Parallel Multisensory Integration for Robotics

eingereichte
PRAKTIKUMSBERICHT
von

B.Sc. Igor Krawczuk

Matr.Nr.
geb. am 05.09.1992
wohnhaft in:
Connollystrasse 3/Q59
München
Tel.: 0178/8782631

B.Sc. Edvarts Berzs

Matr.Nr. 03632725
geb. am 07.01.1993
wohnhaft in:
Hatzfelder Weg 23
81476 München

Lehrstuhl für
STEUERUNGS- und REGELUNGSTECHNIK
Technische Universität München

Univ.-Prof. Dr.-Ing./Univ. Tokio Martin Buss

Betreuer:	M.Sc. Cristian Axenie
Beginn:	07.10.2014
Zwischenbericht:	02.12.2014
Abgabe:	14.01.2015

Abstract

Sensor fusion is a biologically inspired concept, which enables a system with many small computation units to reach an agreement about a state of the system based on information exchange among these nodes.
In this projektpraktikum we worked with a network implementation based on this concept. We ported an existing prototype which was written in MATLAB to C, made it scalable and parallelized it with OpenMP.
The results showed an almost ten-fold performance increase for the sequential C implementation, while a naive OpenMP implementation proved to not be suitable, because each separate task is very small and the overhead for each task is too large to see any direct performance benefits from paralellization.

Contents

1 Introduction **5**
1.1 Problem Background 5
1.2 Problem Statement 7

2 Project Plan **9**
2.1 Tasks 9
2.2 Plan of Operation 9

3 Implementation **11**
3.1 Architecture 11
3.1.1 General Architecture And Paradigm 11
3.1.2 Synchronous and Asynchronous Update 11
3.2 Discussion of Results 12
3.2.1 Network Quality 12
3.2.2 Performance 12

4 Conclusion **15**

List of Figures **17**

Bibliography **19**

Chapter 1

Introduction

1.1 Problem Background

An essential component for motor planning and navigation for both real and artificial organisms is egomotion estimation. During navigation these systems build their spatial knowledge and behaviours by continuously refining their internal belief about the environment and own state. In order to frame the problem and motivate the proposed approach three main aspects must be tackled. Building up an internal representation of the environment and own state assumes a coherent alignment of the acquired sensory cues. As sensory cues are conveyed from both dynamic egomotion related signals (e.g. odometry, inertial signals) and static external environmental signals (e.g. visual, or auditory) the precise position of the self links and keeps the representation coherent and has a subsequent impact on behaviour. Egomotion estimation contributes to the understanding of high-level features of the environment like structure and layout such that the organism can direct actions and control its movement in the environment. In order to cope with the diversity of the environment, continuous and simultaneous incoming sensory data streams from different sensors must be combined into a robust representation of spatio-temporal properties of the environment. This representation should be informative and plausible such that it can disambiguate the scenario. This process, termed sensor fusion, is not trivial. It assumes that reference systems of the different sensory cues are aligned depending on inferred correlations while interference or conflicts between them are minimized.
Most current sensor fusion algorithms are based on probabilistic models of observations and processes. They use Bayes rule to integrate observations and system's model into an unified estimate of the system's state. Probabilistic methods replace point representations of perceptual estimates with probability distributions such that the statistics of the sensory estimates can be quantified and used for inference. In these terms Bayesian methods provide an optimal estimation scheme, in the sense that their estimate of the given state is unbiased (e.g. true on average) and has minimum variance. Hence, a narrow distribution will represent reliable data,

while a broader distribution will represent unreliable data. In Bayesian inference belief is progressively updated as new data from the sensors is fed in such that the initial belief evolves towards an informed posterior. In many simple cases is possible to describe the posterior distribution analytically. This mainly accounts for the cases in which the prior and likelihood function are given by Gaussian normal distributions. In many real-world scenarios though this is not the case and usually the expression of the posterior cannot be expressed analytically. Although various methods were developed to cope with approximating the posterior distribution there are some perceived limitations of probabilistic techniques. The first aspect is complexity, which emerges from computing with probabilities distributions, which in a real-time robotic system might not be adequate. A second aspect is inconsistency, as there are inherent difficulties involved in specifying a consistent set of beliefs in terms of probabilities and using these to obtain consistent deductions about states of interest. Finally, handling uncertainty about uncertainty, is a major problem as prior selection is not trivial and there is difficulty in assigning probability in the face of uncertainty, or ignorance about the source of information.

Introduction to the cortically inspired network for egomotion estimation

The proposed model is a network of processing units whose connectivity is provided by arbitrary functions (e.g. relations) defined between the units (e.g. given by physics of the sensors or the inter-sensor interactions). The relations between the units can be hand-designed or learned. In the current stage the relations are embedded in the network at design time but the network dynamics and integration capabilities are independent of the setup. We expect that in the next step the network learns the correlations between the sensory modalities during environment interaction and adapts the connectivity accordingly (e.g. plasticity). Using either hard-coded or learned relations each unit processes, stores and exchanges only local information. Furthermore, both feed-forward and feedback processing pathways are established between the units such that the mutual exchange of information between the units is kept to a consistent state. Network dynamics is based on a random update process using gradient descent, to the extent that values in each unit's map take small steps towards minimizing the mismatch with the relations in which that unit is involved. This way each unit balances the influence from all the other units and the external sen sor contribution. The amplitude of the update step is computed on-line, based on a confidence factor, which accounts as measure of reliability. The confidence factor is defined on a per map basis and it adapts proportionally with the mismatch between a source of information and network's belief. Using this mechanism the network is able to penalize strongly conflicting sources of information (e.g. fault tolerance) and enhance the contribution of consistent sources. Using these design principles we instantiated the proposed model for egomotion estimation scenario. [AC13]
The developed model was tested in a real environment using an autonomous omni-

directional mobile robot. In the basic scenario the robot moved in an uncluttered environment while an overhead camera tracking system kept track of its position and orientation.

1.2 Problem Statement

The initial software implementation of the network was a prototype written solely in MATLAB, which yielded unsatisfying performance as shown in Table 1.1. It can be observed, that a single run of the network for pre-cached sensor data of a single robot route takes more than an hour (5029.693 seconds) to be evaluated, while the actual movement of the robot took not more than a few minutes. Such network performance is completely unsatisfying for a real-time implementation of a sensor fusion concept.

Function Name	Calls	Total Time	Self Time*
self_motion_net_sim_edited	1	**5029.693 s**	4396.412 s
update_confidence_factor_translation	23769552	298.487 s	298.487 s
update_confidence_factor	17827164	207.869 s	207.869 s
clamp	58433482	116.476 s	116.476 s
linkaxes	9	4.661 s	3.164 s
title	14	1.898 s	1.891 s

Table 1.1: Excerpt from Profiler report for MATLAB implementation

We can observe numerous reasons why the network exhibits such poor performance:

- MATLAB is an interpreted language, which makes a network implementation in MATLAB inferior to any other network implementation in a compiled language such as C or Java.
- With help of the MATLAB Profiler tool we found that a lot of time is wasted on the execution of regular programming expression, for example, on switch-case statements with many cases.
- The program execution is sequential. This means, that, in the case of a multi-core machine, computing resources are not fully used.

The initial code of the network was not flexible due to it's non-modular character. This made it hard to reuse parts of the network in other setups. Also it was not possible to edit many settings, for example, when adding more sensors to the network, without a major rewrite of the code.

Chapter 2

Project Plan

2.1 Tasks

Due to the above mentioned reasons, our task was to completely rewrite the network to make the code execution and network evaluation much faster, to make the code easier to extend and maintain as well as to implement support for parallel processing.

The main task of the project was to implement the core of the network in C and make it possible to parallelize it using OpenMP. It would then be tested on a consumer multi-core machine. Since the updates of the individual nodes are independent of each other, this could give significant performance gains. The benefits would increase with the amounts of sensors used in the robot. With this approach, we can build a robot which is aware of it's own location in the environment without any external observer giving it any additional information. Once ported and evaluated, secondary tasks were the preparation for deployment on a 64-core SpiNNaker machine and the evaluation of different parallelization techniques.

2.2 Plan of Operation

We started by investigating the theory behind the sensor fusion networks in order to get a better understanding of the ideas behind the whole concept. Our main source of information was [AC13].

The first step practical was a direct port of the network into sequential C, with a Matlab MEX-interface for testing purposes. If the library acted as a drop-in replacement for the central network, we could be sure of its correctness. Starting with a sequential port also served as a way to get more familiar with the network, identify its bottlenecks and dynamics before refactoring it to be concurrent.

Step two was to be the parallelization and evaluation of the Network with OpenMP

on a multi-core PC. For this we had to develop a way to ensure atomic writes, while at the same time abstracting the till then hand coded node updates into invariant functions.

If the evaluation completed favorably, the last optional step would be the optional deployment on a 64-core SpiNNaker machine, with or without additional parallelization techniques.

Chapter 3

Implementation

3.1 Architecture

3.1.1 General Architecture And Paradigm

After multiple false starts, a suitable architecture for the library emerged. Since the network uses very few values, we decided to allocate all variables on the stack. To simplify debugging and make parallelization feasible, the state would be held in a set of structures representing the different nodes of the network. These would then be acted upon by evaluation and update functions. We are thus following a functional programming approach, as our functions only change our input values and do not influence the state of any other parts of the program.

3.1.2 Synchronous and Asynchronous Update

In order to make parallelization feasible, we created a list of "update_rule-value" pairs and looped over it, gathering the deltas and applying them in another for-loop after the calculation is done. This ensures that no node is overly influenced by the update order and the calculations are effectively synchronous. This is therefore from now on referred to as the *synchronous update*. The main benefit is that a list of N "rule-value" pairs can in theory be computed in one step by N processors, using a simple "parallel_for" statement from OpenMP. [RH11]

For evaluation and comparison purposes we also implemented the method used in the original implementation. Here we also create the list, but apply the deltas immediately. In order to ensure that no single rule gets influenced by its position in the list, we apply the updates in a random order. As long as the data sampling is done with a high enough sampling rate, this achieves a similar effect as a synchronous update.

3.2 Discussion of Results

3.2.1 Network Quality

As can be seen in Figure **??** the path of the robot as calculated by the network closely resembles the actual path of the robot which is portrayed in picture a in Figure **??**.

There remains a bug in the network which remains to be tracked down that causes the X component of the Maps as well as the offset to stay near 0. This leads to an imprecise scaling of the trajectory, whose shape, however, is as expected. Importantly, the network is stable and consistently produces the same result.

3.2.2 Performance

As can be seen in Table 3.1, the C implementation of the network is significantly faster than the MATLAB version (Table 1.1). It is done with the evaluation of the whole network in under 9 minutes, which is a major improvement when compared with a run time of more than an hour.
Since both network implementations got tested with the same data set, we can compare the run times directly and we get approximately a $\frac{5029.693}{522.452} \approx 9.63$-fold performance increase in the form of run-time reduction for the particular data set.

Function Name	Calls	Total Time	Self Time*
self_motion_net_sim_mex	1	**522.452 s**	254.550 s
num2cell	4951990	71.753 s	71.753 s
deal	4951990	32.856 s	32.856 s
mex_call (MEX-file)	495199	31.033 s	31.033 s
unpack_map1D_conf	990398	29.283 s	29.283 s

Table 3.1: Excerpt from Profiler report for C-implementation

It is noted, that the actual evaluation in the network takes somewhat less time than shown by the Profiler tool. Some amount of time is spent on the set up of the network, I/O reads etc. This, however, is mostly due to the fact, that we use pre-cached data, instead of evaluating the data stream live.

Suitability for Parallelization

We also performed analysis of the network, when parallel execution on multiple cores with the support of OpenMP is realized. Our performance results were worse than those of the serial execution. The explanation lays in the fact, that each step in the evaluation of the network requires a small amount of computational operations and

thus the overhead of initializing parallel instances outweighs any possible gains of parallel computation.

Real-Time Realization Capabilities

From the gathered data we can investigate the possibilities to use the created network for evaluation of a live, real-time data stream arriving from multiple sensors. A real-life application would, for example, be an autonomous robot, which tries to approximate it's current position as well as possible based solely on the noisy data it's sensors gather. To ensure proper functioning, we want the robot to know it's location at any given point in time almost immediately, that is, with as less time delay as possible.
From the MATLAB Profiler tool data in 3.1 we can see, that the core of the network (mex_call) gets executed in 31.033 seconds. Thanks to the knowledge of the underlying input data set, we know, that in this period of time the network has sampled the input data 495199 times. This means, that we manage to evaluate input data with a frequency of $\frac{495199}{31.033} \approx 15950$ input data sets per second or, conversely, that one whole evaluation takes roughly 0.06 ms. This is definitely more than enough for a real-time evaluation of the network. Since the input data itself does not considerably affect the execution time of the network, we argue that our performance results are valid for general input data and thus is usable for a real-time evaluation of a system.

Chapter 4

Conclusion

We managed to fulfill the main tasks of the project, that is, we managed to port the network from the slow MATLAB implementation to a much faster and easier to handle implementation in C. During this process we completely restructured the code so that the underlying data structures are easily extendable and changeable. We also modulated the code, so that parts of it can be reused in other projects with little additional programming effort.
Our implementation of the network successfully found the correct topology of the robot's path from the noisy sensor data. The performance of the network proved to be so good, that it enables us to consider using this network in real-time settings, for example, in microcontrollers on autonomous robots.

List of Figures

Bibliography

[AC13] Cristian Axenie and Jörg Conradt. Cortically inspired sensor fusion network for mobile robot heading estimation. In Artificial Neural Networks and Machine Learning–ICANN 2013, pages 240–247. Springer, 2013.

[CS02] Jörg Conradt, Pascal Simon, Michael Pescatore, and Paul FMJ Verschure. Saliency Maps Operating on Stereo Images Detect Landmarks and their Distance, Int. Conference on Artificial Neural Networks (ICANN2002), p. 795-800, Madrid, Spain, 2002.

[DL13] Christian Denk, Francesco Llobet-Blandino, Francisco Galluppi, Luis A. Plan, Steve Furber, and Jörg Conradt. Real-Time Interface Board for Closed-Loop Robotic Tasks on the SpiNNaker Neural Computing System, International Conf. on Artificial Neural Networks (ICANN), p. 467-74, Sofia, Bulgaria, 2013.

[RH11] Nicolas P Rougier and Axel Hutt. Synchronous and asynchronous evaluation of dynamic neural fields. Journal of Difference Equations and Applications, 17(8):1119–1133, 2011.

License