the depth value referenced to the reference plane is similar to those described in the previous section. Instead of the previous statistical evaluation by standard deviation, the simple difference between the maximum and minimum value of the recalculated depth value was taken into account as the next feature value.

4th feature value derived by using a global angle

The previous three feature values have been designed as a local description without any global reference. The goal of the fourth feature value is to take into consideration also the global surface of the object. The global normal vector is calculated as the angle between pattern normal vector and average vector value out of all local normal vectors across all key points.

$$F\_globalAngle = \arccos\left(\frac{g_1.v_1 + g_2.v_2 + g_3.v_3}{\sqrt{g_1^2 + g_2^2 + g_3^2}.\sqrt{v_1^2 + v_2^2 + v_3^2}}\right) \quad (3)$$

Where $\vec{g}$ is the global average normal vector through all key points.

$\vec{v}$ represents the local average normal vector at a given key point.

3.3 Invariance

The original SIFT descriptor is invariant to different scale, rotation and small perspective transformation of the object. Our goal is to develop a local depth descriptor which will extend the SIFT descriptor and hence we also aim to achieve a comparable rotation and scale invariance.

Scale invariance

The 3D information should be taken into account for the scale definition of the depth descriptor. As we know the distance of the selected key point from the sensor, we can easily determine the corresponding neighborhood size in pixels around the key point according to the real world distance.

Rotation & perspective invariance

To achieve invariance to different rotations of the object, we aim to fill the descriptor with the information which is defined as independent of the rotation and angle of the object. We have already described those values in previous chapters.

4 Experimental setup

4.1 Dataset

Our tested dataset was related to the final application: a self-checkout in a shop. The set of toys fulfill very well the requirements of the application. The set also has a large variability necessary for the testing. Hence, various toys of different size and shape were chosen.

Figure 2. Objects used for training and measurement.

4.2 Hardware and software implementation

In the experiment for the evaluation of the proposed depth-based pre-selection the following hardware was used: Laptop with CPU Intel Core i7, 3632QM, 2.2 GHz, GPU NVidia GeForce GT635M and RAM 8GB DDR3 1600Mhz.

The test program was implemented in C++ with the library OpenCV. Implementation of the SIFT feature extraction and description was computed on the GPU unit [5]. Thanks to the GPU implementation we are able to use full HD images for color description, 512 to 424 pixels for depth image and still provide the results in near real time. The project and the source code are available on the web page: http://vgg.fiit.stuba.sk/3d-object-recognition/

5 Overview of the processing pipeline

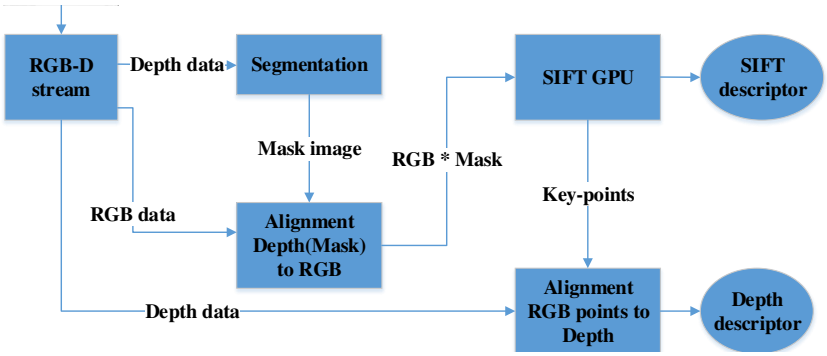


Figure 3. Pipeline of the experimental setup.

5.1 Object segmentation in the D image and geometric alignment of the RGB and D images

The first step in the processing pipeline is the object segmentation. Methods which used the RGB-D sensor and, hence, the depth image for the segmentation of 3D objects can overcome the methods of segmentation using RGB image in lower computational complexity and also in the achieved robustness.

In our approach we use the method of growing regions, where a seed point of the segmentation is the nearest point to the Kinect device. The result of the segmentation: binary mask will be used to retrieve the object texture. The segmentation step is crucial for the recognition as we can easily filter out most of the undesired key points and speed up the matching phase.

Figure 4. Object segmentation based on the depth.

5.2 Key point selection

In the next step, the SIFT detector is used for the calculation of all the key points in the segmented area. The information about the SIFT key point position is then used by the depth-based pre-selection step.

5.3 Matching

For the evaluation of the tested recognition system we have used the standard implementation of the brute-force matching. The Euclidean distance in the SIFT feature for each pair of the features will be calculated and subsequently the two best matches for each query are taken into consideration. The proposed pre-selection method is compared with matching without any pre-selection which serves as a reference method.

6 Evaluation of the depth descriptor (DD) based pre-selection

The goal of the experimental setup is to prove the ability of a depth description based pre-selection of objects in a dataset which should be used in the subsequent recognition system. The choice of the recognition system can be free in general, but we use a SIFT brute-force matching in our experimental system. We can see that the depth descriptor is able to remove large numbers of trained objects and hence decrease the time of the texture matching. Next figure show us the number of objects which passed through the pre-selection phase across several experiments.

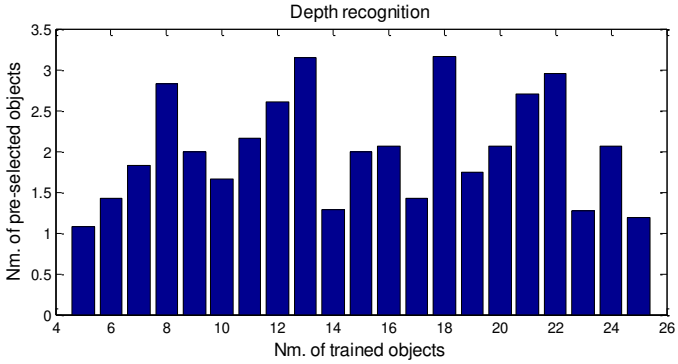


Figure 5. Average number of objects passed depth descriptor matching.

Over several experiments, the matching speed improved by 10 – 20 % when the object was compared against the dataset of 35 objects (at the mentioned setup: ~100ms per object with SIFT descriptor and ~80ms per object when Depth + SIFT was used). While the results improve with increasing dataset, we also noticed slightly negative speed effect for the first ~15 learned objects.

7 Conclusion

The Depth Descriptor (DD) was used for the reduction of the object from the dataset which will be passed into the next recognition section. The proposed feature is composed of four different depth descriptions in the neighborhood of a key point. Using this descriptor, the number of the pre-selected objects could be significantly reduced together with recognition time.

Acknowledgement: This research has been supported by a grant VEGA 1/0625/14.

References

[1] Tombari, Federico; Salti, Samuele; Di Stefano, Luigi.: A combined texture-shape descriptor for enhanced 3D feature matching. In: *ICIP 2011,* IEEE, 2011. p. 809-812.

[2] Tombari, Federico; Salti, Samuele; Di Stefano, Luigi.: Unique signatures of histograms for local surface description. In: *Computer Vision–ECCV 2010*. Springer, 2010. p. 356-369.

[3] Nascimento, Erickson R., et al.: BRAND: A robust appearance and depth descriptor for RGB-D images. In: *Intelligent Robots and Systems (IROS 2012)*, 2012. p. 1720-1726.

[4] Lowe, David G.: Object recognition from local scale-invariant features. In: Computer vision, 1999. *The proc. of the seventh IEEE international conference on. Ieee*, 1999. p. 1150-1157.

[5] Changchang, Wu., "SiftGPU: A GPU Implementation of Scale Invariant Feature Transform (SIFT)" Available at: http://cs.unc.edu/~ccwu/siftgpu/