JOHANNES KEPLER
UNIVERSITY LINZ

Author
Markus Hiesmair

Submission
Institute of Telecooperation

Thesis Supervisor
Karin Anna Hummel

February 2018

ROBUST DRIVE-BY ROAD SIDE PARKING DETECTION ON MULTI-LANE STREETS USING AN OPTICAL DISTANCE SENSOR

Master's Thesis
to confer the academic degree of
Master of Science
in the Master's Program
Computer Science

**JOHANNES KEPLER
UNIVERSITY LINZ**
Altenberger Str. 69
4040 Linz, Austria
www.jku.at
DVR 0093696

Affidavit

I hereby declare that the following dissertation ”Put your thesis title here” has been written only by the undersigned and without any assistance from third parties.

Furthermore, I confirm that no sources have been used in the preparation of this thesis other than those indicated in the thesis itself.

Linz, on October 29, 2017

Markus Hiesmair

Acknowledgment

Summary

Summary . . .

asdf

Abstract

Abstract ...

Contents

To-Do: For now the TOC depth is 5 but will be reduced later

Abbreviations

CAB Compute Aggregate Broadcast (A computing model in parallel computing where computation is strictly partitioned in the three phases compute, aggregate and broadcast)

IaaS Infrastructure as a Service (Cloud computing service layer)

JSF Java Server Faces (Web technology in the arena of Java enterprise)

JSP Java Server Pages (Web technology available in Java Servlet containers)

MPI Message Passing Interface (Standard for implementing parallel algorithms on shared-nothing infrastructures)

OSN Online Social Network (An usually web-based online platform where friends, and acquaintances can connect and share information)

PaaS Platform as a Service (Cloud computing service layer)

SaaS Software as a Service (Cloud computing service layer)

UML Unified Modeling Language

URL Uniform Resource Locator

VM Virtual Machine

WAR Web Application Archive

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Importance of Research and Motivation

Currently the road side parking situation in most cities is rather untransparent. Except from parking garages and the like information about the availability of parking spaces is rarely available. However, finding parking spaces in urban areas can be a really difficult, frustrating and time consuming task for drivers. Furthermore, information about free parking spaces can help to reduce traffic by a tremendous amount. Studies have shown that in urban areas about 30% of traffic congestion is created by drivers looking for free parking spaces [3] and that in 2007 a loss of about $78 billion U.S. dollars was created by the use of about 2.9 billion gallons of gasoline alone in the USA [1]. Obviously this causes a lot of CO^2 emissions which are bad for the environment and furthermore about 4.9 billion hours were wasted by drivers while looking for parking spaces during that year.

1.2 Problem Definition

Detection of road side parking spaces and their states is a challenging task. Of course an obvious approach to the problem would be to put sensors to every parking space in the city, which check, if the corresponding parking space is occupied or vacant. This, however, has the drawback to be very expensive as, for big cities, thousands of sensors would have to be bought, installed and maintained. Furthermore, because the parking situation does not change often, the high frequency of sensing with such a system would be rather inefficient.

Another promising option to sense a city's parking situation is the use of mobile sensors instead of static ones. Crowd sensing has the advantage to be usually more cost effective and can provide sufficient accuracy. There has already been done some research on cars which analyze parking availability while they drive through the city. For instance, Mathur et al. [2] developed a system which uses distance information from the vehicle to the right side of the road to reason about parking spaces. Another approach, which was done by Zhou et al. [4] looks for car bumper shaped signal parts in the distance measurements to identify parking cars.

However, all of the mentioned mobile sensing approaches only work in single lane scenarios. Multi lane streets bring much more complexity in the recognition of parking spaces. There are many special cases which have to be addressed to work properly on multi lane roads. For example, the recognition of other driving vehicles and the recognition of the lane the sensing vehicle is on at the moment.

1.3 Detailed Approach and Goals

The overall aim of this thesis is to evaluate if it is possible to reach a sufficient high accuracy in road side parking detection on multi lane roads using a sensing vehicle which drives through the city and senses parking spaces while driving by. For the parking detection an optical distance sensor will be used to measure the distance to the nearest obstacle on the right side of the road (in many cases a parking car). This sensor will be mounted on the co-driver's side of the car and will continuously measure the distance while the car is driving. Furthermore, a GPS sensor will be used to include the spatial information of the vehicle. Using these measurements, the prototype should support accurate detection of free parking spaces in challenging road situations. Potential challenges for road-side parking detection are:

- multi-lane detection
- handling inaccuracies in GPS measurements
- differentiation of free parking spaces and other free spaces
- varying vehicle speed
- differentiation between perpendicular/parallel parking spaces

In a first step, after the sensors have been mounted on a car, test measurements will be collected while driving through some selected streets in Linz, Austria. The test scenes should include single lane- as well as multi lane streets and measurements in all streets should be done several times. To determine the ground truth of the parking availability a camera will be used, which takes pictures of the parking situation at the street during the tests.

After these measurements have been taken, the measurements should be analyzed, pre-processed, and then an algorithm should be developed (or learned) to classify the current parking situation. An important part of the algorithm will be the handling of multi lane roads, because there are many special cases which have to be considered. First of all, the lane in which the sensing vehicle is going has to be detected and has to be incorporated in the algorithm. Furthermore, the system should also detect when the car overtakes another driving car, because this could lead to falsely detected parking spaces.

In a final step, possible approaches should be evaluated, which can further enhance the results. For example, the cooperation of multiple sensing vehicles which are going at the same time at the same street can maybe help to increase the parking detection accuracy. Finally, the results of single- and multi lane detection should be evaluated and compared in terms of parking space count accuracy and parking occupancy rate accuracy.

Chapter 2

Related Work

Chapter 3

Reference Implementation of the Distributed Processing Framework

This is a chapter that references other chapters **??**

Chapter 4

Results and Discussion

Chapter 5

Conclusions and Future Work

Appendix A

Tooling

The real world implementation of this thesis available as open source software. The code, detailed instructions, and code samples can be found at `http://www.dynamograph.net/`. The following chapter gives a brief overview of the tools in use, lists their versions, and purpose they were used for.

A.1 Java, JavaScript, Web

As highlighted in chapter 3 the prototype implementation was mainly implemented with the Java programming language. There are basically two components worth mentioning in this respect. The first component is the computing cluster which is implemented as a standalone distributed Java application that can be deployed to possibly many worker nodes in a computer cluster. The second component is the web based user interface which is implemented as a Java Enterprise web application that runs in a Servlet container. The the following paragraphs and sections describe the tools and versions used in these components.

For software project management in general and automated dependency management the infamous Apache Maven 3.1.1 toolset is used. All components of the prototype are available as a Maven configured artifacts and can be automatically built and tested on state-of-the-art software integration services such as Jenkins `http://jenkins-ci.org`. Although continuous integration is currently performed from the developers machines since the project team was rather small and most of the time consisted only of the thesis author.

Although Java 8 was already available at the time of writing this the prototype was still compiled and tested with Java 1.7.0_45-b18 since first tests with Java 8 VMs have shown that some minor changes in object serialization could cause problems with the distributed compute cluster and the new features available in Java 8 were not used by the project anyway.

In the following list shows all Maven projects in alphabetical order and their purpose is briefly explained.

analytics: The analytics project is a helper project that contains Java and R code for general statistical analysis of datasets and is not available on production installations of the system.

census: This contains lists of firstname and lastname, and their frequency of occurance in the 1990 USA census [?]. The project contains a library that can be used to assign unique firstname, lastname pairs to vertices which is used to assign random names to make anonymized datasets. This process has proven to make anonymized datasets better readable.

commandline: This project contains command-line utilities that allow some control over a running temporal graph processing cluster. Mainly it can be used to properly shut down a cluster and to perform other maintenance tasks.

common: This project contains general data-structures to describe a temporal graph such as vertices and edges and also contains more general data-structures such as the implementation of a temporal map which is the basic foundation of most data components in the system.

configuration: As the name suggests this project is just a helper that allows to read configuration files from disk and provides the loaded configuration as static variables to the many other components of the system.

datasources: This project provides libraries that enable access to diverse data-sources such as static data-sets that hold temporal edge lists, relational databases, e-mail database parsers, and dynamic data-sources such as directly streaming data from Twitter.

distributed: This resembles the core of the distributed compute cluster and holds the code for all components such as the master node, the worker node, election process, graph partition table, and all the communication and multi-threading code. Implementation details of this project are described in greater detail in A.1.1.

dynamo-graph: This is an almost empty Maven parent project. It just holds a Maven project object model that can be used to build, test, and deploy all system components in one coherent build process.

frontend: Here the web based user interface is implemented. The project can be built to a standard Java EE web application contained in a WAR file and deployable to any standard compliant Servlet container as explained in more detail in A.1.3.

opencv: This project contains code that allows to reconstruct past social networks from image data. Such that user provided picture libraries can be imported and a temporal social network is extracted from them.

playground: The playground project is a potpourri of different code snippets that were used for testing. To the interested reader this project might be interesting because it shows how code can be run on top of the distributed computing framework. Especially interesting might be the parts that allow developers to run the system in so called local developer mode to debug distributed algorithms.

staticdata: This is a project that contains static data-sets used during testing and evaluating this thesis. The data-sets are mostly compiled from temporal edge lists. Each data-set available to the system also contains a configuration file that can be parsed by the frontend application and contains information like a data-set description, the color coding used and the edge types contained in the data-set.

uml: This project contains ObjectAid 1.1.6 UML class diagrams that are automatically synced with the code in the other projects. The UML diagrams are used for documentation purposes in this thesis and on the project website.

utils: Finally the utils project contains general purpose utilities that did not fit in with any other project but are used throughout many components of the project. Such utilities are a extended version of an URL resolver, parser for files encoded as comma separated values, and a tool that is capable of merging colors available as RGB color codes.

A.1.1 Distributed Java Applications

As already explained in great detail the distributed graph computing framework is implemented as a standalone Java application.

To-Do: ZooKeeper

A.1.2 Client API Methods

addEdge(String, Edge, long) addNode(String, Node) benchMarking() containsModel(String) createModel(String, Resolution) deleteModel(String) deleteVertex(String, long) deleteVertex(String, Node) executeAlgorithm(String, String) executeAlgorithm(String, String, SuperStepContext) executeAlgorithm(String, String, SuperStepContext, Timeframe) getMaximumTimeframe(String) getModelDescriptor(String) getVertex(String, long) getVertex(String, long, Timeframe) listModels() loadCode(String, byte[]) queryAlgorithmContext(long) queryAlgorithmState(long) queryExceptions(long) queryVertices(String, GraphQuery) unloadCode(String) updateVertex(String, Node) waitForAlgorithmState(long, ExecutionProfileState...) waitForCompletion(long)

A.1.3 Web-based User Interface

As explained in **??** users working with a system like the presented will most likely prefer to use web-based user interfaces. The high volume of data do be processed is just one argument for web-based access which practically allows the user to have the data completely stored in the cloud and just interface the processing from a web-browser. In the case of this project state-of-the-art web technology was used to build the user interface. The application is built as Java web application compliant with version 3.0 of the Servlet, version 2.2 of the JSP (Java Server Pages), and 2.2 of the EL (Expression Language) specifications. The visual rendering of the web pages is based on the JSF (Java Server Faces) 2.1.7 implementation provided by PrimeFaces 3.3.1 which guarantees that the user interface also complies with current standards for cross platform web-design.

For graph visualization the frontend relies on the JavaScript based graph drawing library SigmaJS 0.1 which was extended by a few simple modules which allow the user to move vertices in the graph via drag and drop, enable context information for each vertex such that computed scores for each vertex can be displayed, and the users are able to directly interact with vertices to drill deeper in a graph when necessary.

In general the web-based user interface can be built via Maven by running a build in the *frontend* project. The result is a web application archive (WAR) that can be deployed on any arbitrary Servlet 3.0 container. During the implementation and testing of this thesis mainly Apache Tomcat 7.0.29 and earlier was used and thus apache Tomcat is highly recommend for production environments also.

To-Do: add: jQuery

To-Do: add: Highcharts.js

A.2 Cloud Stacks

A.2.1 openNebula

A.2.2 openStack

A.2.3 Infrastructure Abstraction

A.2.3.1 Chef, Puppet, Dockr.io

Bibliography

[1] Texas Transportation Institute. Urban mobility report. 2007.

[2] Suhas Mathur, Tong Jin, Nikhil Kasturirangan, Janani Chandrasekaran, Wenzhi Xue, Marco Gruteser, and Wade Trappe. Parknet: Drive-by sensing of road-side parking statistics. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys '10, pages 123–136, New York, NY, USA, 2010. ACM.

[3] Sarfraz Nawaz, Christos Efstratiou, and Cecilia Mascolo. Parksense: A smartphone based sensing system for on-street parking. In *Proceedings of the 19th Annual International Conference on Mobile Computing & Networking*, MobiCom '13, pages 75–86, New York, NY, USA, 2013. ACM.

[4] J. Zhou, L. E. Navarro-Serment, and M. Hebert. Detection of parking spots using 2d range data. In *2012 15th International IEEE Conference on Intelligent Transportation Systems*, pages 1280–1287, Sept 2012.