USING RANGE INFORMATION TO REDUCE LOCAL DESCRIPTOR COMPUTATIONS

Stefán Freyr Stefánsson
Master of Science
Computer Science
January 2011
School of Computer Science
Reykjavík University

M.Sc. RESEARCH THESIS

ISSN 1670-8539

Using Range Information to Reduce Local Descriptor Computations

by

Stefán Freyr Stefánsson

Research thesis submitted to the School of Computer Science
at Reykjavík University in partial fulfillment of
the requirements for the degree of
Master of Science in **Computer Science**

January 2011

Research Thesis Committee:

Björn Þór Jónsson, Supervisor
Associate Professor, Reykjavik University, Iceland

Kristinn R. Thórisson
Associate Professor, Reykjavik University, Iceland

Laurent Amsaleg
Research Scientist, IRISA/CNRS, France

Using Range Information to Reduce Local Descriptor Computations

Stefán Freyr Stefánsson

January 2011

Abstract

Object recognition is of particular interest for mobile robot vision systems. Local image descriptor methods are widely used for this purpose. However, hardware limitations, coupled with the amount of processing power needed to generate these descriptors, limit their usefulness. Meanwhile, range imaging has been rapidly becoming more affordable and easy to deploy on mobile platforms with the introduction of mass-market time-of-flight range cameras. In this thesis, we present a case study of a complete vision system employing such a range camera along with a regular RGB camera. In the first part of the thesis we describe and evaluate the architecture and performance of this system, which is based on YARP, an open source, cross-platform toolkit that enables a clean decoupling of modules. In the second part of the thesis, we propose two methods for improving the performance of local descriptor computation using range information. First, segmenting the RGB image based on distance reduces the number of descriptors produced. Second, using the range information reduces computation spent on enforcing scale-invariance. Our findings indicate that the proposed techniques can result in significant performance improvements with minimal compromising of result quality.

Notkun fjarlægðarupplýsinga til að flýta útreikningum á staðbundnum lýsipunktum

Stefán Freyr Stefánsson

Janúar 2011

Útdráttur

Í tölvusjónarkerfum fyrir kvika róbóta er mikilvægt að greina og þekkja hluti. Staðværir lýsipunktar eru oft notaðir í þessum tilgangi. Hins vegar er nýting þeirra bundin við takmarkanir á vélbúnaði og hversu mikið reikniafl þarf til að búa þá til. Fjarlægðarmyndavélar hafa lækkað ört í verði og minnkað að stærð og því er orðið fýsilegt að nota slíkar vélar á kvikum róbótum. Í þessari ritgerð kynnum við rannsóknir okkar á heildstæðu tölvusjónarkerfi sem notar slíka fjarlægðarmyndavél ásamt venjulegri litmyndavél. Í fyrri hluta ritgerðarinnar leggjum við mat á hönnun og útfærslu kerfisins, en það er byggt á YARP sem er samsafn opins hugbúnaðar sem keyrir á mörgum stýrikerfum og einfaldar aðskilnað eininga kerfisins. Í seinni hluta ritgerðarinnar leggjum við til tvær aðferðir til að nýta fjarlægðarupplýsingar til að hraða gerð staðværra lýsipunkta. Sú fyrri er að hluta niður litmyndina samkvæmt upplýsingum úr fjarlægðarmyndinni og fækka þannig lýsipunktum. Hin síðari nýtir fjarlægðarupplýsingarnar til að minnka vinnsluna sem fer í að tryggja þol gagnvart stærðarbreytingum. Niðurstöður okkar gefa til kynna að með þessum aðferðum sé hægt að ná fram talsverðri aukningu á hraða með lágmarks fórn á gæðum niðurstaðna.

To my parents, for their support and encouragement throughout the years.

Acknowledgements

It's been a long journey, but I've made it. I would like to acknowledge the people who helped me get here.

My supervisor, Dr. Björn Þór Jónsson, for his endless patience, support, and constructive criticism.

My committee, Dr. Kristinn R. Thórisson and Dr. Laurent Amsaleg, as well as Dr. Patrick Gros, for their valuable input and ideas.

My good friend, Arnar Birgisson, for all his invaluable help and advice.

The YARP team (especially Paul Fitzpatrick, Giorgio Metta, Lorenzo Natale) and Rob Hess for their excellent software and, particularly, for making it open-source for me to play with.

And last but certainly not least, my family. Without their support and encouragement I never would have gotten here in the first place.

Publications

Part I of this thesis is adapted from technical report (Stefánsson et al., 2008). A part of that technical report was also published in Proceedings of the Fourth International Conference on Computer Vision Theory and Applications (VISAPP) (Stefánsson et al., 2009). While Björn Þór Jónsson and Kristinn R. Thórisson contributed significantly to the writing of the technical report and conference paper, the implementation and experimentation in Part I is entirely my own work, as is the material in Part II.

Contents

List of Figures **xiii**

1 Introduction **1**
1.1 Object Recognition and Local Image Descriptors 1
1.2 Range Information 2
1.3 Overview and Contributions 3

Part I: Architectural Framework **5**

2 Architectural Requirements **7**
2.1 Sensory Requirements 7
2.2 Hardware Requirements 8
2.3 Processing Requirements 9
2.4 Communication Requirements 9

3 Communication Infrastructures **11**
3.1 YARP 11
3.2 Alternative Architectures 12

4 Experimental Environment **15**
4.1 Hardware 15
4.2 Video Stream Producer 15
4.3 Image Processing Components 15

5 Experimental Evaluation **17**
5.1 Experiment 1: Transport Mechanisms 18
5.2 Experiment 2: Parallel Pipelines 20
5.3 Experiment 3: YARP Overhead 22

6 Summary **25**

Part II: Range Enhanced SIFT Processing **27**

7 Background **29**
7.1 Local Image Descriptors 29
7.2 Scale-Invariant Feature Transform (SIFT) 30
7.3 Database Storage and Retrieval of Descriptors 33
7.4 Range Imaging 34

8 Methods **37**
8.1 The Processing Pipeline 37
8.2 Image Segmentation 40
8.3 Scale-Space Reduction 41
8.4 Summary 43

9 Experimental Setup and Hypotheses **45**
9.1 Hardware and Software 45
9.2 Object Collection and Workload 46
9.3 Compensating for Collection Size 47
9.4 Metrics 50

10 Results **51**
10.1 Performance 51
10.2 Quality 53
10.3 Conclusion and Summary 56

11 Conclusions **57**

List of Figures

2.1 Architectural Requirements. 8

5.1 The basic pipeline setup. 17
5.2 Exp. 1: Frame rate at receiver. 18
5.3 Exp. 1: Frame drops in pipeline. 19
5.4 Exp. 1: Latency of received frames. 19
5.5 The parallel pipelines setup. 20
5.6 Exp. 2: Frame drops per receiver. 21
5.7 Exp. 2: Latency of received frames. 21
5.8 Exp. 3: Latency of YARP and stand-alone process. 23

7.1 Difference-of-Gaussians pyramid. 32
7.2 Extrema detection in a DOG scale. 33
7.3 Example of a color/range image pair. 35

8.1 The processing pipeline. 38
8.2 Illustration of range partitioning. 42
8.3 Illustration of the rescaling principle. 43

9.1 Cameras and experiment setup. 47
9.2 Movie posters used for the experiments. 48
9.3 Comparison of descriptor distance and scale difference (query: 2m; DB: 2m). 49

10.1 Performance (FPS) of Sifter with database distance fixed at 4 meters. . . . 52
10.2 Performance (FPS) of Sifter with database distance fixed at 2 meters. . . . 53
10.3 Quality of Searcher with database distance fixed at 4 meters. 54
10.4 Comparison of descriptor distance and scale difference (query: 3m; DB: 4m). 55
10.5 Quality of Searcher with database distance fixed at 2 meters. 56

Chapter 1

Introduction

One of the most important sensory mechanisms for mobile robots is a sense of vision that robustly supports movement and manipulations in a three-dimensional world. Here, we use the term "vision" broadly to encompass any visuospatial sensory inputs and processing required for an understanding of the environment. Accumulated experience has shown, however, that for such robotic vision it is necessary to employ a number of sensors and processing mechanisms, integrated in various ways—often dynamically—to support real-time action in various contexts. Such systems are often constrained by heavy processing requirements but limited resources, because on-board computers are typically chosen for their power consumption characteristics, and may yet need to handle a large amount of data in real-time for immediate reactive processing.

In this thesis, we focus on such a vision system, which employs a number of techniques, including (a) color video cameras, which provide shape and color information but do not easily give any distance information, (b) time-of-flight cameras, which can yield sufficient range information to create a depth map of the environment, (c) image descriptions, such as local image descriptors, which can be used for object recognition and obstacle detection, and (d) a communications infrastructure.

1.1 Object Recognition and Local Image Descriptors

Robot vision systems commonly use advanced image processing techniques such as object recognition. Object recognition is an important sub-domain of computer vision and serves as a foundation for many higher level solutions. One of the primary algorithms used for object recognition is SIFT (Lowe, 2004). The SIFT algorithm produces a set

of descriptors for a given input image. These descriptors are called local descriptors, referring to the fact that each of them contains a detailed description of a single point of interest in the processed image, and furthermore, that all descriptors are independent from one another. A major benefit of this approach for object recognition is that it is possible to keep a reference database of recognizable objects and, with a high degree of confidence, locate these objects in a scene despite partial occlusion, rotation or changes in illumination.

A downside to using SIFT for object recognition is the processing time required for generating the descriptors. Considering the robustness of the descriptors, the SIFT algorithm could be considered quite fast, but nevertheless it is not suitable for real-time applications. Some variations have been proposed to SIFT to increase processing speed, such as PCA-SIFT (Ke and Sukthankar, 2004) and GLOH (Mikolajczyk and Schmid, 2005) but these improvements still fall short for real-time applications. Processing speed has also been improved by implementing the SIFT algorithm using the GPU; while this provides significant speed improvements it requires specialized hardware which is typically energy consuming.

With most of these improvements to the algorithm itself, the focus has primarily been on reducing the dimensionality of the descriptors or otherwise reducing the computation time per descriptor. Intelligently reducing the number of descriptors that are generated is a viable option for reducing the overall computation time but this has received relatively little attention thus far.

1.2 Range Information

Recent hardware advances have resulted in the availability of range cameras for accurate distance measurements. A question then arises whether such distance information can be used to speed up processing for methods such as SIFT.

By segmenting the input image in such a way that segments contain only interesting regions of the image, such as foreground objects, the number of descriptors that are generated may be reduced, and thereby the total processing effort needed for both generation and matching. Such segmentation can be implemented by using range information to detect objects that protrude from their surroundings.

Another way to utilize range information is by reducing the effort spent on ensuring scale invariance for the descriptors. If the collection of objects to be recognized is created using a reference distance, it is possible to compensate for any scale differences between

the segment and the reference database, using the range data associated with each image segment.

1.3 Overview and Contributions

This thesis presents a case study of a system to facilitate efficient processing of SIFT descriptors for object recognition by utilizing distance information from a range camera. The thesis makes the following two major contributions.

The first part of the thesis describes and evaluates our computer vision architecture. This architecture is based on YARP, which is an open source, cross-platform collection of APIs and utilities that enable a clean decoupling of modules and abstracts the communication between them. We demonstrate that the resulting infrastructure is flexible, yet allows for efficient communications between processes and/or computers. This architecture is then used in the remainder of the thesis.

In the second part of the thesis, we explore the possibility of improving the performance of object recognition through two techniques. First, we segment the image using range information, thereby reducing the number of SIFT descriptors produced. Second, we utilize the range information to reduce computation spent on scale-invariance in the SIFT algorithm. The potential efficiency of the resulting system is then demonstrated through an experimental study.

Part I: Architectural Framework

In order to study the use and interactions of complex vision systems it is clearly necessary to use an architectural framework supporting flexible manipulation of such compound, multimodal data on diverse hardware platforms. Such a framework must allow for easy runtime configuration of the processing pipeline, while incurring limited overhead. Low-level options, such as shared memory and/or remote procedure calls, are not flexible enough, as they must be augmented with mechanisms for handling variable latency, priorities or other necessary features of complex architectures and soft-realtime response generation. What is needed is a higher-level framework that supports free selection of communication methods, including shared memory and TCP/IP, depending on the data and architectural constraints at any point in time.

Several frameworks exist which partially address our needs, but very few address all of them. One of these frameworks is YARP (Yet Another Robot Platform), which is a set of libraries to cleanly decouple devices from software architecture (Metta et al., 2006; Fitzpatrick et al., 2008). It is an attempt to provide a foundation that makes robot software more stable and long-lasting, while allowing for frequent changes of sensors, actuators, processors and networks. As the authors themselves say: "YARP is written by and for researchers in robotics, particularly humanoid robotics, who find themselves with a complicated pile of hardware to control with an equally complicated pile of software" (Metta et al., 2006, p. 1).

In this part we report on an effort to evaluate the use of YARP for architectures that capture and manipulate data from a standard video stream. We explore how well the platform supports our basic needs, such as for sequential processing in a pipeline architecture, how easy the platform is to use and how well it performs.

The remainder of this part is organized as follows. We outline our requirements in Chapter 2. Then we describe YARP and other communication infrastructures in Chapter 3. We report on our architecture in Chapter 4 and analyze its performance in Chapter 5, before summarizing the results in Chapter 6.

Chapter 2

Architectural Requirements

Any computer vision architecture has a set of requirements that must be satisfied. The main requirements for our project are shared with many other mobile robot projects and put a strong emphasis on multimodal integration. The requirements are illustrated in Figure 2.1, which depicts four major categories of requirements: sensory aparati, actuators and other hardware, software, and communication. We now describe each of these.

2.1 Sensory Requirements

As Figure 2.1 shows, a robot sensory and vision system must handle input from a variety of sensors, such as:

- Video cameras, which capture 2-D image streams.
- Range sensors, such as time-of-flight cameras. Their output can be used to build a map of the area, be combined with data from other range sensors such as laser range finders, and/or be combined with the video stream to yield image+depth information.
- Proximity sensors can augment a standard computer vision system, for example through collision detection, as range cameras typically have an upper and lower limit on their range.
- Position sensors for head motion provide information about the direction that cameras and other sensors point in; the robot's head has cameras and a directional microphone, which, when combined with depth information, could be used to determine the source of environmental sounds, e.g. human speech.

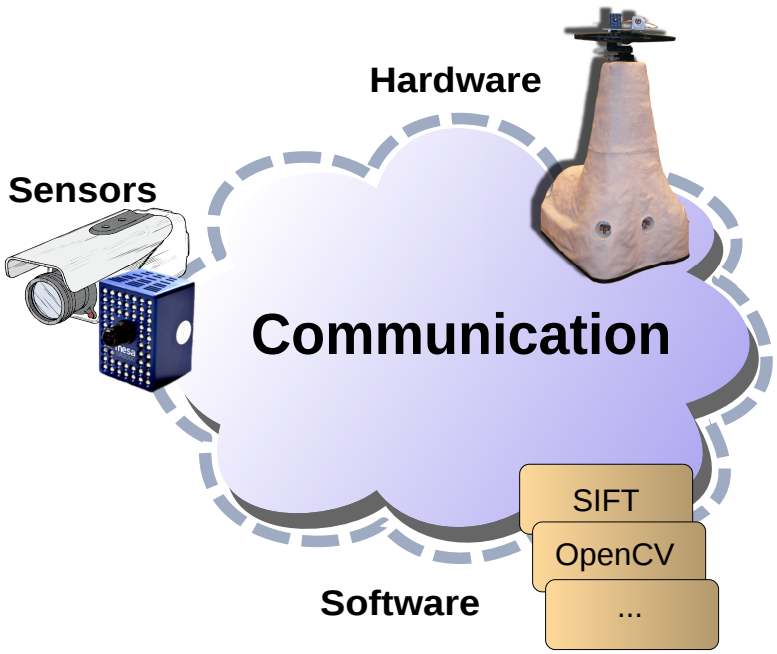


Figure 2.1: Architectural Requirements.

Furthermore, advanced processing methods can be used to build “super sensors”, such as head-motion sensors, through a combination of image analysis, face detection, outline detection, and other means. This is discussed further in the section on processing requirements.

Finally, most sensors require or allow some parameter settings and it is imperative that the infrastructure allows easy runtime access to those.

2.2 Hardware Requirements

Mobile robots typically have at least one on-board computer. The processing power of on-board computers, however, tends to be less than the average desktop machine for reasons of energy efficiency. Such low-power, slow computers can nonetheless be used for important real-time analysis such as collision avoidance and other critical tasks. Performing more significant processing on a mobile robot thus requires off-robot processing, e.g. on computing clusters, necessitating network communication with the robot. Ideally the developer should not need to be much concerned with whether processes are located locally or remotely; changing the configuration should be as seamless as possible. In a similar manner, the sensor/processing architecture must gracefully handle access to sensors that necessitate dedicated hardware and/or software.

2.3 Processing Requirements

We divide processing requirements roughly into three categories, based on complexity:

- Routine signal processing and pre-processing tasks. Good examples are standard routines included in OpenCV and other image processing libraries, including image resizing, sharpening, blurring, brightness adjustment, and so on.
- Ad-hoc image processing tasks, including multimodal fusion such as the merger of information from a multitude of sensors.
- Advanced image description tasks such as database searches, semantic analysis, and/or tagging and the like.

An example of an ad-hoc image processing task is the segmentation of a color image based on distance information. As the depth camera can not be located at exactly the same spot as the color image camera, and is quite unlikely to have exactly the same focal length, the two images will always differ in some respects, especially for close objects. Determining how to project the distance information onto the image is a typical complex and ad-hoc image processing task (Lindner and Kolb, 2007).

As an example of an advanced image description task, the SIFT image description scheme supports well the recognition of objects (Lowe, 2004). In this scheme, each image is described through a set of high-dimensional vectors of numbers; an object is recognized through the matching of SIFT descriptors from a current image to descriptors from pre-described images in a reference collection. The architectural framework should be able to handle the conversion of the image to such an image description, communicate with a search engine, and deliver lists of matching objects.

2.4 Communication Requirements

The communication infrastructure is a key component of any vision or multimodal sensory system. To satisfy the requirements above, it must:

- Transparently allow any hardware to work together, by abstracting the communication protocol from the processing tasks. It must at least support shared memory and TCP/IP transport (other protocols may be useful as well).

- Provide support for processing tasks of any complexity, e.g. by allowing processes to communicate any data structures between themselves, whether locally or remotely over networks. For example, it must be possible to augment video streams with additional information, and it must be easy to publish such information even though it may be represented by non-standard data types, such as image descriptor streams or object lists. In many cases, control and tagging messages must also be transmitted.
- It must be easy to use for the developer. The chosen infrastructure should enable the developer to focus on the sensory/perception system exclusively and not draw away their attention to communication issues.
- It must strive to minimize overhead. Although some overhead is acceptable—and indeed unavoidable as a trade-off for all the above benefits—any significant overhead will eventually lead to the abandonment of the communication infrastructure in favor of hard-coded, specialized solutions.

Chapter 3

Communication Infrastructures

There are several potential candidates that can be chosen as underlying communication infrastructure for video data, including YARP (Metta et al., 2006; Fitzpatrick et al., 2008), OpenAIR (Thórisson et al., 2007), CAVIAR (List et al., 2005), Psyclone (Thórisson et al., 2005), Robot Operating System (ROS) (Quigley et al., 2009), and others. In the remainder of this chapter we first give a short description of YARP, and our reasons for evaluating it, and then briefly describe some of the alternatives.

3.1 YARP

YARP (Yet Another Robot Platform) is a set of libraries to decouple devices, processing, and communication. YARP provides loose coupling between sensors, processors, and actuators, thus supporting incremental architecture evolution. The processes implemented on top of YARP often lie relatively close to hardware devices; YARP does therefore not "take control" of the infrastructure but rather provides a set of simple abstractions for creating data paths.

A key concept in YARP is that of a communications "port". Processes can have zero, one or more input ports, and produce output on zero, one or more output ports. Ports are also not restricted to a single producer or receiver—many producers can feed a single port, and many receivers can read from a single port. To keep track of ports, YARP requires a special registry server running on the network. The data communicated over the ports may consist of arbitrary data structures, as long as the producer and receiver agree on the format. YARP provides some facility to translate common data types between hardware architectures and such translation can be easily implemented in user defined data

types as well. Each port may be communicated via a host of transport mechanisms, including shared memory, TCP/IP and network multicasting. YARP is thus a fairly flexible communication protocol that leaves the programmer in control.

Our main reason for for evaluating YARP is the fact that it is unobtrusive and basic. Other reasons include the following:

- YARP abstracts the transport mechanism from the software components, allowing any software component to run on any machine. It supports shared memory for local communication, and TCP/IP, UDP, and multicast for communication over a network.
- YARP interacts well with C/C++ code, which is required for our use of the SR-3000 time-of-flight camera. YARP can be used with several other programming languages too.
- YARP can communicate any data structure as long as both receiver and sender agree on the format. Furthermore, it provides good built-in support for various image processing tasks and the OpenCV library.
- It is open-source software. As we wish to make our framework freely available, the communication infrastructure must also be freely available (indeed, we have already sent in a few patches for YARP).
- Finally, although this was by no means obvious from any documentation, the support given by YARP developers has been very responsive and helpful.

These requirements are undoubtedly also met by alternative frameworks and libraries; we have not yet made any formal attempt to compare YARP to these other potential approaches. As described in Chapter 6, the overall experience of using YARP has been good. With a host of tradeoffs the choice of low-level or mid-level middleware/libraries can be quite complex. We leave it for future work to compare YARP in more detail to the approaches described next.

3.2 Alternative Architectures

OpenAIR is "a routing and communication protocol based on a publish-subscribe architecture" (Thórisson et al., 2007). It is intended to help AI researchers develop large architectures and share code more effectively. Unlike YARP, it is based around a blackboard information exchange and optimized for publish-subscribe scenarios. It has thoroughly

defined message semantics and has been used in several projects, including agent-based simulations (Thórisson et al., 2005) and robotics (Ng-Thow-Hing et al., 2007). OpenAIR has been implemented for C++, Java and C#.

CAVIAR (Tweed et al., 2005) is a system based on one global controller and a number of modules for information processing, especially geared for computer vision, providing mechanisms for self-describing module parameters, inputs and outputs, going well beyond the standard services provided by YARP and OpenAIR. The implementation contains a base module with common functionalities (interface to controller and parameter management).

Psyclone (see www.cmlabs.com) is an AI "operating system" that incorporates the OpenAIR specification. It is quite a bit higher-level than both OpenAIR and YARP and provides a number of services for distributed process management and development. Psyclone was compared to CAVIAR by List et al. (List et al., 2005) as a platform for computer vision. Like CAVIAR, Psyclone has mechanisms for self-describing semantics of modules and message passing. Unlike CAVIAR, however, Psyclone does not need to pre-compute the dataflow beforehand but rather manages it dynamically at runtime, optimizing based on priorities of messages and modules. Both CAVIAR and Psyclone are overkill for the relatively basic architecture we intend to accomplish at present, at least in the short term, but it is possible that with greater expansion and more architectural complexity, platforms such as Psyclone would become relevant, perhaps even necessary.

Robot Operating System (ROS) (Quigley et al., 2009) is a relative newcomer to the field but is rapidly gaining popularity. As described in http://www.ros.org/wiki: "ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. There are numerous similarities in how ROS and YARP handle process communication, although in ROS messages are defined in a more abstract way to support language neutrality while YARP allows lower level control of the message types being sent. In addition, at the time of starting work for this thesis, ROS was in its infant state while YARP was mature and had an active development community behind it.

Compared to CAVIAR, Psyclone and OpenAIR, YARP looks like a fairly standard library—neither does it do its own message scheduling nor does it provide heavy-handed semantics for message definitions or networking. It incurs minimal semantic overhead to the image processing pipeline and offers an almost transparent way of changing the underlying transport mechanism with which image streams are communicated. This combination of

simplicity and flexibility, as well as its maturity, make YARP seem the most attractive option for our purpose.

It should be noted that all of the above toolkits provide significant benefits to any robotics software system such as ours. Choosing one over the other will always be heavily dependent on the context and characteristics of each project.

Chapter 4

Experimental Environment

We have constructed a preliminary vision system using YARP as the communication infrastructure. We now describe the hardware, video stream producer, and image processing components implemented in our setup.

4.1 Hardware

Our experimental setup runs on a 2.6 GHz Pentium 4 Dell OptiPlex GX270 computer with 1.2 Gb RAM. It is equipped with an NVIDIA GeForce 6600 GT 3D accelerated graphics card. No processing is done on the GPU in our case, although YARP does provide modules and libraries to aid with such tasks.

4.2 Video Stream Producer

We use a simple application to produce a 320x240 pixel video stream. Since the processing in our experiments does not depend on the video stream content, the producer simply outputs a static image repeatedly at a given frame rate. This minimizes the processing spent on producing the video stream.

4.3 Image Processing Components

While YARP handles module communication, we use OpenCV (see http://www.opencv.org) for most image and signal processing. OpenCV is an open source computer vision and

machine learning software library. It provides numerous algorithms for image manipulation, everything from basic convolution algorithms to advanced high-level functionality such as face recognition. In our experiments, we use a trivial blur operation to represent a processing module in the YARP pipeline. This is described in more detail in Chapter 5.

Chapter 5

Experimental Evaluation

In this chapter, we report on an initial performance study of the YARP transport mechanisms, focusing on single processor configurations. At present, the goal is not to study the scalability of the system, but rather to compare some configuration choices of YARP for vision.

To that end, we set up a basic processing pipeline, shown in Figure 5.1. The pipeline consists of 1) a producer, which produces 320x240 pixel image frames at a given frame rate; 2) a number of blur operators, which run the "simple" OpenCV blur algorithm over the frames; and 3) a receiver, which receives the frames. We change the processing pipeline length, or the number of blur operators, to study the effects of overloading the computer.

Each frame is augmented by sequence numbers and time stamps by each of these components, which are used to measure dropped frames and latency, respectively. Other metrics collected include the frame rate observed by the receiver (lower frame rate occurs when frames are dropped) and CPU load.

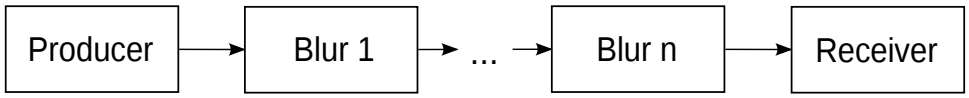


Figure 5.1: The basic pipeline setup.

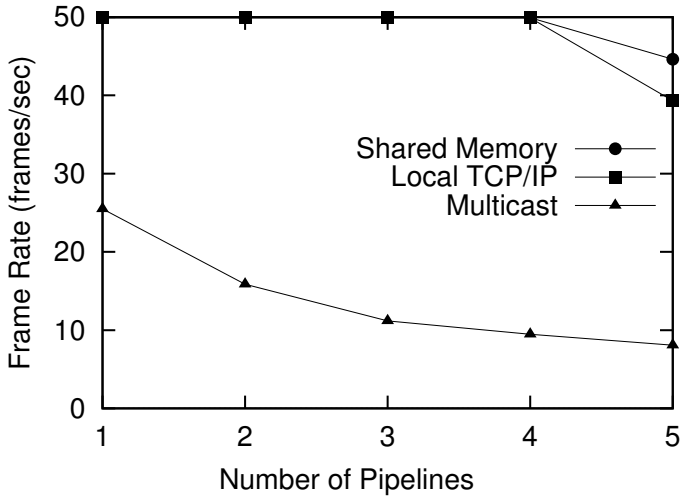


Figure 5.2: Exp. 1: Frame rate at receiver.

5.1 Experiment 1: Transport Mechanisms

In this experiment, the frame rate of the producer was set to 50 frames per second, which is similar to a high-quality video stream. The length of the processing pipeline was varied from one to five consecutive blur operators. We ran measurements using shared memory, local TCP/IP and network multicast connections, with the expectation that shared memory should be fastest. For each configuration, the experiment was run until the receiver had received 50,000 frames.

Figure 5.2 shows the frame rate observed by the receiver. The x-axis shows the length of the processing pipeline. Overall, two effects are visible in the figure. First, using local TCP/IP and shared memory maintains a frame rate of 50 frames per second, until the pipeline consists of four or more blur processes. At that point, the processor is overloaded and frames are dropped as a result, leading to lower frame rates observed by the receiver. Shared memory performs slightly better due to lower communication overhead.[1]

Second, turning to the performance of multicast, Figure 5.2 shows that the processing pipeline achieves a much lower frame rate, ranging from 25 to 8 frames per second. The reason for the lower frame rate is clearly visible in Figure 5.3, which shows the number of frames that are dropped for each configuration. As Figure 5.3 shows, even with only one blur operator, every other frame is dropped with the multicast transport mechanism. The reason for this remains unclear but as a result, the frame rate observed by the receiver is only half the frame rate of the producer. As more blur operators are added, more frames are dropped, explaining the lower frame rates seen in Figure 5.2.

[1] Our early experiments demonstrated a problem in the shared memory transport implementation, which has subsequently been fixed by the YARP developers.

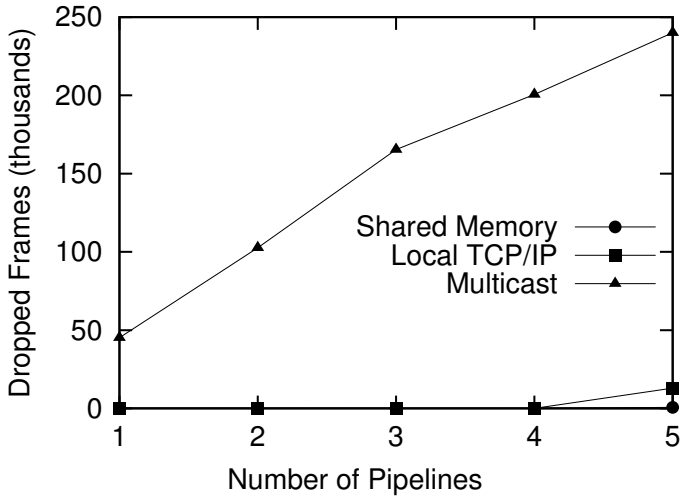


Figure 5.3: Exp. 1: Frame drops in pipeline.

Turning to latency, Figure 5.4 shows that, as expected, latency of the multicast transport mechanism is very high and constantly increasing with pipeline length as frames can be dropped anywhere in the pipeline. For the other two transport mechanisms, latency is relatively low until the pipeline consists of five blur operators. At that point, the CPU is saturated and scheduling conflicts occur. Again, latency is significantly lower using shared memory than TCP/IP due to the lower communication overhead.

We conclude that the multicast transport mechanism is not suitable for local processing, and that the shared memory transport is slightly more efficient than the TCP/IP transport.

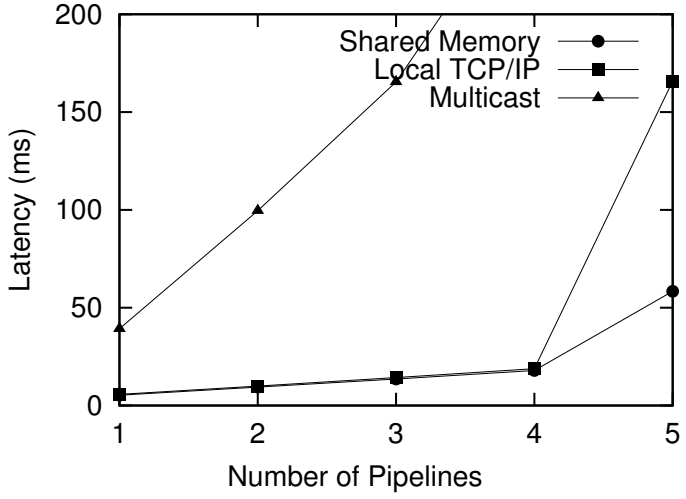


Figure 5.4: Exp. 1: Latency of received frames.

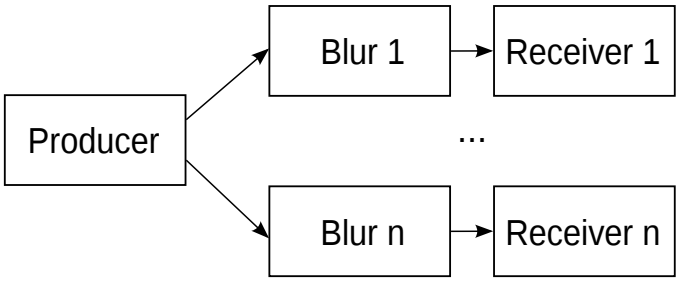


Figure 5.5: The parallel pipelines setup.

5.2 Experiment 2: Parallel Pipelines

The previous experiment showed that while the shared memory and TCP/IP transport mechanisms have similar performance, the multicast mechanism performs much worse. Since the pipeline was linear, however, the experiment did not exercise the potential benefit of the multicast mechanism. To achieve this we set up an experiment with multiple parallel processing pipelines each consisting of a chain of a single blur operator and a receiver which logs the same information as in our previous experiment.

A single producer still provides a stream of images at 50 frames per second. This stream then gets published to all the independent pipelines, using one of the three transport mechanisms (we use the shared memory transport mechanism between each blur operator and the corresponding receiver). In this experiment we thus increase the number of pipelines that the producer sends the video stream to as opposed to increasing the number of blur operators within a linear pipeline. Figure 5.5 shows this setup.

Figure 5.6 shows the number of frames dropped by each of the transport mechanisms. As before, the x-axis shows the number of blur operators in the configuration, but in contrast to the previous experiment each blur operator is now part of a separate pipeline. The figure shows that while dealing with one or two blur operators, frame drops are virtually non-existent for all of the transport mechanisms. When the third blur operator is added, however, the shared memory and TCP/IP transport mechanisms still have negligible frame drops, while the multicast transport mechanism suddenly starts dropping about half of the frames that are produced. Close examination of the log files revealed that roughly every other frame that is produced gets dropped before it reaches any of the blur operators. The reason is that as before, the cost of the multicast transport mechanism means that full CPU utilization can not be achieved. Since frames are being broadcast they either reach all blur operators or none.

The shared memory and TCP/IP transport mechanisms start experiencing frame drops with four concurrent blur operators and the drop rate increases slightly more than twofold

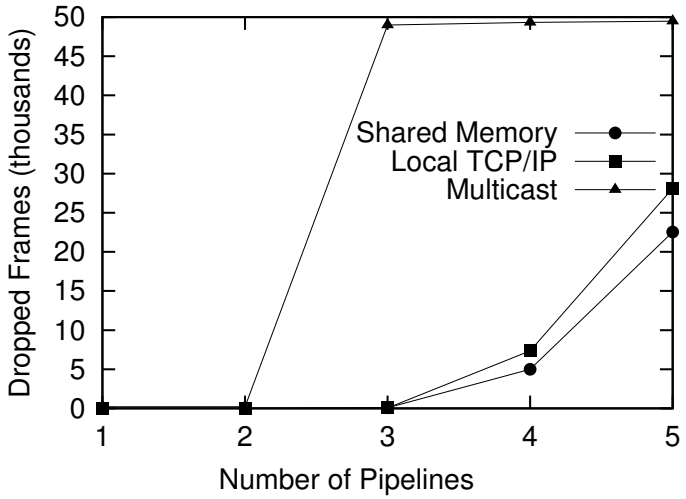


Figure 5.6: Exp. 2: Frame drops per receiver.

once the fifth blur operator is added. These frame drops are explained by the fact that once four blur operators are started, the CPU is fully utilized and processing each frame takes too long for the operators to be able to keep up with the frame rate of the producer.

Figure 5.7 shows the latency of frames as the number of blur operators increases. As the figure shows, the latency increases steadily for the multicast transport mechanism, while the latency for shared memory and TCP/IP jumps once there are four blur operators. The different behavior is due to the different ways that frames get dropped depending on the transport mechanism used.

With the multicast transport mechanism, every other frame is not being received by the blur operators and so no processing is wasted on them. The blur operators therefore have

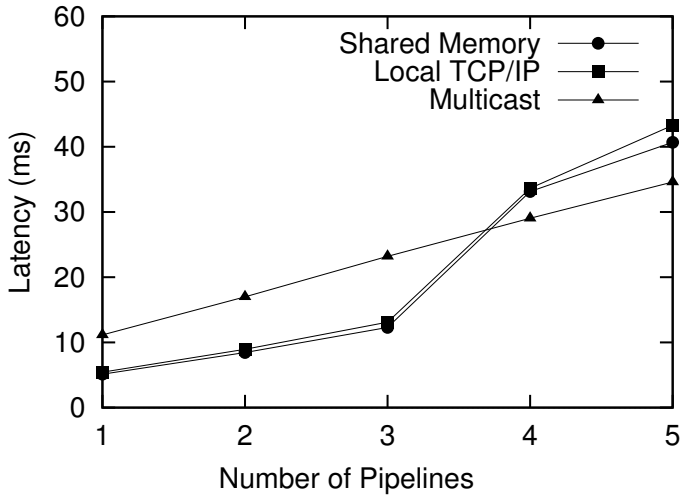


Figure 5.7: Exp. 2: Latency of received frames.

enough CPU power to keep up with the frames that they receive and the only increase in latency is because each frame is being processed at nearly the same time by all the parallel blur operators. This results in longer processing times and increased latency due to saturation of the CPU.

The shared memory and TCP/IP transports display the same gradual increase for up to three concurrent blur operators but once the fourth is added a jump in latency is observed. The reason for this jump is that at four blur operators, the CPU is saturated. This means that the blur operators cannot keep up with the frame rate and frames that are sent from the producer do not get picked up instantly and wait until either the blur operator finishes or until the next frame is produced. In the former case the wait results in increased latency while in the latter case the frame that was waiting will get dropped.

The conclusion we draw from this experiment is consistent with that of the previous one. Due to significant frame drops, the multicast transport mechanism is not suitable for local processing even when using parallel processing pipelines.

5.3 Experiment 3: YARP Overhead

The previous two experiments show that the multicast transport mechanism is not suitable for local processing, and that using shared memory is slightly more efficient than using local TCP/IP communication. The goal of our final experiment is to measure the overhead of the shared memory transport mechanism, compared to a stand-alone process running the entire pipeline.

With a stand-alone process, there is no inter-process communication, and all the CPU power is spent on the processing pipeline. In order to measure the overhead accurately, we modified the basic processing pipeline of Figure 5.1 slightly. The producer now produces a single frame and waits until it is received by the receiver (the receiver sends a notification to the producer). In order to guarantee delivery, we used a synchronization feature of YARP. This experiment therefore gives a strict upper bound on the overhead of YARP.

Due to the simple configuration, the overhead is identical whether measured in terms of CPU cost, latency, or observed frame rate. Figure 5.8 shows the latency of YARP compared to the stand-alone process. The overhead is most significant for a short processing pipeline, about 77%, but is quickly reduced to 50–60%. The reason for the high overhead for shorter pipelines is that for a pipeline of length n there are $n+1$ communication ports; as there are more blur operators the effect of the additional port are less pronounced.

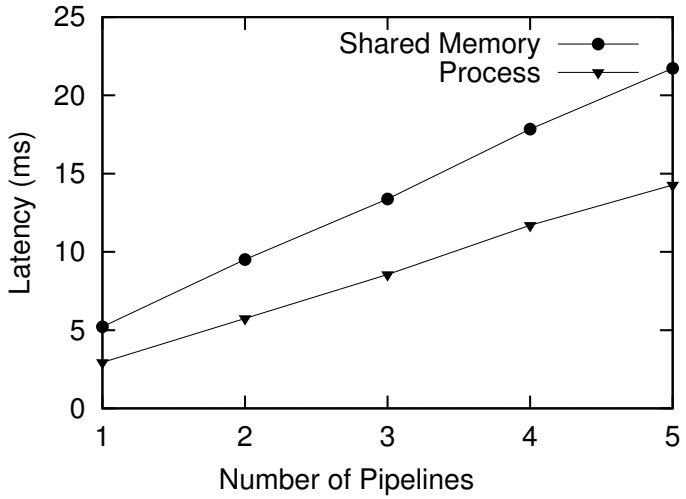


Figure 5.8: Exp. 3: Latency of YARP and stand-alone process.

Based on this experiment, we conjecture that for a complex processing pipeline, we could expect about 50% overhead compared to a well-tuned, handwritten code. We believe that to be a good tradeoff for all the convenience that YARP has to offer.

Chapter 6

Summary

We have described our efforts towards a flexible computer vision infrastructure based on the YARP toolkit. YARP greatly simplifies making the infrastructure flexible towards sensors, hardware, processing, and communication requirements, compared to starting from scratch. We also compared YARP to a number of other toolkits and while we found all of them to be applicable to our scenario we chose YARP mostly for its lightweight characteristics and small semantical footprint.

Overall, we have found YARP to be satisfactory and easy to use. Installing and learning to use YARP took about one man-week, while most of our time was spent on creating hardware drivers and working with the cameras. Our experiments showed that the overhead is a reasonable tradeoff for the convenience gained.

Part II: Range Enhanced SIFT Processing

In this second part of this thesis, we explore two methods for utilizing range images, in combination with regular color images, to increase the performance of SIFT based object recognition.

The first method involves segmenting an input image based on range information. Protruding objects are isolated in the range image and their coordinates translated to the color image. Instead of generating local descriptors for the whole image, the processing effort is limited to the segments of the image that are likely to contain foreground objects.

The second method we investigate focuses on the SIFT algorithm itself. We use the distance information from the range image to compensate for any scale changes between the observed object and the objects kept in a reference database. When SIFT descriptors are calculated for a segment of the input image, it is possible to reduce the computation needed to maintain scale-invariance.

The remainder of this part is organized as follows. We provide the necessary background information in Chapter 7. In Chapter 8 we describe our methods. Chapter 9 details our experimental environment and an evaluation is provided in Chapter 10. Finally, we conclude and discuss future work in Chapter 11.

Chapter 7

Background

In this chapter we provide the necessary background information on the SIFT algorithm and range imaging. We begin by giving a general overview of local descriptors in Section 7.1. Then we provide some details of the SIFT algorithm needed to understand our work in Section 7.2. In Section 7.3 we discuss methods of storing and retrieving local image descriptors. Finally we explain range imaging in Section 7.4.

7.1 Local Image Descriptors

Local image descriptors are multi-dimensional data points describing a small area of an image. These descriptors are typically robust and invariant to a number of transformations and have high applicability in a range of areas, such as robotic mapping and navigation, image stitching, 3D modeling, gesture recognition, video tracking, and more.

Local image descriptors are also one of the fundamental techniques used in computer vision tasks such as object recognition. A database of descriptors for images of known objects enables a robot to recognize these objects by generating descriptors for a frame from a camera and performing a nearest neighbor search against the descriptors stored in the database. Each neighboring descriptor found in the database then contributes a vote for the object to which that descriptor belongs, and the object with the most votes is considered the most likely candidate.

Many algorithms exist for generating such local descriptors. These include SIFT (Lowe, 2004), GLOH (Mikolajczyk and Schmid, 2005), RIFT (Lazebnik et al., 2004), SURF (Bay et al., 2006), PCA-SIFT (Ke and Sukthankar, 2004), Eff^2 (Lejsek et al., 2006), and more.

Studies have shown that SIFT descriptors (and their derivatives) provide the most robust and stable features (Mikolajczyk and Schmid, 2005; Lejsek et al., 2006).

7.2 Scale-Invariant Feature Transform (SIFT)

Scale-Invariant Feature Transform (SIFT) has been one of the predominant methods for local image descriptor computation since it was published in 1999 by David Lowe (Lowe, 1999). They were further described in (Lowe, 2004) where they were shown to be invariant to image scaling, translation and rotation, and partially invariant to illumination changes and affine or 3D projection.

The SIFT method can, in general, be split into two phases. The first phase is the interest point detection where potential interest point locations in the image are found using scale-space extrema detection. This identifies points in the image that are likely to produce stable local descriptors at different scales of the image. Some of these points are filtered out based on measurements of their potential instability, such as low contrast regions or regions with high edge response. The second phase is the actual local descriptor generation, where the descriptors for the remaining interest points are calculated.

Attempts have been made to improve SIFT in various ways to either make the local descriptors more distinctive and/or robust, or to decrease the processing effort needed to generate and use the descriptors. PCA-SIFT uses principal components analysis to increase robustness and reduce dimensionality resulting in faster matching times (Ke and Sukthankar, 2004). RIFT descriptors also reduce dimensionality to increase processing speed (Lazebnik et al., 2004). Eff^2 descriptors reduce the dimensionality as well, but also constrain the number of descriptors and apply various optimizations (Lejsek et al., 2006). SURF is a SIFT-inspired descriptor that is based on sums of approximated 2D Haar wavelet responses and makes efficient use of integral images for faster descriptor computation (Bay et al., 2006).

For the purpose of this thesis, we chose to use the original SIFT method as the baseline for our study. There are three main reasons for this. First, we feel that this will provide a better insight into the properties of our methods, as numerous other comparative studies use the original SIFT method as their baseline. Secondly, the SIFT method is well documented and discussed in the literature. Finally, open source implementations exist for SIFT that we are able to use. One such in particular, developed by Rob Hess at Oregon State University (Hess, 2010), serves as the basis of our work.

In the following we describe the steps involved in the creation of SIFT descriptors. For further details, see (Lowe, 2004).

7.2.1 Scale-Space

SIFT searches for stable features across all possible scales, using a continuous function of scale known as scale-space (Witkin, 1983). More specifically, Lowe proposes using the difference-of-Gaussian function to construct a scale-space pyramid.

7.2.2 Difference-of-Gaussians

In the first step, a scale-space is constructed by convolving the input image several times with a Gaussian function. The scale-space for an input image $I(x, y)$ is defined by the function:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \tag{7.1}$$

where $*$ is the convolution operator and $G(x, y, \sigma)$ is a variable-scale Gaussian. In each iteration, the width of σ is increased by a constant factor k.

After producing a scale-space for an image, a number of difference-of-Gaussian (DOG) images are computed by subtracting adjacent scales. These are defined by the function:

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma) \tag{7.2}$$

7.2.3 Octaves

Instead of continuing the increase of σ, the image that has twice the initial value of σ in that scale can be re-sampled by taking every second pixel in each row and column. This reduces the size of the image by half without affecting the accuracy of sampling relative to σ and thereby greatly reduces computation. This generates a scale-space pyramid where each set of same-sized scale images is known as an octave.

In (Lowe, 2004), features are searched for across all possible scales. In the context of octaves, this means that a new octave is generated until the size of the generated image

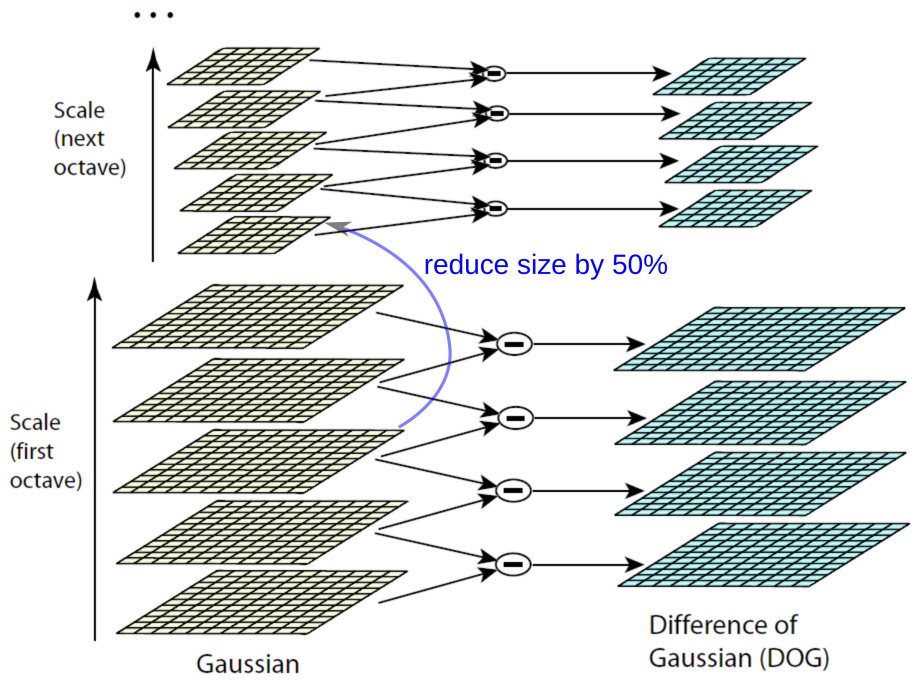


Figure 7.1: Difference-of-Gaussians pyramid [image adapted from (Lowe, 2004)].

becomes less than the size of the Gaussian kernel being used. Figure 7.1 shows a graphical explanation of the difference-of-Gaussian pyramid and the relation of octaves and scales.

7.2.4 Extrema Detection

After the scale-space pyramid has been created, keypoint candidates are found by locating local maxima and minima in the scale-space. This is done by comparing each sample point to its eight neighbors in the current image and nine neighbors in the scale above and below, as shown in Figure 7.2. It is selected as a potential interest point only if it is either larger or smaller than all of its neighbors.

7.2.5 Filtering

After locating keypoint candidates, a detailed fit to the nearby data for location, scale and ratio of principal curvatures is performed on each of them. This allows points to be rejected that either have low contrast or high edge response. Filtering is important at this point in the algorithm as it reduces the needed computation at later stages since low quality keypoint descriptors will not be generated.

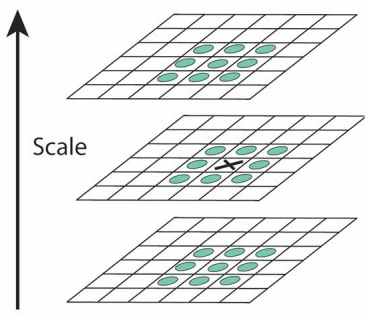


Figure 7.2: Extrema detection in a DOG scale [image from (Lowe, 2004)].

7.2.6 Orientation Assignment

For the remaining keypoints, a consistent orientation is assigned to each one. This is done by creating a 36 bin gradient magnitude histogram from pixels in the neighboring area of the keypoint with each bin representing a 10° range. The dominant gradient direction (the bin with the highest magnitude) is used as the orientation for the keypoint. Additionally, when other gradient directions have a magnitude of 80% or higher compared to the largest one, a new keypoint is created with that orientation. Keypoint descriptors can then be represented relative to this orientation and are therefore invariant to image rotation.

7.2.7 Descriptor Generation

The final step is to calculate the actual SIFT descriptor for each of the remaining keypoints. This is done based on gradient magnitude and orientation around the keypoint and the result is a 128 dimensional vector. Further explanation is beyond the scope of this thesis as it does not relate to the methods proposed here; for a detailed explanation of the descriptor generation, refer to (Lowe, 2004).

7.3 Database Storage and Retrieval of Descriptors

Local image descriptors can be compared to each other using simple nearest neighbor calculations. In order to compare a set of SIFT descriptors generated for a query image, descriptors for reference images need to be stored in some sort of a database along with a reference to their origin. This origin is often the image that the descriptors were generated from, but in some cases the origin may refer to some other entity such as an identifier of

a real world object that the reference image depicted. In this thesis we will use the latter definition, that is, a descriptor database maps SIFT descriptors to an object ID.

A lookup can be performed in such a database by comparing the query image descriptors to the descriptors stored in the database. For each query descriptor, a k-nearest-neighbor search is performed against the database, either by performing a full scan or using some indexing technique, such as the NV-Tree (Lejsek et al., 2009), or some of the methods described in (Samet, 2006). A simple voting mechanism is then employed where each of the k nearest neighbor descriptors vote for the object ID they refer to. After searching for all the descriptors generated from the query image, the object ID with the most votes is considered to be the closest match.

7.4 Range Imaging

A range image, sometimes referred to as a depth map, is a single channel image, where the value of each pixel represents the distance from the camera to a solid object, as opposed to the intensity of light reflected by that object as in regular images. The higher the pixel value, the longer the distance from the camera to that point.

Many techniques can be used to produce a range image. Some examples include stereo triangulation (stereo vision), structured light, and time-of-flight measurement. Since we use a time-of-flight camera in this thesis, we briefly describe the basic principle here.

The time-of-flight principle is used in a variety of devices to measure distances. Several technologies exist but for the purpose of this thesis, we use a device that uses an amplitude-modulated near-infrared light source to illuminate the scene. A CMOS/CCD sensor picks up the reflected light and the phase-shift of the incoming signal is calculated by taking four samples, each phase-shifted by 90° relative to each other. The phase of the incoming signal is compared to that of the source and the phase-shift between the two used to calculate the distance value.

Figure 7.3 shows an example of a color image (Figure 7.3(a)) and a range image (Figure 7.3(b)) pair from our experimental setup. The black area in the range image is "undefined" as the camera does not register a proper reading for it. The white area around the bottom of the poster, however, is the background wall. Different material reflects the near-infrared light in different ways and here we can see that the wall is highly reflective while the floor and a panel mounted on the wall are not reflecting enough light back for the camera to determine a distance value.

(a) Color image (b) Range image

Figure 7.3: A color image (a) and the corresponding range image (b). The poster stands about 4 meters from the back wall.

Chapter 8

Methods

In this chapter we describe in some detail the methods that we use in our experimental setup. In Section 8.1 we describe the overall vision processing pipeline, and outline the techniques used in each module. We then focus on the two methods that we propose to reduce the cost of local descriptor computations.

First, the range image is partitioned based on the distance information, and these partitions are used to create the image segments which protrude from the background. In Section 8.2 we describe the image segmentation process in more detail.

Second, each image segment is rescaled to match a reference distance used in our object collection, allowing to reduce the requirements for scale invariance. In Section 8.3 we describe the method used to reduce scale-space computations.

8.1 The Processing Pipeline

Figure 8.1 shows an overview of the pipeline setup. The input to the pipeline is produced by two cameras, a regular video camera and a range camera. These images traverse through the pipeline and are manipulated by a number of modules. At the end of the pipeline, a database lookup is performed in an attempt to identify objects in the input images. In the following, we describe each module of the pipeline in more detail.

8.1.1 Fuser

The Fuser acquires the frames that get produced from each camera and produces a composite RGB/range frame.

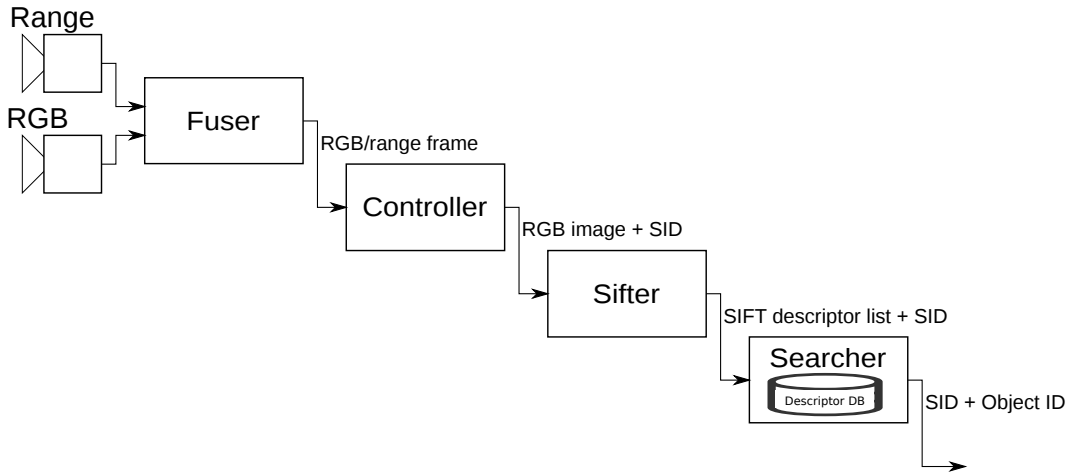


Figure 8.1: The processing pipeline.

Input: Two ports receive image streams from the cameras. One port receives the three channel RGB image from the color camera, while the second one receives the single channel range image from the range camera.

Output: A single port produces a composite data structure, containing one RGB frame and one range frame.

Method: Since no hardware synchronization is done between the two cameras, the Fuser minimizes the latency between the two frames by always keeping track of the two most recent frames from each camera ($[C_0, C_1]$ for the color camera and $[R_0, R_1]$ for the range camera). It also stores the arrival time of each frame, represented here by $t(\cdot)$. When a new composite frame is published, it picks a pair of frames $[C_i, R_j]$ such that $|t(C_i) - t(R_j)|$ is minimal.

To avoid publishing redundant frames, the Fuser then waits until it acquires a fresh frame from both cameras before publishing a new composite frame.

8.1.2 Controller

The Controller is the logic unit of the pipeline. It receives the composite RGB/range frame from the Fuser and produces a set of (zero or more) RGB segments for further processing. Each segment produced is assigned a unique identification number that we call a segment ID (SID) to enable later stages in the pipeline to reference a specific segment. This SID is a simple running counter that is reset each time the system is restarted so it is only unique within each session.

Input: A single port receives a composite RGB/range frame.

Output: A single port produces RGB segments along with their SID. The segment images should contain one or more potentially interesting subjects, depending on the pipeline configuration.

Method: There are two main processing operations that the Controller executes:

- **Image Segmentation**: The Controller attempts to isolate foreground objects in a received frame and produces one RGB image per detected object. We call these produced images *segments*. Each segment is assigned a unique segment identification number (SID) to enable referencing in later stages of the pipeline. The produced segments are essentially only the regions of the original RGB input frame that contain foreground objects as detected in the range image.

  The main purpose of this image segmentation is to eliminate the processing time that would be spent on generating SIFT descriptors for the regions of the input frame that contain only the background. This reduces the response time for object recognition and may also facilitate parallel processing if multiple objects are isolated from a single input frame. The image segmentation is further described in Section 8.2.

- **Segment Rescaling**: After isolating foreground objects, the Controller may rescale the segment images depending on the pipeline configuration. The idea is that the range information can help eliminate expensive computation steps in the SIFT algorithm. The segment rescaling is described in more detail in Section 8.3.

8.1.3 Sifter

The purpose of the Sifter is to compute and produce a list of SIFT descriptors for an image it receives.

Input: A single port receives a segment RGB image along with its segment ID.

Output: A single port produces a list of SIFT descriptors computed from the input image, along with the segment ID of the incoming segment image.

Method: The SIFT generation software of (Hess, 2010) is applied to the input image. If the Sifter receives image segments that have been rescaled, it only generates a single octave when calculating the input image scale-space. This reduces both the computation time and number of descriptors that are generated. Further details are provided in Section 8.3.

8.1.4 Searcher

The Searcher performs a nearest neighbor search in a SIFT descriptor database. The database contains descriptors for images of objects that we wish to recognize. It produces its results, enabling any interested modules to subscribe to notifications of any object matches.

Input: A single port receives a list of SIFT descriptors along with the segment ID they were generated from.

Output: A single port produces the results from a database lookup. This is a key-value pair where the key is the segment ID of the incoming SIFT descriptors, and the value is an object ID of the matched object from the database.

Method: The Searcher executes a sequential scan over all descriptors in its database. This proved to be sufficient for our needs but for a real-world application, more sophisticated indexing needs to be implemented.

For the purpose of this thesis, we only print match results to a log file for analysis but more sophisticated logic units could easily manipulate this information for higher level intelligence.

8.2 Image Segmentation

As described above, in our setup we use two cameras, a color camera and a range camera, to obtain color+range information. In this section we describe the methods used to isolate foreground objects from the color video stream by using the range image provided by the range camera.

8.2.1 Alignment

The alignment of video streams from color cameras with range data from range cameras has been the subject of a number of studies (Reulke, 2006; Santrac et al., 2006; Lindner et al., 2008). Most of these methods strive for pixel accurate alignment using complicated algorithms. While our system would clearly benefit from high alignment accuracy, we opted to only perform a simple, manual calibration of the cameras for our experiments. This is sufficient to enable a simple projection from the range coordinates to the RGB image with trivial linear scaling.

The manual camera alignment was done by setting up markers in the environment and adjusting the camera angle while visually observing the video streams from both cameras.

8.2.2 Range Partitioning

When a pair of images (color+range) are received, the range image is partitioned based on pixel values. In our setup we use 9 range bins that are divided equally over the potential pixel value range. As the range camera we are using has an effective range of about 8 meters, using 9 range bins means that each bin represents roughly a 90 cm range. We create a binary image for each of the bins and then scan once through the original range image. For each pixel, we check which bin it falls into and flip that pixel on for the corresponding bin image. This results in 9 binary images, each containing white blobs if objects are found in that image's assigned range. A top-down view of a hypothetical scene is depicted in Figure 8.2 along with the binary images for the bins that contain objects in that particular scene. Note that the objects, depicted as blue shapes in the top-down view, are standing on the ground in our hypothetical example.

The optimal number of bins will be highly dependent on the type of workload and the expected level of detail in the scenes being analyzed. The choice of using 9 bins for our setup is somewhat arbitrary but in the context of our workload which is described in Section 9.2 we believe that this is a good fit.

8.2.3 Image Segmentation

After generating the range-partitioned binary images, we apply the contour-finding algorithm built into the OpenCV library to each of them. Once we have the contours of the "blobs", we calculate their bounding box, resulting in a region of interest that corresponds to an isolated foreground object[2]. The coordinates are projected using simple linear scaling and the region extracted from the RGB image.

8.3 Scale-Space Reduction

If the pipeline is configured to perform image rescaling, some further processing needs to be done on each image segment. Image rescaling means that the Controller rescales

[2] A single segment may contain more than one object. In this study, we do not consider this case.

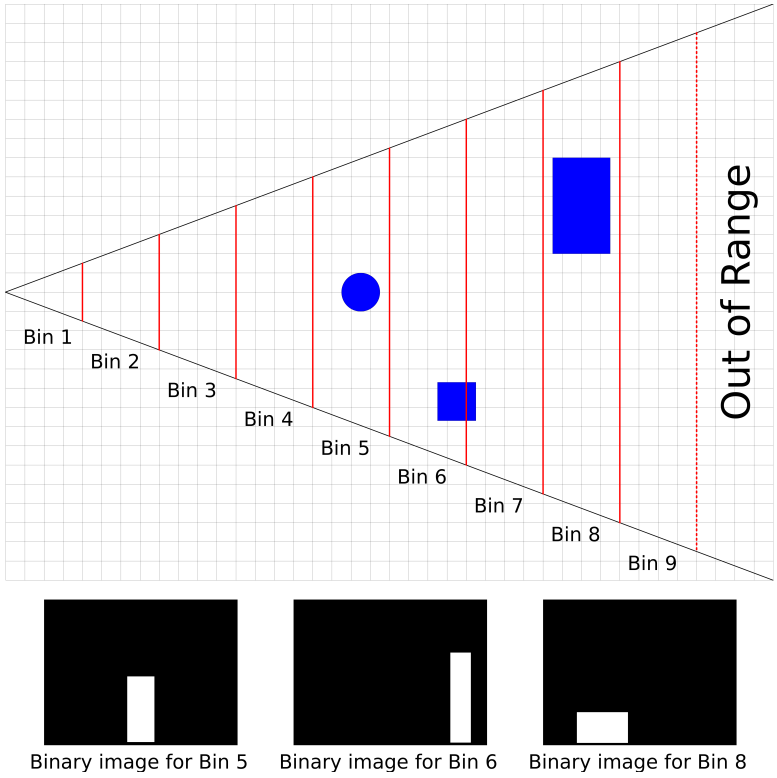


Figure 8.2: Illustration of range partitioning (see text).

the image segment so that it looks as if the camera was positioned at a specified distance.

8.3.1 Segment Rescaling

In order to rescale segments, the Controller must know three things: 1) the distance to the object, 2) the height of the input segment, and 3) the distance at which the object should appear to be. The object distance is simply read from the range image by taking the median value of all range pixels for that segment region. The median value is chosen to minimize errors caused by potential outliers and it is also likely to be representative for the dominant object in the segment if there should be more than one objects. The segment height is simply the pixel height, and the desired distance is given as a runtime configuration option to the Controller.

The scaling factor is found by looking at the problem as a set of similar triangles separated by the point B which represents the focal point of the camera. This is depicted in Figure 8.3. When considering a particular object, f and h are fixed. Given a value for d as the measured distance from the range camera, and a corresponding value for h' as the

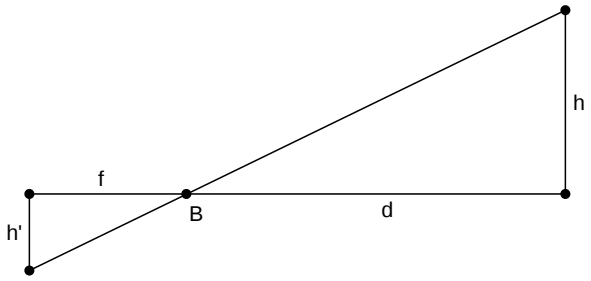


Figure 8.3: Illustration of the rescaling principle (see text).

observed pixel height of the segment, we can calculate a new value for h' for any value of d since $\frac{f}{h'} = \frac{d}{h}$.

8.3.2 Processing Reduction

If a segment has been rescaled before local descriptors are calculated, the need for scale invariance should be reduced significantly. This means that we should be able to reduce the SIFT computation by limiting the scale-space. This is in fact one of the main subjects of our experiments and to support this, the Sifter can be started with a runtime configuration option that specifies the number of octaves to generate when computing SIFT descriptors.

In Chapter 10, we consider two cases: multi-octave computation where processing is not reduced, and single-octave computation, where only a single octave is generated.

8.4 Summary

In this chapter, we have outlined the two techniques used to reduce the cost of local descriptor computation using range information. In the following chapters we analyze the impact of these techniques.

Chapter 9

Experimental Setup and Hypotheses

Our hypothesis is that we can use the two methods described in chapters 8.2 (Image Segmentation) and 8.3 (Scale-Space Reduction) to increase the throughput of our vision system while positively affecting the object recognition result quality.

In this chapter we explain the setup and execution of our experiments. Due to time constraints, we present simple experiments that do not prove the validity of our hypothesis. The results do, however, strongly support the hypothesis and they give very valuable insight into the tradeoffs between performance and quality.

9.1 Hardware and Software

In our setup, we use two cameras; a Point Grey Chameleon CCD color camera and the Swiss Ranger SR-3000 depth camera from Mesa Imaging AG. The two cameras were put together on a rig, as shown in Figure 9.1.

The Point Grey Chameleon camera is a USB 2.0 color camera with a Sony ICX445 1/3“ EXview HAD CCD. It can provide 1296x964 pixel video resolution at 8 bits per pixel and 18 frames per second. As the driver software for the Chameleon camera is proprietary software, we had to implement a YARP device driver wrapper for it.

The SR-3000 camera provides both intensity and depth information. Depth information is obtained using a phase-measuring time-of-flight principle. The camera allows adjustments of various parameters, such as integration time and amplitude threshold, which makes it suitable for a variety of applications and environments. The camera produces 176x144 pixel images with 16-bit depth resolution.

We also developed a small utility application for publishing the SR-3000 camera video streams onto YARP ports. Due to a limitation of the YARP device implementation, we were unable to incorporate this into the standard YARP device appliance as it only supports devices that produce a single output stream. Discussion with the YARP developers has produced some ideas on how to improve this, which may be incorporated into YARP at some point.

To simplify the execution of experiments, a utility application was developed that recorded the output of the Fuser module—the combined camera and range image stream—to a file. The file contains the exact data stream sent through the Fuser's output port along with message timing data. Another utility application could then be used to read this file and play back the data stream in real-time. This greatly simplifies the execution of experiments since the camera equipment does not have to be connected to the computer and the experiment scene does not need to be set up at all times.

9.2 Object Collection and Workload

The workload was created by placing specific objects at specific distances from the camera rig and recording a short sequence. The objects that we chose were six movie posters which were placed 2, 3, 4, and 5 meters away from the camera rig. The background was a white wall at a distance of approximately 7 meters. One sequence was recorded for each object at each distance, resulting in a total of 24 recordings. Since the scene is stationary, only a short recording was made (~3 seconds) which was then looped during actual experiment runs. Figure 7.3 showed an example of a movie poster and the background.

Once all the recordings were done, a set of reference object databases was created as follows. First, a single color image frame from each recording was extracted and saved as a regular image file. The poster in the image was then manually isolated by using image editing software to crop the image. This provided us with a still image of each poster at each distance.

An object database then consists of the SIFT descriptors of all images at a given distance. Each poster was assigned an ID that was consistent between all databases. For each distance, two types of databases were created, one containing the original SIFT descriptors and the other containing descriptors that were generated from a single scale-space octave. We refer to the former version as multi-octave (mo) and the second as single-octave (so). We therefore produced 8 reference databases in total (2 types for all 4 distances).

(a) The camera rig. (b) The setup of the camera equipment.

Figure 9.1: Cameras and experiment setup.

Admittedly, this workload is not representative for many real world scenarios in which robots might have to deal with a much more complex environment such as an office where multiple objects overlap and clutter the scene. Our setup also assumes that a single object will fall into a single range bin which will not always be the case in real world situations. Properly segmenting such a complex scene is a highly complicated task and outside the defined scope of this thesis due to time constraints. We opted to focus instead on evaluating the feasibility of using the segmentation and rescaling methods described in Sections 8.2 and 8.3 to increase performance of object recognition tasks using local image descriptors. The use of movie posters was convenient as we had access to many such posters and chose ones with different properties. One had very specific visual properties (a poster for a cartoon) while another was very uniform and generated few descriptors. We also chose a few that had similar visual properties amongst themselves. Figure 9.2 shows the posters that we used for our experiements.

9.3 Compensating for Collection Size

We acknowledge the fact that the object databases we are working with are quite small. One reason for using a small database is simplicity as we can use a simple sequential scan to perform descriptor searches. By using a sequential scan we also avoid relying on approximation methods that are commonly used for large multidimensional data collections. Another reason is to keep the effort spent on searching to a minimum to avoid it

(a) Megamind (b) Life As We Know It (c) The Tempest

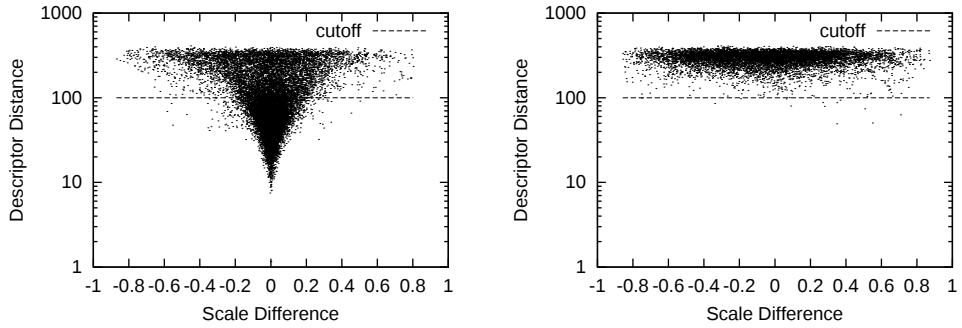
(d) Due Date (e) Charlie St. Cloud (f) The Town

Figure 9.2: Movie posters used for the experiments.

affecting our results. The Searcher should not be a bottleneck in the pipeline as we are focusing on the SIFT processing methods.

To compensate for this small database size to some degree, we set a limit to the distance between a query descriptor and its nearest neighbor found in the database. If the nearest neighbor is farther away than this cutoff distance, we do not consider it to be a match. This simulates a large-scale collection, where descriptors that are far away can not be expected to be found.

To determine this cutoff value, we ran a single-octave query against a single-octave database where the database distance was the same as the query distance. We ran a separate query for each object and logged both the Euclidean distance between the query descriptor and its nearest neighbor from the database as well as the difference in scale between the query descriptor and its nearest neighbor. Since the database and query are both single-octave, the scale difference is in the range $[-1...1]$. We also logged whether the nearest neighbor descriptor found in the database belonged to the query object. If that was the case, we defined that nearest neighbor to be a match. Otherwise, if the nearest neighbor descriptor belonged to a different object, we defined it to be a miss.



(a) Matches (b) Misses

Figure 9.3: Comparison of descriptor distance and scale difference (query: 2m; DB: 2m).

Figure 9.3 shows the result of these runs. Each point in the figure represents a query descriptor and its nearest neighbor in the database; the figure aggregates the results for 10 frames for each query object, for a total of 60 frames. The x-axis shows the scale difference between the two while the y-axis represents their Euclidean distance. Figure 9.3(a) shows the pairs that were defined as a match (the query object was the same as the object that the nearest neighbor descriptor from the database belonged to) while Figure 9.3(b) shows the pairs that were defined as misses.

Consider first the distribution of matching descriptors in Figure 9.3(a). The figure clearly shows that a large number of descriptors are found in the exact same scale-space level and at a very small distance. There are, however, also many descriptors that are found at a significantly larger distance, and many of these are found in different scale-space levels, indicating that these are worse matches that are likely to be lost in a large collection.

Looking at the distribution in Figure 9.3(b), which shows the distribution of misses, it is clear that the cutoff value lies close to 100 as nearly all misses fall above that distance. We also see in Figure 9.3(a) that there are plenty of matches below this value, indicating that a cutoff at 100 will still provide a robust match to an object.

Of course, some of the matches are not actual matches, but rather misses where the nearest neighbor happens to come from a descriptor that belongs to the query object. If we assume that the distribution of those misses is similar to the ones in Figure 9.3(b), we will also have eliminated these by choosing this value for the cutoff.

Note that this cutoff value is consistent with the observations of (Lejsek et al., 2009). They showed that in a large scale collection, descriptor distance impacts the likelihood of being found. Descriptors with distance less than 25 were consistently found, while descriptors with distance greater than 150 were rarely found. Descriptors with a distance

of 100 had a 50% chance of being found, which indicates that 100 is indeed a good cutoff value.

9.4 Metrics

The primary performance metric that we use is the throughput of the pipeline, measured in frames-per-second (FPS). The frame rate was measured for all the modules in the pipeline but for the purposes of our experiments, we concentrate on the Sifter module as our methods are focused on improving the performance of descriptor generation. Measurements also allowed us to make two observations: 1) That the throughput of the Sifter and Searcher modules were very close, which means that the effort spent on searching for descriptors does not affect the throughput of the pipeline in our experiments. 2) That the overhead of performing image segmentation is very small (typically about 0.002%).

Any improvement in performance must be weighed against a potential loss in the quality of the results. We indicate this quality by measuring the ratio of descriptors that match the correct object to the total number of descriptors generated for the query, using the definition of matching descriptors from Section 9.3 above.

Chapter 10

Results

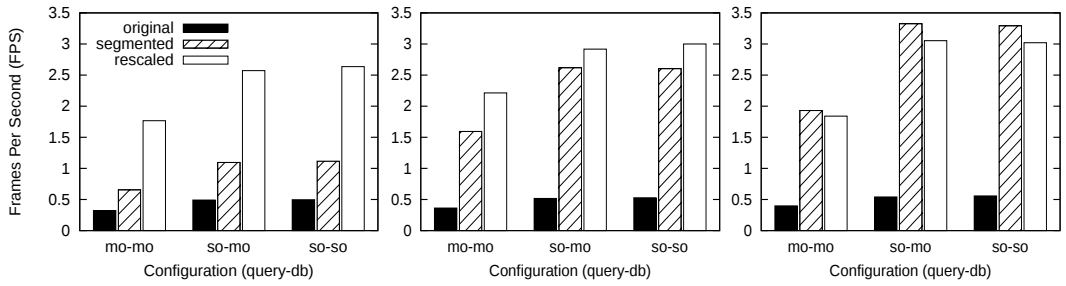
In this chapter, we report on our evaluation of the the two methods that we propose for utilizing range information to speed up local descriptor generation, namely image segmenting and segment rescaling. We compare these two methods to a baseline measurement and provide insight into the results.

The results presented here are extracted from a single experimental run where all the recordings were run through the pipeline with multiple configurations. Each recording was run until the Searcher module had received 10 frames. In total, we ran 384 recordings through the pipeline (6 objects, 4 query object distances, 4 database distances, 2 query types, multi-octave (mo) and single-octave (so), and 2 database types, also multi-octave and single-octave). Unless otherwise specified, results for a specified configuration are aggregated for the 6 objects. In the following, we report representative results that clearly demonstrate the tradeoffs between performance and quality.

10.1 Performance

The first thing we wish to understand is whether using our proposed methods result in increased performance of the overall computer vision pipeline. To evaluate the performance impact of our methods, we compare runs where the distance to the query object varies while the distance to the objects in the database is fixed. For the case where we scale the query segments, we consider both the case where a segment size is increased (up-scaled) to match the database distance, as well as when it is decreased (down-scaled).

Figure 10.1 shows the case where objects in the reference database are located 2 meters away from the camera while the distance to the query object is varied between 2,



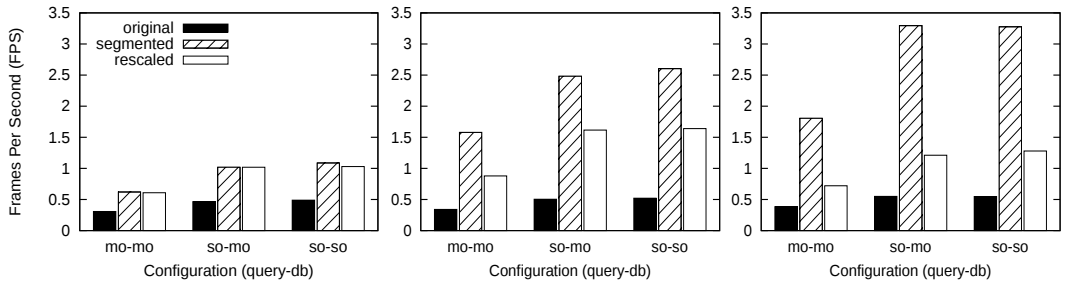
(a) Query distance: 2m (b) Query distance: 3m (c) Query distance: 4m

Figure 10.1: Performance (FPS) of Sifter with database distance fixed at 4 meters.

3, and 4 meters (Figures 10.1(a), 10.1(b), and 10.1(c) respectively). The three clusters of bars in each of the histograms represent different configurations of the types of query and database being used. The first cluster is a multi-octave query and database (mo-mo), then, a single-octave query but a multi-octave database (so-mo) and the third cluster is where both query and database are single-octave (so-so). Within each cluster, there are three bars. The first one (original) shows the throughput of the original method, that is, where the Controller sends all received frames directly to the Sifter without performing any modifications or processing. The second (segmented), shows the throughput where segmentation is performed by the Controller. The last bar (rescaled), shows the throughput where the Controller performs both segmentation and rescaling.

When considering Figure 10.1, we immediately notice that the throughput of the original method stays relatively constant regardless of the query distance. This to be expected since the whole frame is sent through the pipeline in all cases, so the amount of processing stays about the same. There is a very slight increase in the original method throughput as the query object is positioned farther away. We attribute this to the fact that the background in our setup is a very plain, white wall. The number of generated descriptors, therefore, depends primarily on the size of the object in the image, and as it is placed farther away, it appears smaller and therefore produces fewer descriptors. We also notice a 38-55% increase in the throughput for the original method if single-octave query descriptors are being generated. This is because the number of descriptors that are generated decreases by about half when descriptors are only generated for a single octave. These properties of the original method remain consistent throughout all configurations.

With the segmented method, we see a significant increase in throughput compared to the original method. The gain is least pronounced in Figure 10.1(a), but increases steadily as the query distance grows. This is because at a query distance of 2 meters the object fills a very significant portion of the image. As the query distance increases, the size of the object decreases, and along with it the size of the segment sent to the Sifter for processing.



(a) Query distance: 2m (b) Query distance: 3m (c) Query distance: 4m

Figure 10.2: Performance (FPS) of Sifter with database distance fixed at 2 meters.

Again, we see a similar proportional gain in generating single-octave query descriptors as we did with the original method. This applies throughout all query distances.

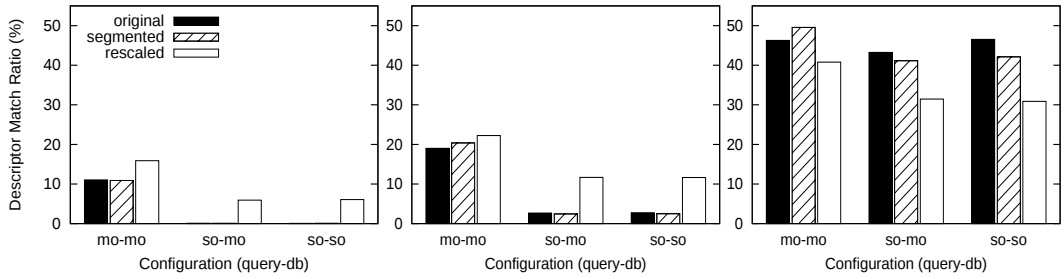
The last method we examine in Figure 10.1 is the rescaled method, where the segment is resized to match the size of objects in the database before being sent for processing by the Sifter. The key observation here is that the throughput increase is immediate even though the query distance is small. As the segment is down-scaled to the size of images in the database (in this case, at a distance of 4 meters), much less processing is performed. The throughput of the rescaled method remains constant over all query distances as the Controller always sends the same size of segments for processing. This also indicates that the cost of the resizing operation is negligible.

We now turn our attention to Figure 10.2 which shows throughput measurements with a database distance of 2 meters and query distance varied over 2, 3 and 4 meters as before. In this case, the segments are up-scaled and increase in size.

First of all, we notice that both the original and segmented methods behave very similarly as before. This is not surprising as there is no difference in the queries being made. On the other hand, we see that the throughput of the rescaled method has decreased significantly compared to Figure 10.1. The reason for this is simply that the rescale method resizes the query image to match the size of the images in the database. In this case, the database distance is 2 meters, which means that large segments are being sent to the Sifter for processing.

10.2 Quality

The second thing we have to establish is how the use of our two proposed methods impacts the quality of object recognition in our computer vision pipeline. We measure the quality of query runs using the ratio of the number of descriptors that contributed to a correct



(a) Query distance: 2m (b) Query distance: 3m (c) Query distance: 4m

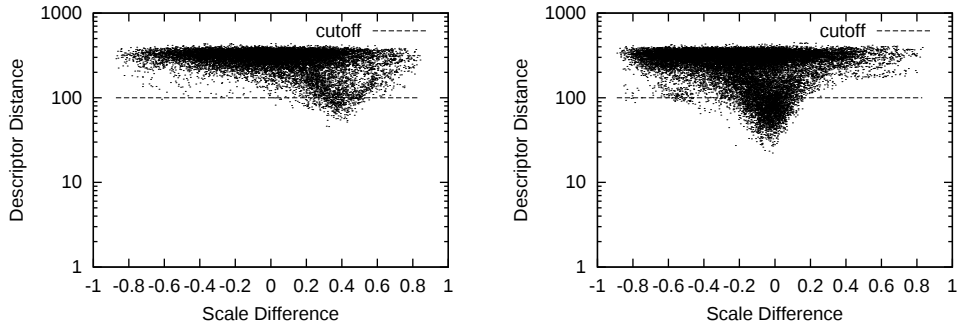
Figure 10.3: Quality of Searcher with database distance fixed at 4 meters.

match to the total number of descriptors generated for that query. As with the performance measurements, we run a set of queries for different configurations over a database at a fixed distance while varying the query distance. We also consider both up-scaling and down-scaling of the query image.

Figure 10.3 shows the same configuration combinations as before. Objects in the database are at a fixed distance of 4 meters while query distances are 2, 3, and 4 meters for Figures 10.3(a), 10.3(b), and 10.3(c) respectively. In these figures, the y-axis shows the percentage of descriptors that are contributing to a correct object match.

We first note that as the query distance is closer to the database distance, the results improve. This is consistent for all configurations and methods. Next, we see that for the multi-octave query and multi-octave database (mo-mo), there is only a slight difference in quality between the original and segmented method, in favor of the segmented method. This is because, in actuality, the query being sent to the Searcher is quite similar between these two methods. The majority of descriptors in the original method are being generated in the same region as the segment that is sent in the segmented method. The only difference is that in the original method, a very small number of descriptors is generated on the plain background. These background descriptors explain the slightly lower quality of the original method. Of course, if the background was more complex, this difference would be more pronounced.

Looking at figures 10.3(a) and 10.3(b), we notice that both the original and segmented methods display very low quality for single-octave query configurations as the query distance is not the same as the database distance. A closer look at the data reveals that out of an average of 530 descriptors that are generated per query, an average of only 0.5 descriptors fall below our cutoff distance of 100 and are, therefore, used to contribute to a match. Figure 10.4(a) shows how the nearest neighbor descriptors found in the database are shifted due to the difference in distances between the query and database descriptors



(a) segmented (b) rescaled

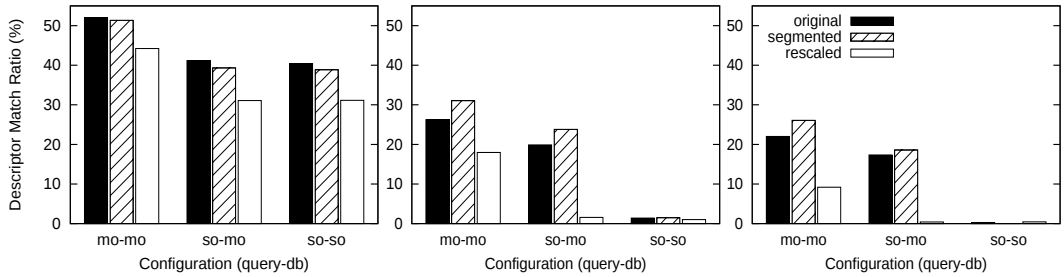
Figure 10.4: Comparison of descriptor distance and scale difference (query: 3m; DB: 4m).

(note that the figure shows accumulated results for all 60 frames of the run). It also shows that only a small percentage of the descriptors fall within the cutoff value.

In contrast, Figure 10.4(b) shows how the rescaled method seems to be producing much better results for the single-octave query configurations, albeit not quite as good as the baseline original method in multi-octave query configuration. Looking again at the data behind the rescaled method quality, we see that out of an average of 460 descriptors being generated per query, roughly 28 of them now fall within our cutoff range and nearly all of those are matching the correct object.

Although the rescaled method displays slightly worse result quality in all configurations where both query and database distance is 4 meters (Figure 10.3(c)), the match quality index is still at around 30% which is much higher than, for instance, the multi-octave baseline method when the query and database distances differ. We therefore do not consider this to be a problem. The reason for this decrease in quality is that although no rescaling should be done (since the query distance is the same as the database distance), minor measurement errors are resulting in a small rescaling which affects the result quality.

Finally, we look at Figure 10.5 where the database distance is fixed at 2 meters while query distance is varied as usual. We immediately notice the same trend as in Figure 10.3 that the original method, using multi-octave queries and multi-octave database, gives result quality that is directly related to the difference of distances between the two. A noticeable difference here is that for any single-octave configuration, where the query distance is greater than the database distance, the rescaling method produces very bad results. This means that enlarging the query image does not produce similar descriptors as were generated from the original image stored in the database.



(a) Query distance: 2m (b) Query distance: 3m (c) Query distance: 4m

Figure 10.5: Quality of Searcher with database distance fixed at 2 meters.

10.3 Conclusion and Summary

In this chapter, we have demonstrated the performance and quality characteristics of our proposed techniques. We draw three main conclusions.

First, we found that generating only a single octave for the SIFT query descriptors results in a significant performance gain. However, using single-octave descriptors in the object database does not improve performance and can have a negative impact on the result quality. Assuming that a scalable search technique is employed, we suggest using the single-octave query, multi-octave database configuration.

Second, when querying for an object located closer than the object database reference distance, we gain significant speed improvements from rescaling the query image segment. Also, rescaling improves the quality of lookups performed with single-octave query descriptors. However, if the query object is located farther away than the reference database distance, we observe a drop in both quality and performance. We therefore suggest that segments only be down-scaled as needed before being processed.

Third, the distance of a reference database must be decided by weighing a few factors. One is the expected detail of the objects we want to recognize, as smaller objects require shorter distance. Another factor is the quality of the color camera being used. The third factor is perhaps the most subjective, as there is a clear tradeoff between performance and quality when choosing the database distance.

We conclude that by dynamically deciding whether to downscale a segment, based on the known facts and measured observations, the proposed techniques can provide a significant performance increase and, in some cases, also improve quality of results.

Chapter 11

Conclusions

In the first part of this thesis we described our efforts towards a flexible computer vision infrastructure based on the YARP toolkit. We found that YARP greatly simplifies making the infrastructure flexible towards sensors, hardware, processing, and communication requirements, compared to starting from scratch. We found YARP easy to use, and our experiments showed that the overhead is a reasonable tradeoff for the convenience gained.

In the second part we explored the impact of using range information to improve the performance of SIFT descriptor generation. We proposed two techniques to do this. First, by segmenting the image based on the range data, thereby isolating objects protruding from their surrounding, and sending only the region of the image that contain foreground objects for SIFT processing. Second, by utilizing the range information to decrease the need for scale-invariance in SIFT. Two options were explored to obtain this goal, the first was using the range information to rescale the image to match the size of objects in the reference database, while the second was reducing the number of descriptors generated by limiting the scale-space to a single octave. We found that using a single-octave query configuration improves the performance by a significant factor. Also, while up-scaling the image segments results in both worse performance and quality, down-scaling when needed results in a significant speed improvement and can in some cases provide better result quality. Finally, we observed that carefully selecting the reference distance for the object database is important and must be considered by weighing a tradeoff between performance and quality.

We have demonstrated that the concept of using range information to increase performance of object recognition appears viable. Of course, further experimentation is required in order to fully understand the properties of using our methods in a more realistic

scenario; we have only skimmed the surface of possibilities. One aspect that could improve this work is using more advanced techniques for aligning the color and range image streams either by using specialized hardware (Hahne and Alexa, 2008; Zhu et al., 2008) or advanced algorithms and camera calibrations (Lindner and Kolb, 2007; Reulke, 2006; Santrac et al., 2006). Doing this would then lead to using more advanced segmentation algorithms resulting in better object isolation. Another interesting aspect to investigate is to use a faster method for interest point detection than the difference-of-Gaussian method used by SIFT, such as the Harris corner detection (Harris and Stephens, 1988), that do not necessarily show as much scale invariance.

Bibliography

Bay, H., Tuytelaars, T., and Gool, L. J. V. (2006). SURF: Speeded up robust features. In *Proceedings of the 9th European Conference on Computer Vision (ECCV)*, New York, NY, USA.

Fitzpatrick, P., Metta, G., and Natale, L. (2008). Towards long-lived robot genes. *Robotics and Autonomous Systems*, 56(1):29–45.

Hahne, U. and Alexa, M. (2008). Combining time-of-flight depth and stereo images without accurate extrinsic calibration. *International Journal of Intelligent Systems Technologies and Applications (IJISTA)*, 5(3/4):325–333.

Harris, C. and Stephens, M. (1988). A combined corner and edge detection. In *Proceedings of The Fourth Alvey Vision Conference (AVC)*, Manchester, UK.

Hess, R. (2010). An open-source SIFT library. In *Proceedings of the 18th ACM International Conference on Multimedia (ACMMM)*, Firenze, Italy.

Ke, Y. and Sukthankar, R. (2004). PCA-SIFT: A more distinctive representation for local image descriptors. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, Washington, DC, USA.

Lazebnik, S., Schmid, C., and Ponce, J. (2004). Semi-local affine parts for object recognition. In *Proceedings of the British Machine Vision Conference (BMVC)*, Kingston, UK.

Lejsek, H., Ásmundsson, F. H., Jónsson, B. Þ., and Amsaleg, L. (2006). Scalability of local image descriptors: A comparative study. In *Proceedings of the 14th ACM International Conference on Multimedia (ACMMM)*, Santa Barbara, CA, USA.

Lejsek, H., Ásmundsson, F. H., Jónsson, B. Þ., and Amsaleg, L. (2009). NV-Tree: An efficient disk-based index for approximate search in very large high-dimensional collections. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(5):869–883.

Lindner, M. and Kolb, A. (2007). Data-fusion of PMD-based distance-information and high-resolution RGB-images. In *Proceedings of the International IEEE Symposium on Signals, Circuits & Systems (ISSCS)*, Iasi, Romania.

Lindner, M., Lambers, M., and Kolb, A. (2008). Sub-pixel data fusion and edge-enhanced distance refinement for 2D/3D images. *International Journal of Intelligent Systems Technologies and Applications (IJISTA)*, 5(3/4):344–354.

List, T., Bins, J., Fisher, R. B., Tweed, D., and Thórisson, K. R. (2005). Two approaches to a plug-and-play vision architecture - CAVIAR and Psyclone. In *Workshop on Modular Construction of Human-Like Intelligence*, Pittsburgh, PA, USA.

Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision (ICCV)*, Washington, DC, USA.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110.

Metta, G., Fitzpatrick, P., and Natale, L. (2006). YARP: Yet another robot platform. *International Journal Of Advanced Robotic Systems*, 3(1):43–48.

Mikolajczyk, K. and Schmid, C. (2005). A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(10):1615–1630.

Ng-Thow-Hing, V., List, T., Thórisson, K. R., Lim, J., and Wormer, J. (2007). Design and evaluation of communication middleware in a distributed humanoid robot architecture. In *Workshop on Measures and Procedures for the Evaluation of Robot Architectures and Middleware*, San Diego, CA, USA.

Quigley, M., Conley, K., Gerkey, B. P., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). ROS: an open-source robot operating system. In *International Conference on Robotics and Automation (ICRA) Workshop on Open Source Software*, Kobe, Japan.

Reulke, R. (2006). Combination of distance data with high resolution images. In *International Society for Photogrammetry and Remote Sensing (ISPRS) Commission V Symposium*, Dresden, Germany.

Samet, H. (2006). *Foundations of Multidimensional And Metric Data Structures*. Morgan Kaufmann.

Santrac, N., Friedl, G., and Rojas, R. (2006). High resolution segmentation with a time-of-flight 3D-camera using the example of a lecture scene. Technical Report B-06-09, Freie Universität.

Stefánsson, S. F., Jónsson, B. Þ., and Thórisson, K. R. (2008). Evaluation of a YARP-based architectural framework for robotic vision applications. Technical Report RUTR-CS08004, Reykjavik University School of Computer Science.

Stefánsson, S. F., Jónsson, B. Þ., and Thórisson, K. R. (2009). A YARP-based architectural framework for robotic vision applications. In *Proceedings of the Fourth International Conference on Computer Vision Theory and Applications (VISAPP)*, Lisboa, Portugal.

Thórisson, K. R., List, T., Pennock, C. C., and DiPirro, J. (2005). Whiteboards: Scheduling blackboards for semantic routing of messages & streams. In *Workshop on Modular Construction of Human-Like Intelligence*, Pittsburgh, PA, USA.

Thórisson, K. R., List, T., Pennock, C. C., and DiPirro, J. (2007). OpenAIR 1.0 specification. Technical Report RUTR-CS07005, Reykjavik University School of Computer Science.

Tweed, D., Fang, W., Fisher, R., Bins, J., and List, T. (2005). Exploring techniques for behaviour recognition via the CAVIAR modular vision framework. In *Workshop on Human Activity Recognition and Modelling (HAREM)*, Oxford, UK.

Witkin, A. P. (1983). Scale-space filtering. In *8th International Joint Conference on Artificial Intelligence*, Karlsruhe, West Germany.

Zhu, J., Wang, L., Yang, R., and Davis, J. (2008). Fusion of time-of-flight depth and stereo for high accuracy depth maps. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, Anchorage, Alaska, USA.

School of Computer Science
Reykjavík University
Menntavegi 1
101 Reykjavík, Iceland
Tel. +354 599 6200
Fax +354 599 6201
www.reykjavikuniversity.is
ISSN 1670-8539