Industrial Robot Collision Handling in Harsh Environments

Knut Berg Kaldestad

Industrial Robot Collision Handling in Harsh Environments

DOCTOR OF PHILOSOPHY AT THE FACULTY OF ENGINEERING AND SCIENCE, SPECIALIZATION IN MECHATRONICS

University of Agder

Faculty of Engineering and Science
2014

Doctoral Dissertation by the University of Agder 87
ISBN: 978-82-7117-768-3
ISSN: 1504-9272

Printed in the Printing Office, University of Agder
Kristiansand

Preface

The work described in this thesis has been carried out at the University of Agder and ABB Process Automation Division, Oslo during the period 2011 — 2014. The work has been conducted under the supervision of Professor Geir Hovland from University of Agder and Dr. David A. Anisi from ABB. In addition, four and a half months were spent on research abroad at the German Aerospace Center (DLR), Institute of Robotics and Mechatronics. My supervisor during this stay was Dr.-Ing. Sami Haddadin.

Acknowledgments

I would like to express my very great appreciation to my main supervisor Professor Geir Hovland for guiding me through this time as a PhD student. I am grateful for his supervision and serenity.

At ABB I will like to thank my second supervisor Dr. David A. Anisi for valuable guidance and input throughout the thesis. I also appreciate the effort of Dr. Charlotte Skourup for making the funding of the PhD through the Research Council of Norway possible.

A part of the PhD was spent abroad at the German Aerospace Center (DLR) Institute of Robotics and Mechatronics. This stay added valuable input to the time as a PhD student and I would like to thank Dr. Sami Haddadin for his guidance and for making the stay possible.

I would like to thank my all my good colleagues, it has been a pleasure to work with you. Form the office I would like to thank Ilya and Yulin. In particular I would like to thank Magnus and Øyvind for contributing to a stimulating work environment. I have never had this fun at work before.

The lab personnel at the university have been very helpful and provided valuable input. I would like to thank Eivind, Roy, Kalle and Yousef.

Finally, I am particularly grateful for the support from my family and friends.

Knut Berg Kaldestad
Grimstad, Norway
April 2014

Abstract

The focus in this thesis is on robot collision handling systems, mainly collision detection and collision avoidance for industrial robots operating in harsh environments (e.g. potentially explosive atmospheres found in the oil and gas sector). Collision detection should prevent the robot from colliding and therefore avoid a potential accident. Collision avoidance builds on the concept of collision detection and aims at enabling the robot to find a collision free path circumventing the obstacle and leading to the goal position.

The work has been done in collaboration with ABB Process Automation Division with focus on applications in oil and gas. One of the challenges in this work has been to contribute to safer use of industrial robots in potentially explosive environments. One of the main ideas is that a robot should be able to work together with a human as a robotic co-worker on for instance an oil rig. The robot should then perform heavy lifting and precision tasks, while the operator controls the steps of the operation through typically a hand-held interface. In such situations, when the human works alongside with the robot in potentially explosive environments, it is important that the robot has a way of handling collisions.

The work in this thesis presents solutions for collision detection in paper A, B and C, thereafter solutions for collision avoidance are presented in paper D and E. Paper A approaches the problem of collision avoidance comparing an expert system and a hidden markov model (HMM) approach. An industrial robot equipped with a laser scanner is used to gather environment data on arbitrary set of points in the work cell. The two methods are used to detect obstacles within the work cell and shows

a different set of strengths. The expert system shows an advantage in algorithm performance and the HMM method shows its strength in its ease of learning models of the environment. Paper B builds upon Paper A by incorporating a CAD model of the environment. The CAD model allows for a very fast setup of the expert system where no manual map creation is needed. The HMM can be trained based on the CAD model, which addresses the previous dependency on real sensor data for training purposes.

Paper C compares two different world-model representation techniques, namely octrees and point clouds using both a graphics processing unit (GPU) and a central processing unit (CPU). The GPU showed its strength for uncompressed point clouds and high resolution point cloud models. However, if the resolution gets low enough, the CPU starts to outperform the GPU. This shows that parallel problems containing large data sets are suitable for GPU processing, but smaller parallel problems are still handled better by the CPU.

In paper D, real-time collision avoidance is studied for a lightweight industrial robot using a development platform controller. A Microsoft Kinect sensor is used for capturing 3D depth data of the environment. The environment data is used together with an artificial potential fields method for generating virtual forces used for obstacle avoidance. The forces are projected onto the end-effector, preventing collision with the environment while moving towards the goal. Forces are also projected on to the elbow of the 7-Degree of freedom robot, which allows for null-space movement. The algorithms for manipulating the sensor data and calculating virtual forces were developed for the GPU, this resulted in fast algorithms and is the enabling factor for real-time collision avoidance.

Finally, paper E builds on the work in paper D by providing a framework for using the algorithms on a standard industrial controller and robot with minimal modifications. Further, algorithms were specifically developed for the robot controller to handle reactive movement. In addition, a full collision avoidance system for an end-user application which is very simple to implement is presented.

The work described in this thesis presents solutions for collision detection and

collision avoidance for safer use of robots. The work is also a step towards making businesses more competitive by enabling easy integration of collision handling for industrial robots.

Publications

The following four papers are appended and will be referred to by their Latin character. The papers are printed in their originally published form except for changes in format and minor errata.

Paper A. Kaldestad, K. B., Hovland, G. and Anisi, D. A. Obstacle Detection in an Unstructured Industrial Robotic System: Comparison of Hidden Markov Model and Expert System. *In 10$^{\text{th}}$ IFAC International Symposiums on Robot Control.* Dubrovnik, Croatia, 2012.

Paper B. Kaldestad, K. B., Hovland, G. and Anisi, D. A. CAD-Based Training of an Expert System and a Hidden Markov Model for Obstacle Detection in an Industrial Robot Environment. *In IFAC International Conference on Automatic Control in Offshore Oil and Gas Production.* Trondheim, Norway, 2012.

Paper C. Kaldestad, K. B., Hovland, G. and Anisi, D. A. 3 D Sensor-Based Obstacle Detection Comparing Octrees and Point clouds Using CUDA. *In Modeling, Identification and Control.* 33.4, 2012: 123-130.

Paper D. Kaldestad, K. B., Haddadin, S., Belder, R., Hovland, G. and Anisi, D. A. Collision Avoidance with Potential Fields Based on Parallel Processing of 3D-Point Cloud Data on the GPU. *In IEEE International Conference on Robotics and Automation (ICRA),* Hong Kong, China, 2014.

Paper E. (Submitted) Kaldestad, K. B., Hovland, G. and Anisi, D. A. Implementation of a Real-Time Collision Avoidance Method on a Standard Industrial Robot Controller. *Submitted to IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS),* Chicago, USA, 2014.

Contents

List of Figures

List of Tables

Abbreviations

d	Translation vector
R	Rotation matrix
T	Homogeneous transformation matrix
CPU	Central processing unit
CUDA	Compute Unified Device Architecture
DoF	Degree of Freedom
ES	Expert System
GPU	Graphics Processing Unit
IR	Infrared
RGB	Red, green, blue
SDK	Software development kit
TCP	Tool centre point
TCP/IP	Transmission Control Protocol/Internet Protocol

Markov Models

$\alpha_t(i)$	The forward variable

β The backward variable

$\delta_t(i)$ The single best state sequence

$\gamma_t(i)$ The probability of being in state i at time t where

λ The HMM parameters A, B and π

π Initial state distribution vector

$\psi_t(j)$ Index to the value which maximised $\delta_t(j)$

$\xi_t(i,j)$ The probability of being in state i at time t and making a transition to state j at time $t+1$

c Scaling variable

N Number of states

O Observation

P^* Probability of the most probable state sequence

q_t^* The most probable state sequence

q_t The actual state at time t

S State vector

v_k Observation symbol

x_t State at time t

A State transition matrix

B Observation symbol probability matrix

HMM Hidden Markov Model

Octree

lvs Number of levels

n_c Node centre

nl Node length at the current level

sl Side length of the smallest node

v_r Minimum required voxel resolution

Potential field

γ Attraction gain

κ Repulsive gain

τ_{goal} Threshold where the quadratic function changes to a conical function

τ_{obs} Repulsive field threshold distance

D Distance from the object to the obstacle

d Distance to the goal

F_{art} Artificial potential field

$U(x)$ Potential field

x object position

x_goal Object goal position

Chapter 1

Introduction

1.1 Motivation and Problem Statement

Safety for human and environment is one of the numerous challenges many of today's industries are facing. In competitive markets, being more cost efficient can sometimes come at the cost of safety for people and equipment. The oil and gas industry is one of the most safety aware sectors in Norway, and is currently looking at the possibilities for using industrial robots at petro-chemical plants and oil rigs.

Most of the easily accessible oil reservoirs are already explored and the trend goes towards exploring fields in remote locations, such as the Barents Sea. In these locations, the winter temperatures can range from -20^{o}C to -50^{o}C, and parts of the sea are covered by ice. Due to the harsh weather conditions in these areas the icy conditions can make them unreachable by vessels, but the long distances can also make travel by helicopter impossible. It is therefore of interest to study the possibilities of relocating people from oil rigs to onshore control rooms. From there they will be able to operate the oil rig or vessel remotely with the aid of remote operation systems and tools such as industrial robots.

Operations of large plants such as oil rigs are in themself challenging, but operations of such plants at remote locations even more so. Among these challenges is increased level of automation and safe use of industrial robots.

Many of today's industrial robots are regarded as reliable and have proven successful for solving many different tasks such as assembly, painting, palletising and

welding to name a few. It is however not enough to have robots that are only proven to be reliable, they will also have to cope with an unreliable environment. Examples can be unintentional relocation of a structure caused by a breakdown or intentional object relocation to a position which unintentionally is obstructing the robot path. If the environment blocks the robot path without a dedicated robot collision handling system in place, the robot will inevitably collide with the surrounding structure. Industrial robots may have an inbuilt collision awareness algorithm which stops the robot if there is an unintended interaction with the environment, but this is a safety feature that is first enabled after a collision. A collision with the environment can have unanticipated consequences and should be avoided, but in potentially explosive environments a collision is not an option as it can lead to sparks which in the worst case will ignite a fluid or a gas causing fire or explosion.

Systems developed to address the challenges above are not only limited to the oil and gas industry, but are also applicable for e.g. offshore wind parks. There is a trend towards remote locations for wind parks as well (Kaldellis & Kapsali, 2013). Further, systems for collision handling which are developed for industrial robots can also be adapted to service robots (Ferrein et al., 2013) for safer human-machine interaction.

The market for industrial robots in potentially explosive environments is currently small. Although this may change in the future, robots from the leading manufactures which are certified for potentially explosive environments do not currently have the capability of high payloads. The technology for creating such robots is definitely available, but the demand is currently not at a stage where it would make sense to mass-produce them. An example of a high payload robot aimed for use in potentially explosive environments is the Deckrobot 1500 from Robotic Drilling Systems, which is capable of lifting 1500 kg. The robots used in the work presented in this thesis are not certified, but the methods will of course still be applicable to such robots.

One of the main motivations of the thesis is the development and implementation of real algorithms on real robots. Standard industrial robots are used such that the path to realisation is not only restricted to universities or research entities. This type of solutions can benefit and motivate small and medium-sized enterprises

(SMEs) to invest in such systems. Consequently, this can contribute to make businesses more competitive and potentially prevent outsourcing of labour. The work presented in this thesis contributes to solutions for industrial robots working in harsh and potentially explosive environments in the not too distant future.

1.2 Outline

The first chapter gives an introduction to some of the topics used in the work described by this thesis. Chapter two gives a brief description of selected concepts and methods used in the papers. The methods are described in some detail, but the interested reader is referred to the literature for an in-depth knowledge on the selected topics. The third chapter presents concluding remarks and future work.

The second part consists of a collection of five contributed papers denoted paper A—E. In section 1.6 each paper is summarised in addition to providing the reader with background information and the research contribution.

1.3 Collision Handling

Collision detection and collision avoidance have been the main focus for the work in this thesis. To set the nomenclature, definitions of categories within collision handling with examples are given below.

1.3.1 Obstacle

An obstacle is an object[1] which approaches the robot in an unexpected way. For example, the object does not exist in the work cell CAD model. It blocks the manipulator's path such that the structure of the robot will physically be in contact with the obstacle if the path is followed. An obstacle can be static (non-moving such as a structure) or dynamic (moving, e.g. a human) and is defined as follows.

- An object that is not initially accounted for in the programming of the path.

[1] An object can also be an animal or a person

- An object that the robot is unintentionally approaching due to for example hardware or software failure.

1.3.2 Collision

A collision is the case where the manipulator physically is in contact with an obstacle. Intentional manipulation of objects is not defined as a collision.

1.3.3 Collision Awareness

Most industrial robots have incorporated a mechanism for sensing the force on the end-effector. If it is higher than expected the manipulator may respond by a stop action. More specifically, if the torques on one or more of the motors are higher than predicted plus a tolerated threshold, it is flagged as an error and appropriate action is taken. Hence, collision awareness means that a physical contact between the robot and the obstacle has occurred.

1.3.4 Collision Detection

Collision detection as defined in this thesis refers to the case where an obstacle is detected before collision occurs. In implementation, collision detection would raise a software flag and typically stop the robot. Collision detection is defined as one or more of the following:

- An obstacle enters the working volume of the object.
- The distance between the object and the obstacle is below a predefined threshold.
- The object is expected to collide with an obstacle based on e.g. speed and direction.

Collision detection as defined in this thesis does not necessarily mean that a physical contact between the robot and the obstacle has occurred.

1.3.5 Collision Avoidance

Collision avoidance occurs after collision detection and is the task of finding a new path which avoids collision with the obstacles blocking the path of the robot. If a collision free path exists, it should find a new way to the target point. Collision avoidance can be based on the following actions.

- Purely reactive action

  $(R1)$ The manipulator follows a re-planned path until an obstacle is detected and the collision avoidance behaviour is triggered. The obstacle avoidance behaviour is an input to the system and reactive motion is generated based on the obstacle position.

  $(R2)$ The manipulator movement is based on a virtual attraction to a target and on virtual repulsion from the obstacles.

  $(R3)$ Impulsive behaviour based on a set of rules.

- Planned path

  $(P1)$ The manipulator follows a pre-planned path until an obstacle is detected and the collision avoidance behaviour is triggered. The new obstacle is added to the model and new potentially target reaching collision free path is generated.

In the thesis approach number (R2) is used for collision avoidance.

1.3.6 Potentially Explosive Environments

All equipment operating in petro-chemical plants is exposed to an environment which can potentially be explosive, and must therefore follow strict regulations. Among these regulations is the ATEX directive (Leroux, 2007), describing how equipment and protection of these must be applied for use in potentially explosive atmospheres. This is of course a challenge for most equipment, including industrial robots which are not certified for potentially explosive environments by default. Actually, only a handful of different models, such as paint robots, are certified by the

European standard ATEX. The certification for operation in potentially explosive environments is achieved by purging, meaning that the robot is pressurised with air such that the inner pressure of the structure is higher than atmospheric pressure outside. This prevents gases entering the structure of the robot where it potentially can cause a reaction with the circuitry inside the robot and ignite.

Other challenges which target equipment are different conditions of the outdoor environment. Sunlight is a part of the outdoor environment which typically can render a sensor useless unless precautions are taken. Proper positioning of the sensor is one of the factors that can enhance the reliability of the data, and sensor redundancy can also be an alternative. Even sensors measuring the relevant conditions for proper functionality can be an option. One example is measurements of the energy within the operation band of the sensor, and if a threshold above this level is reached, the corresponding sensor data can be flagged unreliable. Fog, ice, rain and snow are also conditions which can lead to unreliable sensor data. Fog and ice are conditions which can be hard to deal with. Some applications can be dealt with by heating the glass of a casing where the sensor is mounted. For a potentially explosive environment, a heat element might not be a viable solution to make the sensor ATEX certified. The larger size of casing required for the heat element can hamper the application.

1.3.7 Sensors

The availability of rich sensor data of the environment, especially 3D data, was quite limited until recently. It has to be noted that this advancement in technology currently applies for sensors targeted for use in indoor environments. In November 2010 the Microsoft Kinect was released, initially as a human interface for interacting with their game console Xbox (Zhang, 2012). The interest in the Kinect sensor exploded, as it sold 10 million devices during the first three months after its launch, and sat a new Guinness World Record as the fastest selling device in the category for consumer electronics (Zhang, 2012). The device's ability to deliver 3D environment data at an affordable price, has led to development of Linux drivers and libraries for gathering data from the Kinect sensor. Microsoft has also launched its own software development kit (SDK) which is currently supported only for Windows. One of the

challenges with using this 3D sensor data for collision avoidance purposes is that it generates a lot of information. Rich information can lead to a better foundation for decision making in the algorithms, but on the other hand more calculations are needed in order to fulfil the task. On an algorithmic level this can be dealt with by reducing the resolution and filter away unnecessary data. On another level, the rate which the algorithm can execute at is related to the processor frequency or if it is a parallel problem, both the frequency and the number of cores will determine the execution time of the algorithm. Another trend is that sensors are getting cheaper, however there is no known off-the-shelf sensor delivering 3D data which is certified for oil and gas applications.

1.3.7.1 Evaluated sensors

(a) Industrial ultrasound sensor.

(b) Novelda impulse radar.

(c) SICK laser scanner.

(d) Microsoft Kinect sensor.

Figure 1.1: *Four of the evaluated sensors*

As a part of the work described in this thesis, numerous sensors were evaluated in the search for suitable sensors for both collision detection and collision avoid-

ance. Figure 1.1 shows some of these sensors and all of the above were actually tested.

Sensor 1.1a is an ultrasound sensor which sends out sound waves in a cone shaped volume. The sensor works by measuring the time between the sent and received signal and then using the speed of sound in air to calculate the distance to the perceived nearest object. The nearest object is determined by the sound wave which first returns to the sensor, but it is not necessarily the object that is closest to the sensor. Since the sensor is dependent on a returning sound signal, a surface with an angle may deflect the sound wave and therefore it may never return to the sensor and consequently the object would not be detected.

Sensor 1.1b is a radar which emits signals that are timed in a similar manner to the ultrasound sensor. This sensor, also measuring a volume, has some of the same weaknesses as the ultrasound sensor. The main weakness of these sensors is that it is not possible to determine where objects are located in the volume which means that they would not provide sufficient information for many applications.

The laser scanner 1.1c is a sensor that scans the plane each 0.5^{o} with a 270^{o}angular field of view. This sensor provides depth and angle information of each beam and as such it is possible to determine the location of each reflected surface in the scanned plane. In addition, compared to sensor 1.1a and 1.1b, this sensor is not that susceptible to surfaces at an angle since it is based on IR. Even though it gives more desirable results than the previous two sensors, if the laser beam hits e.g. a corner, the measurement might not be valid.

The Microsoft Kinect sensor 1.1d consist of many different sensors; accelerometer, colour camera, Infrared (IR) emitter, IR receiver and microphone. It is the IR emitter and IR receiver that enable the depth data to be produced. The IR emitter is located behind a lens producing a structured pattern which is reflected on a surface in the environment. The IR receiver captures the reflected pattern and the Kinect sensor creates depth values corresponding to pixels in the IR image. Further by using the horizontal and vertical view of the lens, spatial data of these points can be generated.

1.4 State of the Art

The work in this thesis is related to the oil and gas industry and not unlike other high-risk industries there is an evident gap between the state of the art in the industry and academia. A gap that is caused by what is available from an academic research perspective and what is actually implemented by industry. This gap can in some cases be a result of the higher requirements to proper and proven functionality and stricter regulations in industry. Another reason might be that not all state of the art solutions make sense to adopt from a business perspective. This especially applies to the oil and gas sector where an accident can cause large scale harm to humans, equipment and environment.

1.4.1 Collision Handling

1.4.1.1 Industrial Robots

Industrial robots are usually not equipped with any external sensors, these are often located on the fence around the work cell, and as such the standard robot would typically only incorporate collision awareness. Even though the robot may be limited to collision awareness, systems built around the robot can be responsible for handling collision detection. Typical industrial robot systems may consist of virtual lines (e.g. laser) or virtual walls (e.g. laser scanners and safety curtains). Currently, no known industrial robot manufacturer delivers robots equipped with sensors mounted on the robot for handling collision detection or collision avoidance as a standard. The solutions for collision handling are typically developed by a subsidiary of the robot company or delegated to the system integrators.

1.4.1.2 Development Platforms

The research on collision handling is a well-studied topic in robotics. For state of the art systems, it is common to use a depth sensor such as the Microsoft Kinect. 3D data of the environment is one of the key factors together with a proper algorithm for successfully achieving collision avoidance. The work by (Flacco et al., 2012) and (De Luca & Flacco, 2012) are examples where real-time collision avoidance

is made possible by reducing the representation of the robot, such that the amount of data are kept small. For the collision avoidance part, both end-effector and the whole manipulator body are taken into account. The collision avoidance algorithms are based on a modified artificial potential field approach and run on a multi-core central processing unit (CPU).

The work by (Haddadin et al., 2011) builds on the circular fields approach for collision avoidance. Compared to potential fields, the algorithm is less prone to get stuck in local minima, and therefore improves on one of the drawbacks with the traditional artificial potential fields approach. This work experimentally validates that the DLR lightweight robot (LWR) is able to avoid obstacles and find its way to the goal.

One of the main challenges with collision avoidance is that it can involve a lot of data representing the spatial information of the environment and the manipulator. In many cases large data sets require filtering, which can lead to a loss in level of detail. The loss in detail may not affect all types of applications, but when high detail is required the processing time goes up. As the processing time increases, the collision avoidance algorithm may not run in real-time anymore. Collision avoidance is a constant trade-off between speed and detail, therefore smart algorithms and fast hardware are key elements.

1.4.2 Robotic Co-Worker

The concept of a robotic co-worker is one of the considered factors for the work in this thesis. The concept depicted in figure 1.2 is based on the idea that field operator can perform heavy lift or precision tasks with the aid of the robot. The envisioned cooperation scenarios typically require a person to work in close vicinity of the robot. One of the approaches is a pre-programmed robot task, where the person uses a human interface device (HID) to start each part of the robot task. Figure 1.2 shows the robotic co-worker (operator) in a larger context. In many cases there can be an advantage to control the robot manually and access the environment information from a remote location to perform necessary operations. A remote operator can also assist the on-site operator at different levels, such as feedback based on visuals and assist to handle unforeseen events which require expert knowledge. The on-site

operator is not necessarily expected to program the robot, but rather run the program in a step-wise manner or move the robot to near points if necessary.

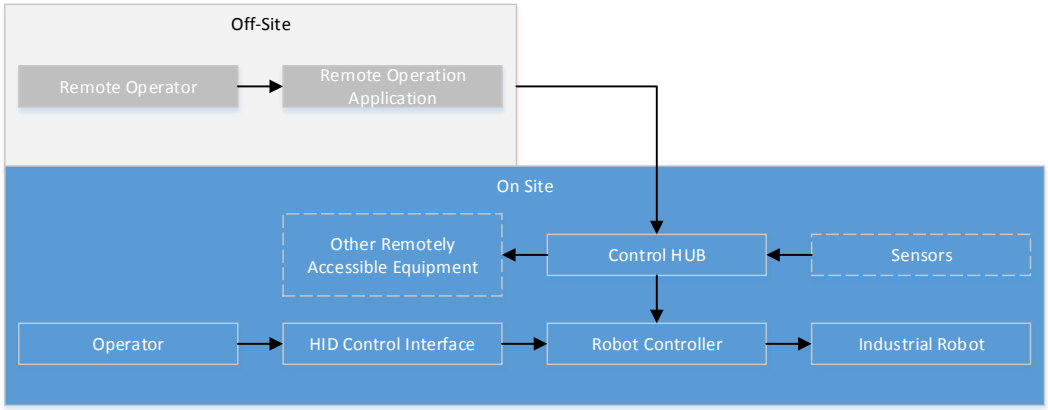


Figure 1.2: *A robot co-worker system*

This type of system can be ideal for normally unmanned platforms. Examples of fields developed with normally unmanned platforms are Arrol and Markham (The Netherlands) and North Morecambe (East Irish Sea) (Centrica Energy, 2010). There is also planned for a normally unmanned platform by Statoil at the Valemon field in the North Sea. Such platforms can run for about 6 to 7 weeks between each visit from maintenance personnel.

Fully unmanned platforms above sea-level are not yet a reality and are not considered realistic in the near future (unmanned subsea platforms do exist). Such platforms would require systems for maintenance and would as such depend on automated handling of these tasks. A scenario can be remotely controlled industrial robots changing parts that are built in a modular fashion. Since it can be almost unfeasible to have surveillance in all areas of a rig, these robots could be mounted on tracks performing regular inspections of equipment and structure. The robot would in that scenario be equipped with necessary equipment for inspection purposes such as camera and depth sensor.

1.4.2.1 Development Platforms

Development platforms are often low payload robots, and therefore fall outside the category of heavy lifting. The work by (Plasch et al., 2012) and (Pichler et al., 2013)

discuss heavy lift applications in a robot co-worker setting, but no real experiments are conducted.

There is more research conducted on the smaller platforms with low payload where many aspects have been taken into account. The research field of a robot co-worker can be divided into many categories such as collision handling, human robot interaction (HRI), physical human-robot interaction (pHRI), safe operations, sensoring, smart task scheduling and fulfilment. (Haddadin et al., 2011) is an example of work which addresses many of these challenges.

The typical co-worker research scenarios are bin-picking (Fuchs et al., 2010), where the robot should fulfil the task of pick and place from an unstructured bin. The worker should then be able to work side by side with the robot in a safe manner.

1.5 A short introduction to ABB industrial robots

ABB robots have been extensively used in the work described in this thesis. An introduction to these robots, their functionality and selected robots that have been a part of the experiments related to this thesis is presented.

Figure 1.3: *ABB robots. From the left, low capacity IRB 1600, high capacity IRB 6600 and paint robot IRB 5500. (Photo courtesy of ABB)*

1.5.1 Interface

The main hardware interface to the robot is the teach pendant, a HID with a touch screen typically used for jogging, path programming, program modification and program execution control. For interfacing with the robot there are different software approaches. First is the ABB Robot Studio which is a program for creating a robot system using standard ABB robots. Trajectories can be created and simulated for further uploading to the robot via network or a more manual approach such as USB transfer. The programs are programmed in the ABB propriety RAPID programming language. The second approach is using the PC SDK (for Microsoft Windows), which enables a developer to create programs for interacting and gathering information from the robot controller. There are some restrictions on what the SDK allows of program execution on the robot, except from starting and stopping programs already present in the robot controller. This requires a work-around which involves creating a RAPID program which checks for a change in the variables (done from PC SDK), and then acts accordingly. Third there is the PC Interface which allows socket communication with the controller. This type of communication does not limit the system which the external algorithm runs on. Socket communication can in practice be performed between a microcontroller and the robot. Devices running other operating systems than Microsoft Windows, such as Linux or Mac OS X would then not have the possibility to benefit from the features in the PC SDK, and are limited to communicate through e.g. a TCP/IP connection using the PC Interface.

1.5.2 Coordinate Frames

The robot coordinate frames are used extensively when programming the robot. The coordinate frames are related to each other (see arrows figure 1.4) and enable programming of the robot in a more intuitive manner. For example if a tool is replaced, it may be a millimetre longer than the previous, only the tool frame has to be adjusted. For a system not benefiting from the tool frame, all the programmed points would have to be modified such that new the tool would provide the same result as the previous tool. Another scenario is a robot program that is made offline,

e.g. using a simulated environment such as ABB RobotStudio. Simulations do not usually correspond one-to-one with the real robot work cell and as such, the frame which the path is programmed in will be adjusted to match the points to the work cell. Last, if the robot should manipulate an object, this can then be programmed in the object frame. Since there can exist multiple similar objects (see figure 1.4) the programming of the manipulation will only be done once, but only the position of the frame will change with regards to the user frame for the next object.

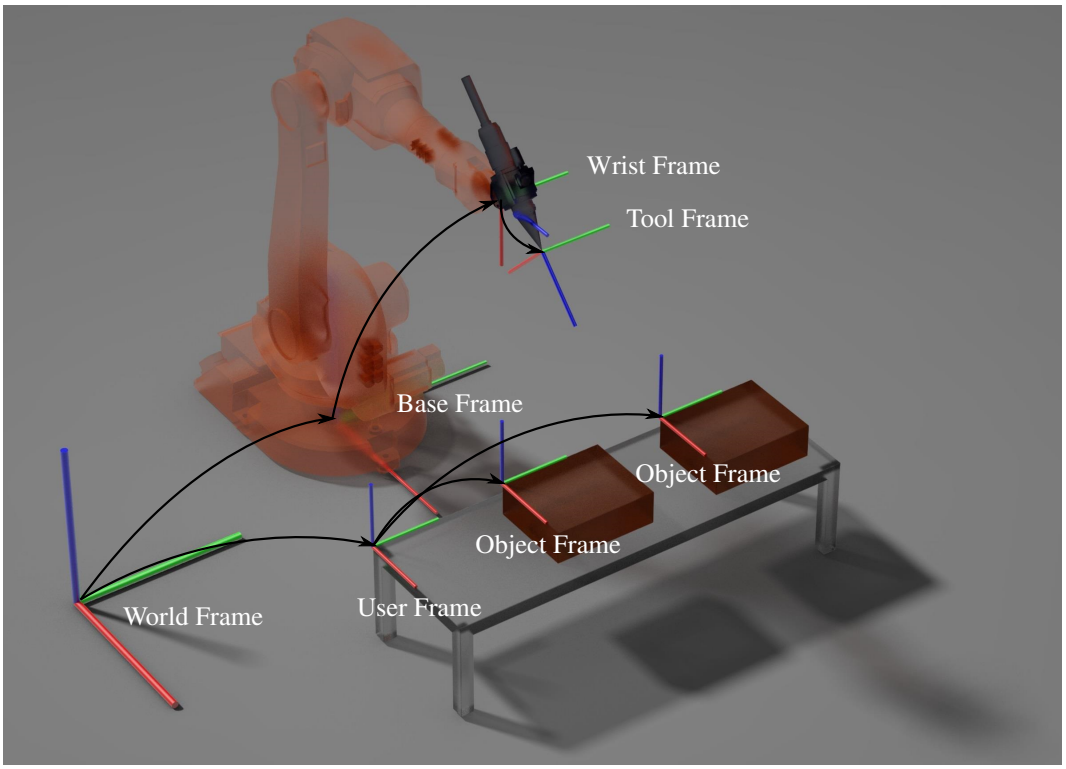


Figure 1.4: *Standard ABB Coordinate Frames. The red, green and blue axis correspond to x-, y- and z-axis respectively. The black arrows show how each frame is related to the other.*

The coordinate frames are at the heart of robotics. Each link of a serial robot is referenced to the previous with the aid of a transformation matrix, which describes the frame to frame relationship:

$$T = \left[\begin{array}{ccc|c} & R & & d \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{1.1}$$

where R is a 3x3 rotation matrix and d is a 3x1 translation vector. The 4x4 matrix T is called a homogeneous transformation.

The different frames are described by individual transformation matrices (T), where the frame for each link is a function of the current joint angles $\boldsymbol{\theta}$ or joint displacements for linear joints. This frame to frame relationship is described by the robot forward kinematics, by relating the joint variable to the position of one frame to another, e.g.

$$T_B^{EE} = T(\boldsymbol{\theta}, \boldsymbol{d}) \tag{1.2}$$

where T_B^{EE} maps the base frame B to the end-effector frame EE. $\boldsymbol{\theta}$ is the joint angle vector.

Determining the orientation and position of one coordinate frame with regards to another is key for describing concepts such as the pose of the end-effector with respect to the robot base. Coordinate frames are among the most documented topics in robotics, see for example (Spong et al., 2006), (Craig, 2005), (Siciliano & Khatib, 2008) and (Jazar, 2010).

1.6 Summary of papers

Paper A: Obstacle Detection in an Unstructured Industrial Robotic System: Comparison of Hidden Markov Model and Expert System

Summary: The experimental setup consists of an industrial robot following a pre-planned path. Both objects and obstacles can be placed inside the working area. The difference between an object and an obstacle is that the object is allowed at certain locations within the robot cell, everything else is regarded as obstacles. It is expected that objects typically are to be manipulated or interacted with. At arbitrary, but predefined locations along the pre-programmed path, the environment is scanned for obstacles. The 2D environment data of the robot cell is acquired by a SICK laser scanner mounted on the first robot link. The paper compares the Hidden Markov Model (HMM) approach with an Expert System (ES) for collision detection purposes. The HMM showed its strength by the ability to learn models of the

environment. The ES showed a calculation advantage, but showed a time disadvantage in the manual map creation.

Background and contribution: In a step towards dynamic collision detection the paper investigates two different methods. The choice of environment observer is important keeping in mind future implementation for real applications in an oil and gas environment.

The paper has been pulished as:

Kaldestad, K. B., Hovland, G., and Anisi, D. A. Obstacle Detection in an Unstructured Industrial Robotic System: Comparison of Hidden Markov Model and Expert System. *In 10th IFAC Intl. Symposiums on Robot Control.* Dubrovnik, Croatia, 2012a.

Paper B: CAD-Based Training of an Expert System and a Hidden Markov Model for Obstacle Detection in an Industrial Robot Environment

Summary: The paper presents two methods for real-time obstacle detection using a laser scanner and is an extension to paper A. The first method is an ES which uses the sensor data and compares it with a map of the environment. In addition the ES algorithm implements object edge filtering to remove false readings by the laser when the reading is near the edge of an object. The second algorithm comes in two versions, the first part generates the HMM $\lambda = (A, B, \pi)$ based on discrete positions, similar to the work in (Kaldestad et al., 2012). The sensor reading (O) in the current position is then scored against λ to find the best state sequence. The other advantage of the HMM method is that it can be directly trained from the map. Depending on the model, λ, the training can be quite time consuming, but training based on a map will allow for offline training which can be calculated when computing resources are available.

Background and contribution: The work conducted in paper A is extended in this paper in two ways. First a dynamic environment which should account for e.g. structural movement or humans is introduced. Also CAD-based training of the

HMMs which enables the system to do real-time obstacle detection is introduced.

The paper has been pulished as:

Kaldestad, K. B., Hovland, G., and Anisi, D. A. CAD-Based Training of an Expert System and a Hidden Markov Model for Obstacle Detection in an Industrial Robot Environment. *In IFAC Intl. Conf. on Automatic Control in Offshore Oil and Gas Production.* Trondheim, Norway, 2012.

Paper C: 3D Sensor-Based Obstacle Detection Comparing Octrees and Point Clouds Using CUDA

Summary: An industrial robot equipped with a Microsoft Kinect sensor on the fourth link serves as the environment observer for collision detection purposes. The Kinect sensor provides a point cloud of the environment and the robot is represented by points from a modified CAD model. By using the forward kinematics of the robot, the two point clouds are transformed to the same coordinate system. The distance from each point on the robot can now be determined for each point in the environment. Two methods are used for the representation of the points, an uncompressed point cloud and an octree representation. The paper shows that for larger amounts of parallel calculations, the GPU outperformed the CPU. By compressing the data using an octree representation, the speed of the GPU calculations increased until a certain limit. At this limit the resolution was drastically decreased and the CPU outperformed the GPU due to a small number of parallel calculations.

Background and contribution: Large amounts of data often allow for more detail, but on the other hand they require more processing. The Microsoft Kinect with libraries can provide at most 640x480 data points of the surroundings, each point represented by e.g. [X,Y,Z] in world coordinates of the environment. Large amounts of data which can be treated in parallel on the GPU can outperform the CPU in certain applications dependent on memory management and program flow. The paper demonstrated the potential for using CUDA in an industrial robot setting for treatment of sensor data.

The paper has been pulished as:

Kaldestad, K. B., Hovland, G., and Anisi, D. A. 3 D Sensor-Based Obstacle Detection Comparing Octrees and Point clouds Using CUDA. *In Modeling, Identification and Control.* 33.4 (2012c): 123-130.

Paper D: Collision Avoidance with Potential Fields Based on Parallel Processing of 3D-Point Cloud Data on the GPU

Summary: The work presents a solution for real-time collision avoidance using the potential field method with GPU calculations. The sensor used in this paper is a Microsoft Kinect which observes the robot and the environment that it operates in. The robot is filtered out of the observed environment such that it is not treated as an obstacle. Further, a model of each link of the robot is transformed to the coordinate system of the filtered environment. Then, for each point in parallel, the repulsive forces generated by the environment are calculated for each link. These forces are then sent to the robot controller and produces reactive movement by the robot tool centre point (TCP). In addition, forces were projected onto the 7-Degree of Freedom (DoF) robot elbow allowing for collision avoiding null-space movement.

Background and contribution: Collision avoidance has high computational requirements for complex models of the observed environment. It is therefore common to simplify both the obstacle and the avoiding object, e.g. reduce the robot to a stick model, and only use a few selected points of the environment. To the authors' knowledge this is the first time that collision avoidance is demonstrated in real-time on a real robot using parallel GPU processing.

The paper has been accepted as:

Kaldestad, K. B., Haddadin, S., Belder, R., Hovland, G., and Anisi, D. A. Collision Avoidance with Potential Fields Based on Parallel Processing of 3D-Point Cloud Data on the GPU. *In IEEE International Conference on Robotics and Automation,* Hong Kong, China, 2014.

Paper E: Implementation of a Real-Time Collision Avoidance Method on a Standard Industrial Robot Controller

Summary: The paper presents a collision avoidance solution for an industrial robot using a standard robot controller. The collision avoidance algorithms are executed on a separate computer equipped with a NVIDIA GPU and the environment data is provided by a Microsoft Kinect. The separate computer communicates with the robot controller over a TCP/IP connection, providing the robot with virtual link forces. The algorithms for the robot controller are made for easy implementation of collision avoidance, in fact only the function name has to be changed in order to enable this ability. The algorithms and the system functionality were demonstrated by experiments on an industrial robot.

Unpublished work related to this paper is attached as an appendix. The work includes an outdoor test and expands on the algorithms related to the implementation of collision avoidance on industrial robots.

Background and contribution: The industry requires solutions that are cost effective and fast to implement. The paper is an extension to paper D and contributes with an additional solution for collision avoidance which can be implemented quickly by an end user without the need for a special robot controller. It demonstrates how the standard framework for the ABB robot can be utilised with minor modifications to program code to achieve collision avoidance. The ease of implementation is a motivating factor for companies investing in new equipment.

The paper has been submitted as:

> Kaldestad, K. B., Hovland, G., and Anisi, D. A. Collision Avoidance with Potential Fields Based on Parallel Processing of 3D-Point Cloud Data on the GPU. *In 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems* Chicago, USA, 2014.

1.7 Contributions

The work presented in this thesis has an industrial focus. The goal of the work has been to develop and implement methods, algorithms, software and hardware which could potentially be used in industrial practice in the not too distant future. The main contributions are summarised below:

- The thesis contains a comparison of an expert system and a hidden markov model for collision detection in an industrial robot system. Such a comparison has not been previously performed.
- The thesis contains a contribution on CAD-based training of a Hidden Markov Model for collision detection in an industrial robot system.
- Another contribution is the comparison of computational performance of two additional methods: octrees and point clouds on a GPU for collision detection.
- Real-time collision avoidance using GPU processing is presented for the first time on an industrial robot on a development control platform.
- Real-time collision avoidance using GPU processing is presented for the first time on a standard industrial robot and controller.
- A solution for easy integration of collision avoidance using a standard industrial robot programming language. The presented algorithms and the required hardware are both quick to implement and set up for an end-user.

Chapter 2

Research Methodology

This chapter gives an introduction to some of the concepts used in robotics as well as some specifics to the ABB industrial robots. Topics which are considered common knowledge in robotics are covered in less detail in this chapter and a referenced instead.

2.1 Expert System

The expert system (ES) is a concept which tries to simulate the expert knowledge such as judgement and behaviour of an operator. Because expert knowledge is required, the literature may contain little or no information on the problem at hand. A scenario for an expert system is the situation where something is about to have an undesirable outcome such as a robot which is about to collide. A typical operator reaction would be to stop the operation by e.g. a push on the emergency stop button. In a similar scenario for an ES algorithm, the input will be some information about the environment. An algorithm will then validate the data and make a decision which can be based on an undesirable change to the environment. Such change can result from a variation in the data exceeding a threshold or a geometrical comparison by using a model of the environment. Different output scenarios are available based on the complexity of the algorithm e.g. different severity levels. If an operator is part of the loop, the algorithm can present a warning based on some criteria, or if the situation is critical, the algorithm can directly halt the system. See for example

(Jackson, 1990) for more information on ES.

2.2 Markov Models

Markov models are named after the Russian mathematician Andrey Markov. He introduced what is known today as the Markov property which describes a memoryless stochastic process, only dependent on the current state.

2.2.1 Markov Chain

The Markov chain is a type of Markov model that undergoes state transitions. Transition to the next state is only dependent on the current state, because it is assumed to possess the Markov property.

The following example is given for clarification purposes. Equation (2.1) shows a Markov chain with three weather states and their state transition probabilities. As seen in the figure, if it is a sunny day it is 31% probability that the next day will be sunny and 54% for cloudy and 15% for a rainy next day. The full transition matrix is written as,

$$A = \begin{array}{c} \\ \textit{Rainy} \\ \textit{Cloudy} \\ \textit{Sunny} \end{array} \begin{array}{c} \begin{array}{ccc} \textit{Rainy} & \textit{Cloudy} & \textit{Sunny} \end{array} \\ \begin{bmatrix} 0.2805 & 0.5610 & 0.1585 \\ 0.2170 & 0.5566 & 0.2264 \\ 0.1461 & 0.5393 & 0.3146 \end{bmatrix} \end{array} \tag{2.1}$$

where this example contains $N = 3$ distinctive states. The state transition matrix A can be used to find the steady state probabilities,

$$\lim_{n\to\infty} A^n = \begin{array}{c} \\ \textit{Rainy} \\ \textit{Cloudy} \\ \textit{Sunny} \end{array} \begin{array}{c} \begin{array}{ccc} \textit{Rainy} & \textit{Cloudy} & \textit{Sunny} \end{array} \\ \begin{bmatrix} 0.2141 & 0.5535 & 0.2324 \\ 0.2141 & 0.5535 & 0.2324 \\ 0.2141 & 0.5535 & 0.2324 \end{bmatrix} \end{array} \tag{2.2}$$

which describes the distribution of the weather in the long run. In this example, the probability for one of the three weather types at any given day will be 55.5% for

cloudy, 23% for sunny and 21.5% chance for rainy.
A more useful result can be obtained by considering the initial state. The state vector would provide information about the weather the current day. The probability for a particular weather in three days if it is cloudy today would be given by,

$$x\,A^{days} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0.2155 & 0.5538 & 0.2308 \\ 0.2142 & 0.5535 & 0.2323 \\ 0.2126 & 0.5533 & 0.2341 \end{bmatrix}^3 = \begin{bmatrix} 0.2155 & 0.5538 & 0.2308 \end{bmatrix} \tag{2.3}$$

where x is the state vector. The result shows that it is over 55% chance that the weather will be cloudy in three days.

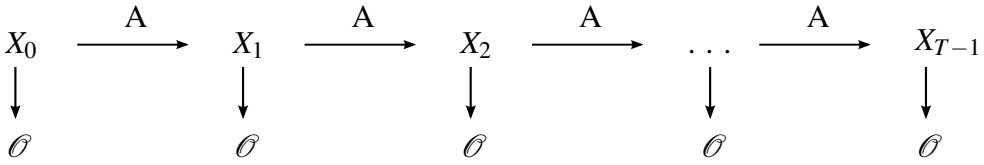


Figure 2.1: *Markov Chain.*

Figure 2.1 shows how the state is changing by applying the state transition matrix A. It can also be seen that the states, X_t, are directly observable ($\mathcal{O}$). (Ching & Ng, 2006) can provide more information on the subject.

2.2.2 Hidden Markov Models

The Hidden Markov Models (HMMs) introduce a new layer to the Markov chain. As can be seen in figure 2.2, the states x_t are not directly observable anymore, therefore the name hidden. The observer can now only observe a symbol v_k, which is related to the state $q_t$at time t by a observation symbol probability matrix B. To calculate the probability of a state or a state sequence, the A and B matrices must be known. Additionally, the initial state distribution vector π must also be known in order to determine the initial state.

One of the practical applications for HMMs is speech recognition. In speech, each particular word can be divided into different combination of letters or more precise states X_t. For the word potato, the states could be [”po”, ”ta”, ”to”] or e.g.

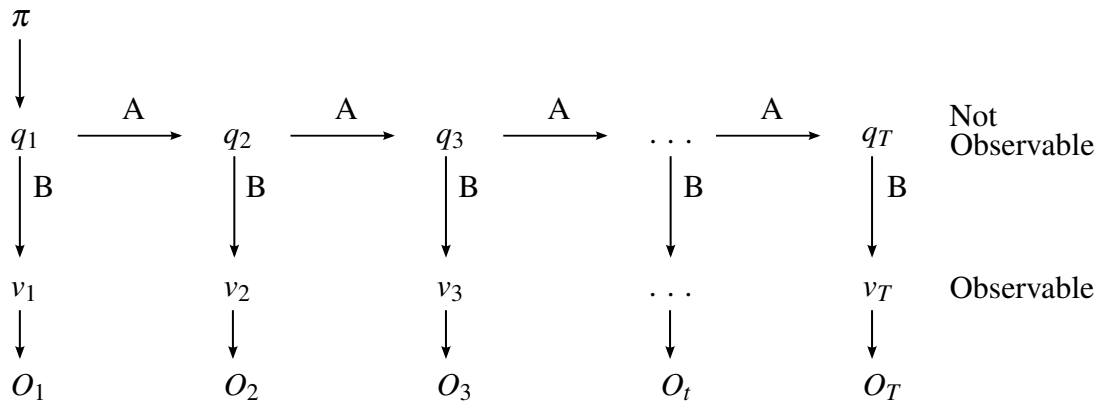


Figure 2.2: *Hidden Markov Model, where A, B and π represents the parameters for one particular model λ.*

["po", "tai", "to"]. Depending on the pronunciation of the word, the observation sequence of the same word can be different. The actual observation O is a part of a sound signal, such as a feature vector, and B would then be the matrix giving the probability of observing this symbol in a given state. The observation vector $\boldsymbol{O}$ and the state vector $\boldsymbol{S}$ are denoted as,

$$\boldsymbol{O} = O_1, O_2, O_3 ... O_T, \ \boldsymbol{S} = x_1, x_2, x_3 ... x_N$$

where $t \in \{1, 2, \cdots, T\}$ and T is the number of observation time steps. N is the number of states in the HMM. The definition of the state transition matrix is,

$$A = a_{ij} = P(q_{t+1} = x_j | q_t = x_i),$$

where q_t is the probability for being in a particular state S at time t where $\sum_{j=1}^{N} a_{ij} = 1$. The emission matrix B which relates an observation v_k to all the states S is given by,

$$b_j(k) = P(v_k = O_t | q_t = x_j), \tag{2.4}$$

where v_k is a particular observation O at time t. Expanding on the weather example in the previous section, the different observations v_k can be done by observing a barometer. In this particular case an observation v_k will be the air pressure or rather a range of air pressures,

$$B = \begin{array}{c} \\ \textit{Rainy} \\ \textit{Cloudy} \\ \textit{Sunny} \end{array} \begin{array}{c} \begin{array}{cccc} p_1\text{: } p < 1000 & p_2\text{: } 1000 \geq p < 1010 & p_3\text{: } 1010 \geq p < 1020 & p_4\text{: } p \geq 1020 \end{array} \\ \begin{bmatrix} 0.6585 & 0.3171 & 0.0244 & 0 \\ 0.4340 & 0.3868 & 0.1651 & 0.0142 \\ 0.2889 & 0.5000 & 0.1667 & 0.0444 \end{bmatrix} \end{array} \tag{2.5}$$

Given that the state $q_t = Cloudy$ and the observation $v_k = p < 1000$, this yields $q_t = 2$ and $v_k = 1$ which gives the probability for the observation symbol $b_{21} = 0.434$ on a cloudy day.

Finally, the initial state matrix π is given by

$$\sum_{i=1}^{N} \pi_i = 1,$$

where π_i is the probability of initially being in state x_j at time 0. In this example π is given by,

$$\pi = \begin{array}{c} \begin{array}{ccc} \textit{Rainy} & \textit{Cloudy} & \textit{Sunny} \end{array} \\ \begin{bmatrix} 0.2135 & 0.5521 & 0.2344 \end{bmatrix} \end{array} \tag{2.6}$$

All the parameters in the HMM are introduced (see figure 2.2), but three main problems still remain to be examined.

2.2.2.1 Three HMM problems

First, calculate $P(O|\lambda)$

The model $\lambda = (A, B, \pi)$ is known, calculate the probability of the observation sequence $\boldsymbol{O} = O_1, O_2, ..., O_T$.

For example what is the probability of observing the air pressures $\boldsymbol{O} = [v_k] =$

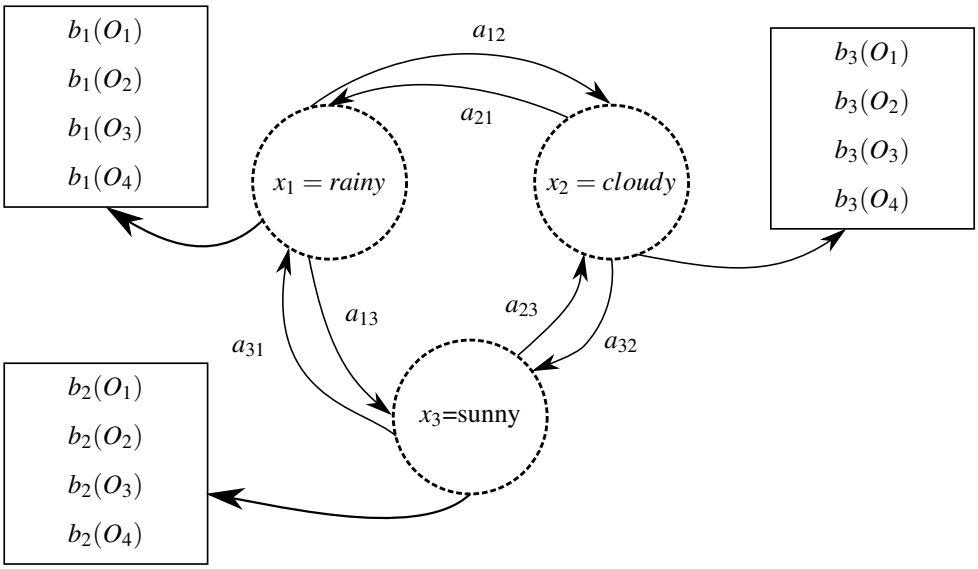


Figure 2.3: *HMM transformation and emission.*

$[p_1, p_3, p_3, p_2]$

Second, find the state sequence corresponding to the observations

The model λ and the observations are known. How can the hidden state sequence $Q = q_1, q_2, ..., q_T$ be determined or "uncovered"?

For example given $\boldsymbol{O} = [p_1, p_3, p_3, p_2]$ and λ, what are the most probable weather conditions.

Third, maximise the $\lambda = (A, B, \pi)$

How to build the model $\lambda = (A, B, \pi)$ and maximise $P(O|\lambda)$ given the data of the states and the corresponding observations?

For example how to create (A, B, π) from a dataset?

These three sub problems will be considered in the following separate sections.

2.2.2.2 Calculate $P(O|\lambda)$

The probability of the observation sequence given a HMM model $\lambda = (A, B, \pi)$ is,

$$P(O|\lambda) = \sum_{\forall Q} P(O|Q, \lambda) P(Q|\lambda)$$

$$= \sum_{i=1}^{N} \alpha_T(i), \tag{2.7}$$

where and $\alpha_t(i)$, $t \in \{1, \cdots, T\}$ is called the forward probability variable. The forward variable $\alpha_t(i)$ is the probability of the partial observation sequence until time t.

Initialization:

$$\alpha_1(i) = \pi_i b_i(O_1) \qquad 1 \leq i \leq N$$

$$c(1) = \sum_{i=1}^{N} \alpha_1(i) \qquad 1 \leq i \leq N$$

$$\hat{\alpha}_1(i) = \alpha_1(i)/c(1) \qquad 1 \leq i \leq N$$

where $c(t)$ is a scaling factor preventing the dynamic range of $\alpha_t(i)$ to exceed the computer's floating point precision.

Induction:

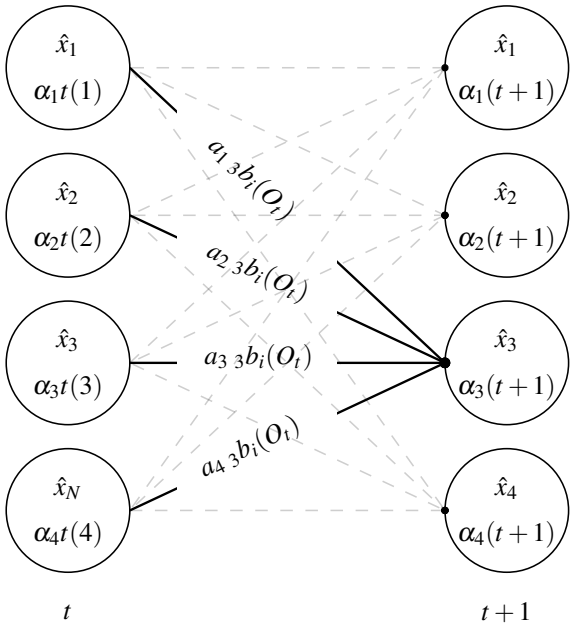


Figure 2.4: *A trellis graph showing the computation of* $\alpha_t(i)$.

$$\alpha_t(i) = \left[\sum_{j=1}^{N} \hat{\alpha}_{t-1}(j) a_{ji} \right] b_i(O_t) \qquad 1 \leq j \leq N$$
$$2 \leq t \leq T$$

which can be written in the intuitively vectorised form,

$$\boldsymbol{\alpha}_t = (\boldsymbol{\alpha}_{t-1}\boldsymbol{A}) \odot \boldsymbol{b}(O_t)^T \qquad 2 \leq t \leq T$$

where $\odot$ is the element-wise product. Figure 2.4 shows the induction step of the algorithm.

$$c(t) = \sum_{i=1}^{N} \alpha_t(i) \qquad 2 \leq t \leq T$$

$$\hat{\alpha}_t(i) = \alpha_t(i)/c(t) \qquad 1 \leq i \leq N$$
$$2 \leq t \leq T$$

Termination:

$$P(O|\lambda) = \sum_{j=1}^{N} \hat{\alpha}_T(j)$$

2.2.2.3 Decode, find the state sequence corresponding to the observations

The state sequence corresponding to the observations can be defined by,

$$\delta_t(i) = max\ P(q_1 q_2 \cdots q_t = x_i, O_1, O_2 \cdots O_t|\lambda)$$

which will give the probability of the most probable path given the observation sequence. This algorithm is known as the Viterbi algorithm and is as follows:

Initialization:

$$\delta_1(i) = \pi_i b_i \qquad 1 \leq i \leq N$$
$$\psi_1(i) = 0 \qquad 1 \leq i \leq N$$

Recursion:

$$\delta_t(j) = \max_{1 \leq i \leq N} [\delta_{t-1}(i) a_{ij}] b_j(O_t) \qquad 2 \leq j \leq N$$
$$\psi_t(j) = \arg \max_{1 \leq i \leq N} [\delta_{t-1}(i) a_{ij}] \qquad 2 \leq j \leq N$$

The recursion algorithm is similar to the calculation of α, but the calculation of $\delta_t(j)$ is only concerned with which of the N state transitions i that gives the highest probability for the next state j. The variable $\psi_t(j)$ stores the index corresponding to the value i which maximised $\delta_t(j)$.

Termination:

$$P^* = \max_{1 \leq i \leq N} [\delta_T(i)]$$
$$q_T^* = \arg \max_{1 \leq i \leq N} [\delta_T(i)]$$

P^* is the probability of the most probable state sequence and q_T^* is the index i in $\delta_T(i)$ which contains the value P^*.

Optimal state sequence backtracking:

$$q_t^* = \psi_{t-1}(q_{t+1}^*) \qquad 1 \leq t \leq T-1$$

When the probability of the most probable state sequence is determined at time T, it is possible to find the sequence of states that created this result. Since all the indices

(which correspond to the state transitions) that are leading to the maximum value of $\delta_t(j)$ are stored in $\psi_{t-1}(j)$, it is possible to go backwards from the known index q_T^*. q_t^* is the set of states which will result in the probability P^* given the model λ.

2.2.2.4 Generate the model $\lambda = (A, B, \pi)$

The model $\lambda = (A, B, \pi)$ can be generated from both supervised and unsupervised learning. For supervised learning the states and the observations with regards to time must be present in the data. For the unsupervised learning it is sufficient that the data only contain the observations, but a re-estimation algorithm such as the Baum-Welch algorithm is needed to find a solution for the model. The following sections present supervised model generation.

Supervised learning:

$$a_{ij} = \frac{\text{count}(\text{t}_\text{i}, \text{t}_\text{j})}{\text{count}(\text{t}_\text{i})} \tag{2.8}$$

where $\text{count}(\text{t}_\text{i}, \text{t}_\text{j})$ is the number of times the data contains a transition from state i to j.

$$b_i(k) = \frac{\text{count}(\text{v}_\text{k}, \text{t}_\text{j})}{\text{count}(\text{t}_\text{i})} \tag{2.9}$$

where $\text{count}(\text{v}_\text{k}, \text{t}_\text{j})$ is the number of times the data contains the observation v_k when making the transition to state j. Counting the number of times each observation in the data appear at time t_1 and calculate the initial distribution π_i.

$$\pi_i = \frac{\text{count}(\text{q}_1 = \text{t}_\text{i})}{\text{count}(\text{q}_1)} \tag{2.10}$$

Unsupervised learning

Since it is not possible in unsupervised learning to count transitions and the number of times in a state given the observation another definition for the variables a_{ij}, $b_i(k)$, π_i is needed. The Baum-Welch algorithm, presented in (Rabiner, 1989), is an expectation maximisation algorithm for re-estimating the model λ.

Baum-Welch Algorithm

The backward variable β is similar to the forward variable α, but it stores the probability from time T calculated backwards. $\beta_t(i)$ is the probability of the partial observation sequence from $t+1$ to time T.

Initialization:

$$\begin{aligned} \beta_T(i) &= 1 && 1 \leq i \leq N \\ \hat{\beta}_T(i) &= 1 && 1 \leq i \leq N \end{aligned}$$

Induction:

For time $t = T-1, T-2, \cdots, 1$

$$\begin{aligned} \beta_t(i) &= \sum_{j=1}^{N} a_{ij}\hat{\beta}_{t+1}(j)b_j(O_{t+1}) && 1 \leq i \leq N \\ \hat{\beta}_t(i) &= \beta_t(i)/c(t+1) && 1 \leq i \leq N \end{aligned} \tag{2.11}$$

For the purpose of re-estimation, two new variables are introduced, $\gamma_t(i)$ and $\xi_t(i,j)$ which are both described below. Given an observation sequence, $\gamma_t(i)$ is the probability of being in state i at time t where,

$$\begin{aligned} \gamma_t(i) &= \hat{\alpha}_t(i)\hat{\beta}_t(i) && 1 \leq i \leq N \\ & && 1 \leq t \leq T \end{aligned}$$

(2.12)

The second variable required for re-estimation is $\xi_t(i,j)$. It is defined as the probability of being in state i at time t and making a transition to state j at time $t+1$:

$$\xi_t(i,j) = \frac{\hat{\alpha}_t(i)a_{ij}b_j(O_{t+1})\hat{\beta}_{t+1}(j)}{c(t+1)} \qquad \begin{array}{l} 1 \leq i \leq N \\ 1 \leq j \leq N \\ 1 \leq t \leq T \end{array}$$

Finally a re-estimation method of the model λ can be introduced.

Re-estimation of state prior density:

$$\begin{array}{ll} \bar{\pi}_i = \gamma_1(i) & 1 \leq i \leq N \\ \bar{a}_{ij} = \dfrac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} & 1 \leq i \leq N \\ & 1 \leq j \leq N \\ \bar{b}_{ij} = \dfrac{\sum_{t=1}^{T} f_{O_t}\gamma_t(i)}{\sum_{t=1}^{T} \gamma_t(i)} & \end{array} \tag{2.13}$$

where f_{O_t} is defined as,

$$f_{O_t} = \begin{cases} 1, & \text{if } O_t = v_k \\ 0, & \text{otherwise} \end{cases} \tag{2.14}$$

For further knowledge on HMMs, the following references are recommended (Rabiner, 1989), (Fink, 2008) and (Yang, 2010)

2.3 Octrees

An octree is a type of data suitable for representing 3D environment which, if visually interpreted, is structured into a set of nested cubes. As can be seen from figure 2.5, representation of the two voxels 5.4 and 5.6 does not require the creation of more nodes than needed. In this context, the voxels are the nodes at the lowest level of the octree. The octree will be designed with as many levels as needed to achieve the desired resolution and is then well suited for representing 3D data. What distinguishes the octree from an unstructured list of points, is exactly that it is structured by nature. This is very efficient when checking whether a volume is occupied, e.g., node 7.1 which is not present in figure 2.5. This would be done by checking node 7 for children, and if it has children, check if child 1 exists. This would amount to only to checks being performed for the octree even though it can contain a large number of children. Generally the octree algorithm is expected to have a complexity of O(log N).

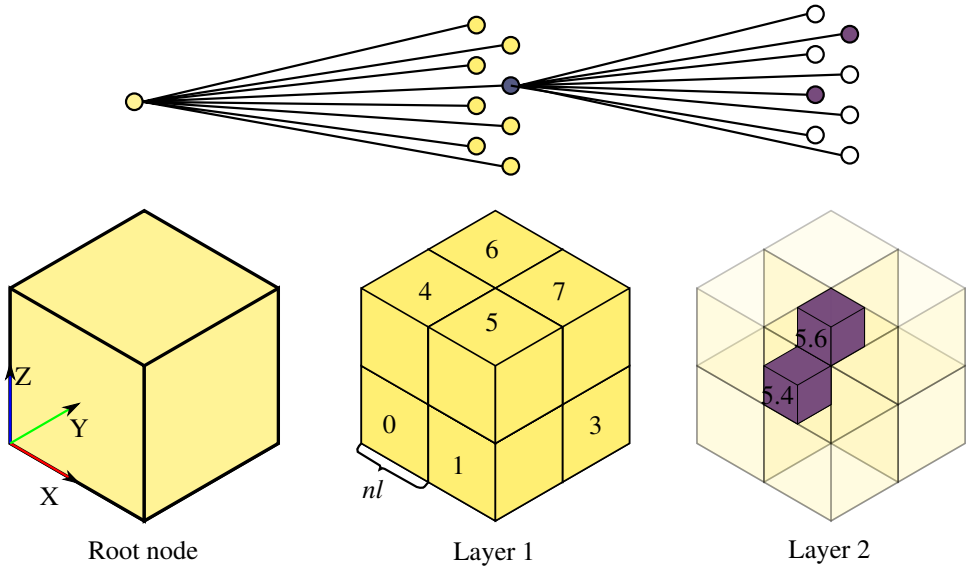


Figure 2.5: *Octree.*

2.3.1 Construction and Searching

The octree is created by first determining the resolution of the voxels and the minimum side length for the volume which is considered. For example the task is to store data from a sensor with a given volume of side lengths in mm,
$V = [10000, 20000, 8000]$. The resolution of the smallest child or voxel is 20 mm. Since the cube must be defined by its longest side, the minimum number of levels lvs which satisfies the resolution and the minimum side length sl is given by,

$$lvs = \left\lceil \frac{\log \frac{v_r}{\max V}}{\log \frac{1}{2}} \right\rceil \tag{2.15}$$

$$sl = \frac{v_r}{(\frac{1}{2})^{lvs}} \tag{2.16}$$

where v_r is the minimum required voxel resolution. In this example the octree will have 10 levels and side length of the root node will be 20480 mm.

When assigning data such as $p = [x, y, z]$ to one of the voxels, the first part of the recursive algorithm is given by,

$$node(\boldsymbol{p}) \quad = \quad lt(x, n_{cx}) \quad + \quad 2 \cdot lt(y, n_{cy}) \quad + \quad 4 \cdot lt(z, n_{cz}) \tag{2.17}$$

where each node centre $n_c = [n_{cx}, n_{cy}, n_{cz}]$ is in coordinates with respect to the base coordinate system and $lt(x, y)$ is given by

$$lt(x, y) \quad = \quad \begin{cases} 0 & \text{if } x < y \\ 1 & \text{if } x \geq y \end{cases} \tag{2.18}$$

The algorithm will run $node(\boldsymbol{p})$ as long as the node length $nl > v_r$, creating a child node for each respective parent.

The search for the presence of a voxel at a certain coordinate can be done with the same recursive approach as for insertion. The main difference is of course to only check if there exists a child node using $node(\boldsymbol{p})$ until the voxel level is reached. There is also the case where the child nodes and the final voxel are never created. In

this case the search will end earlier and the potentially non performed checks have led to faster algorithm execution. (Meagher, 1982) can provide further information on octrees.

2.4 Artificial Potential Fields

The main idea behind the potential field is to describe the energy as a set of attractive and repulsive fields. By following the gradient of a potential field, the object should find a path to the goal without colliding with any obstacles.

The potential field $U(x)$ can be defined as the set of attractive and repulsive fields,

$$U(x) = U_{att}(x) + U_{rep}(x) \tag{2.19}$$

where $x \in R^3$ is the current position of the object and $U(x)$ is the resultant potential. Depending on the application, the attractive potential can be described by a quadratic and a conical function,

$$U_{att}(x) \quad = \quad \begin{cases} \frac{1}{2}\gamma d^2, & \text{if } d \leq \tau_{goal} \\ \gamma \tau_{goal} d - \frac{1}{2}(\tau_{goal})^2, & \text{if } d > \tau_{goal} \end{cases} \tag{2.20}$$

where $d = d(x, x_{goal})$ is the distance between x and goal position, γ is the attraction gain and τ_{goal} is used as a threshold because of the discontinuity of the conical function at zero. The idea is that the object should follow the negative gradient of $U_{att}(x)$, as shown in the upper left of figure 2.6 to reach the target. For this case, the decline in energy is constant until the value τ_{goal} at $|2|$ has been reached. If the distance between the object and the goal is less than τ_{goal}, the decline in energy is exponential until the object reaches the goal and the energy is zero.

$$\nabla U_{att}(x) \quad = \quad \begin{cases} \gamma(x - x_{goal}), & \text{if } d \leq \tau_{goal} \\ \gamma \frac{x - x_{goal}}{d(x, x_{goal})} \tau_{goal}, & \text{if } d > \tau_{goal} \end{cases} \tag{2.21}$$

The repulsive field is defined such that the field does not affect the object when it

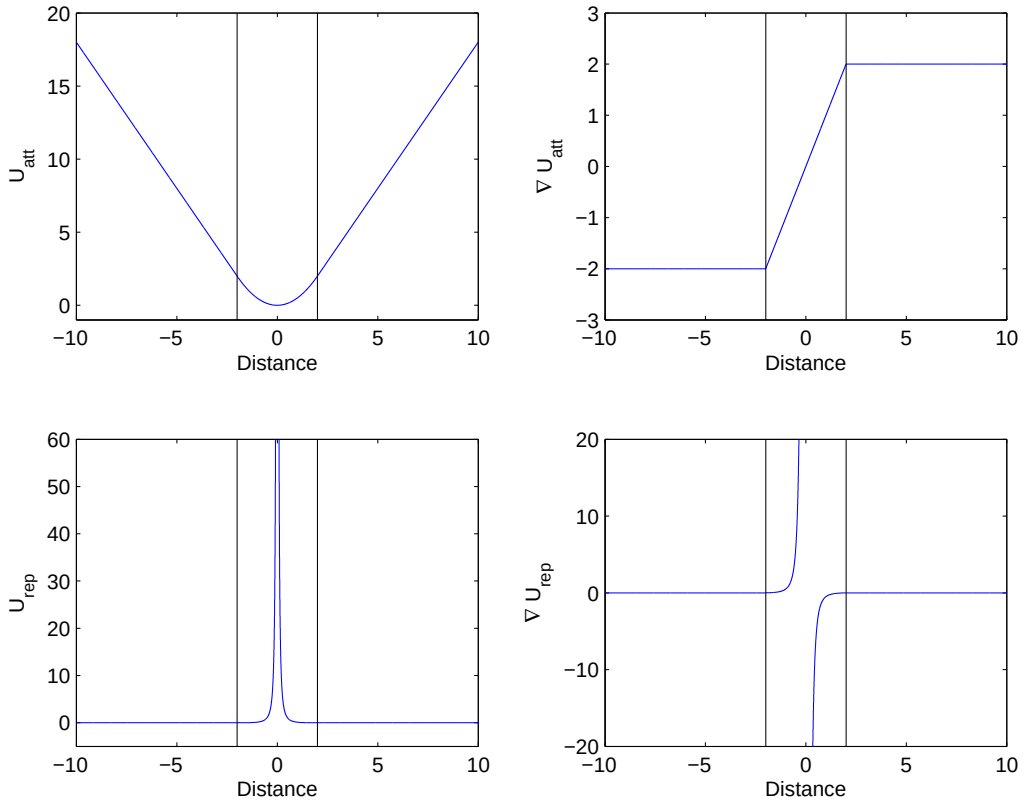


Figure 2.6: *The potential field function U_{att} and U_{rep} with values $x_{goal} = 0$, $x \in [-10, 10]$, $\gamma = 1$, $\kappa = 1$, $\tau_{goal}, \tau_{obs} = 2$*

is further away than a distance τ_{obs}, as opposed to the attractive field which acts on the object regardless of the distance from the goal. The repulsive potential field is defined as,

$$U_{rep}(x) = \begin{cases} \frac{1}{2}\kappa(\frac{1}{D(x)} - \frac{1}{\tau_{obs}}), & \text{if } D \leq \tau_{obs} \\ 0, & \text{if } D > \tau_{obs} \end{cases} \tag{2.22}$$

where D is the distance from the object to the obstacle and κ is the repulsive gain. Finally the rate of change in the repulsive energy is given by,

$$\nabla U_{rep}(x) = \begin{cases} \kappa(\frac{1}{\tau_{obs}} - \frac{1}{D(x)})\frac{1}{D^2(q)}\nabla D(q), & \text{if } D \leq \tau_{obs} \\ 0, & \text{if } D > \tau_{obs} \end{cases} \tag{2.23}$$

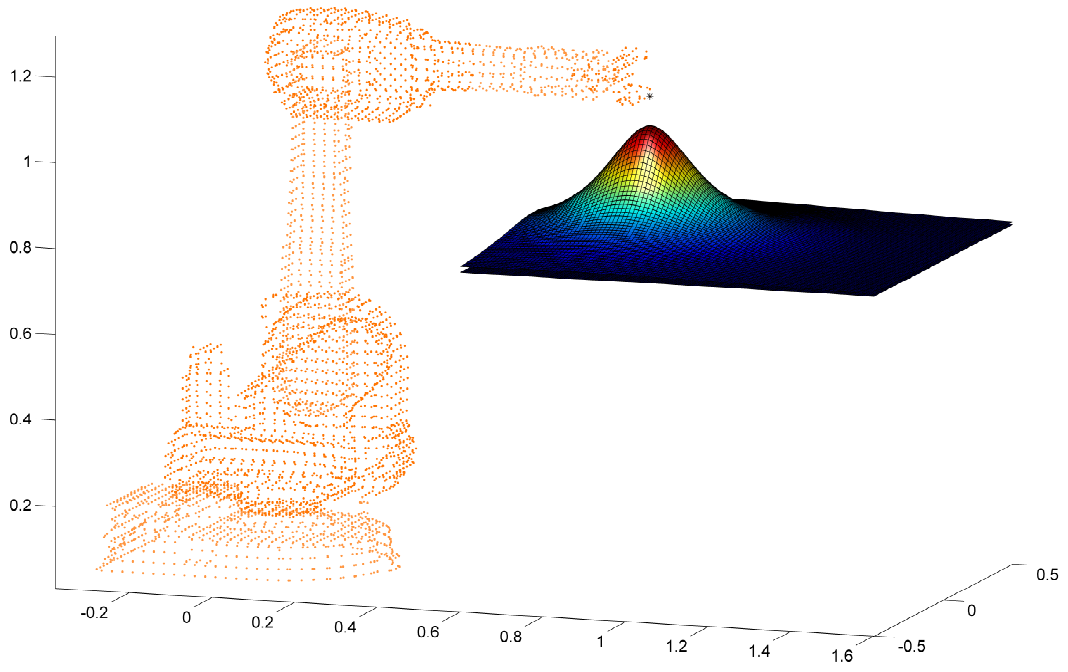


Figure 2.7: *Magnitude of the potential generated by the blue plane (obstacle) and the lower right point on the robot tool.*

For the case of the robot, which in this particular experiment is represented by a number of points as in figure 2.7, each of the points will act like an object. Each of these objects will be affected by the potential field, and they will all have their own position x relating them to the obstacle. The obstacles can also be a set of points such as data from point clouds. Figure 2.7 is an example of a case where only one object, the black point at the tool tip, is depicted with its repulsive potential field. The obstacle, the bottom plate, is represented by N points which result in the repulsive field $U_{rep}(x_i)$ which is calculated for $i \in [1, 2 \cdots N]$,

$$\boldsymbol{U}_{rep}(\boldsymbol{x}) = \sum_{i=1}^{N} \boldsymbol{U}_{rep}(\boldsymbol{x_i}) \tag{2.24}$$

In the case where the whole link of M points is considered, the repulsive field

can be described by

$$\boldsymbol{U}_{rep}(\boldsymbol{x}) = \sum_{m=1}^{M} \sum_{i=1}^{N} \boldsymbol{U}_{rep}(\boldsymbol{x}_m, \boldsymbol{\tau}_{m,obs}) \tag{2.25}$$

In the case of robotics it is sometimes more preferable to use forces to control the robot movement, and for such cases the artificial potential field can be used. Forces are generated based on the potential field and can be defined as,

$$F_{art}(x) = F_{att}(x) + F_{rep}(x) \tag{2.26}$$

where

$$F_{att}(x) = -\nabla U_{att}(x) \tag{2.27}$$

$$F_{rep}(x) = -\nabla U_{rep}(x) \tag{2.28}$$

The work described in this thesis is based on the artificial potential fields algorithms described above. (Siegwart et al., 2011) and (Khatib, 1986) are relevant sources for additional information on the topic.

2.5 Parallel Computing

Parallel computing has received increased interest during the last years. One of the reasons for the increased interest is that the CPU frequency hit the roof with regards to frequency divided by power (Ross, 2008). A way to handle this problem is to increase the efficiency of the CPU, by lowering the size of the transistors and therefore decreasing the heat loss. This problem is not easy to solve when the size of the transistors is approaching the size of atoms. A trend has therefore been to increase the number of cores in the processor. Ideally, each core acts as a separate processor, and as such the frequency per unit power increases.

There is of course a challenge with having two processors (or cores). If the processor has two cores running at 3GHz, this does not automatically mean that the processor in theory runs at 6GHz and twice the amount of work will get done. Actually, to be able to utilise the resources of the CPU fully, the application must

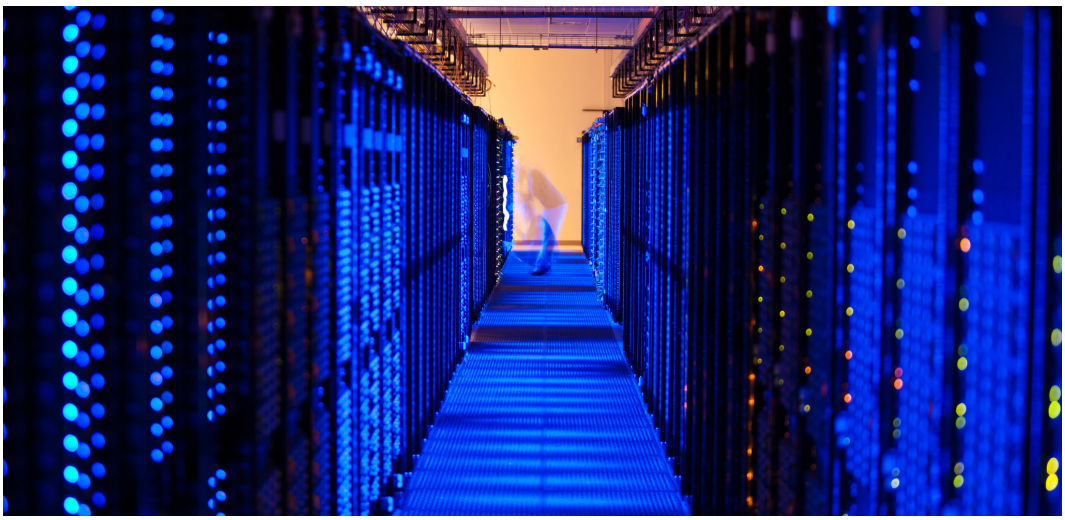


Figure 2.8: *Computers facilitating massive parallel computing at Amazon's web services data centre. (Photo courtesy of The New York Times)*

be written to utilize resources in parallel. Typically, highly parallel applications are not to a wide extent part of an end users platform, but are found on servers getting multiple requests from different end users or other clients.

Availability of parallel computer power changed with cloud computing, allowing individuals to instantiate numerous virtual machines in matter of minutes, connect them in a virtual network and start processes. A few of the known service providers are Amazon Web Services (figure 2.8), Rack Space and Windows Azure.

Cloud robotics Hu et al. (2012) is a concept where robots can utilise cloud resources and services through an internet connection. The robot can benefit from the cloud by redirecting calculations commonly performed by an on-board processor to remote servers. For example by uploading an image of an object that is unknown to the robot. The servers can then run the necessary image recognition algorithms while accessing databases containing relevant information. If the object is recognised, object attributes and other relevant information can be sent back to the robot. One of the advantages with cloud robotics is that computationally intensive tasks can be offloaded to external servers and therefore save time. Another advantage is that lower on-board CPU utilisation can extend the time between battery charges.

Cloud resources and services are well suited for solving many research prob-

lems, but they are not well suited for real-time applications.

For real-time applications, it is common to structure the program and simplify the problem sufficiently such that it is possible to perform the calculations on the CPU in the given time frame. The time frame is dependent on each particular application but can be restricted by external components such as sensor update rates.

A typical application has a serial structure, but usually parts of it can be parallelised. The typical speedup of an application is given by Amdahl's law (Amdahl, 1967),

$$T(n) = t_0(B + \frac{1}{n}(1-B)) \tag{2.29}$$

where T(n) is the calculation time, t_0 is the original calculation time with only one core, $B \in [0,1]$ is the serial fraction of the application and n is the number of cores. All common software applications have a serial fraction which cannot be optimized.

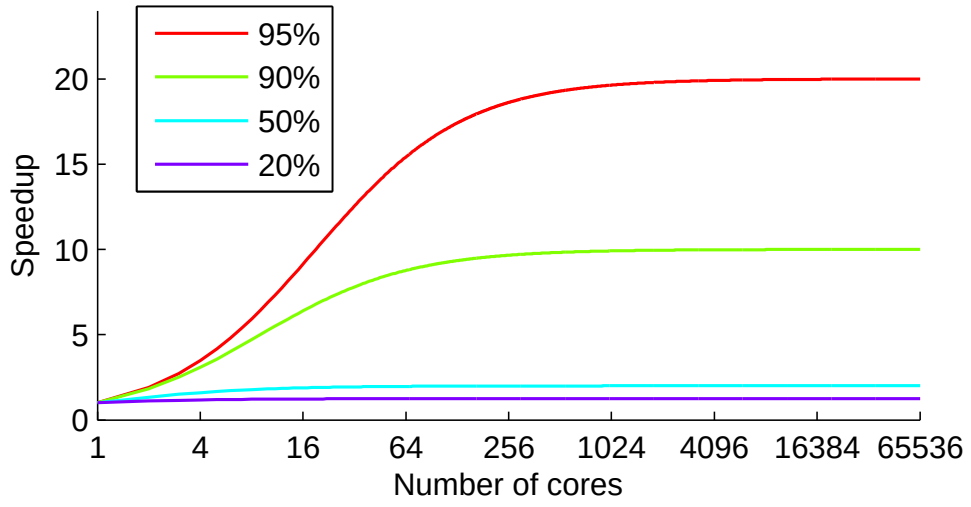


Figure 2.9: *Speedup using parallel computing, where each line denotes the parallel fraction of the application.*

The smaller the serial fraction of the program is, the larger is the potential for increased execution time. If the serial fraction of the program accounts for 10% of the

calculation time, the remaining 90% can be improved. As the number of cores approaches infinity ($n \to \infty$) the only remaining fraction is the serial calculation time. This example would lead to a maximum of 10 times increase in execution time if the core count approaches infinity. As shown in figure 2.9, the speed increase a program can achieve is limited to the execution time of the strictly serial fraction of the program. Parallel processing is not limited to the CPU and can be executed on other hardware, such as the graphics card, or more specifically the graphics processing unit (GPU).

2.5.1 GPU

As an alternative to serial processing on the CPU, or low core count parallel processing on the CPU, the gaming or professional versions of multicore graphic cards have found their place. Initially, the GPU was only for processing graphics, and was not easily accessible to developers who wanted to perform GPU computing. In 2006, NVIDIA released the first graphics card built on the Compute Unified Device Architecture (CUDA) (Sanders & Kandrot, 2010), which enabled developers to target an architecture with components specifically designed for GPU calculations.

CUDA provides a set of drivers and libraries for transferring data to the graphics card to enable calculations on thousands of cores in parallel. The technology can be used to handle some of the challenges that constitute the parts of a collision detection or collision avoidance algorithm. In contrast to the CPU, the graphics processing unit (GPU) cannot run separate functions on all its cores, it has to be the same function running on all the cores simultaneously. However, it is possible in some versions of the architectures to have some core clusters to run independent functions, or more precisely called kernels, of each other, but the essence is that the problem must be calculated in a highly parallel fashion to get a good overall algorithmic speed increase. Problems well suited for parallel processing are the typical scenario where points from sensor data can be examined individually. For the more specific problem of collision detection or collision avoidance, each point generated from the sensor data can be examined individually based on a set of criteria. These criteria will be part of a function which will run on all the cores in parallel. Even though fast execution time can be achieved, it will still rely on data from sensors in

real applications.

The programming of the GPU can be done in the supported programming languages CUDA C/C++ or FORTRAN.

2.5.2 Kernels

In the CUDA programming language, a kernel is similar to what usually is called a function in regular programming. The main difference is in how they are called and executed, and where the developer must put in more effort.

On the GPU there are streaming multiprocessors (SMs) which handle arithmetic, special function units, scheduling of wraps (32 threads) and shared memory among the threads.

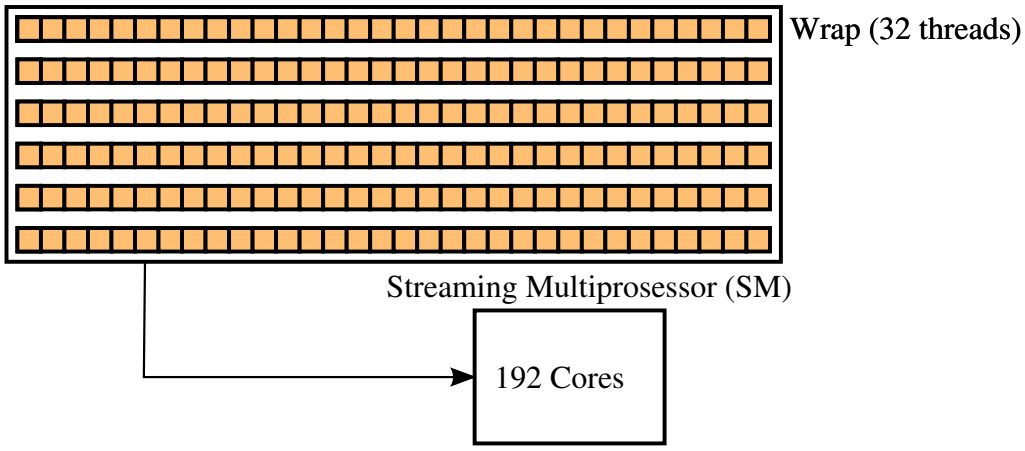


Figure 2.10: *Each thread contains a set of instructions which are processed by the cores in the SM. The Kepler architecture contains 192 cores per SM, and specifically the NVIDIA TITAN contains 14 SMs.*

A kernel is launched with parameters such as number of blocks and threads per block and the product of these two determine the number of kernel launches. A typical kernel call in C++ is given below:

$$\text{MultiplyNumbers} <<< \text{numBlocks}, \text{threadsPerBlock} >>> (\text{Ain}, \text{Bin}, \text{Cout});$$

From this kernel it is clear that there are only two input parameters (Ain and Bin) and one output parameter (Cout). The parameters will typically be pointers to ar-

rays, such that each kernel will work on a data pair. The actual kernel can be defined as,

```
__global__ voidMultiplyNumbers(float * Ain, float * Bin, float * Cout)      (2.30)
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    C[idx] = A[idx] * B[idx];
}
```

The kernel runs $numBlocks \cdot threadsPerBlock = idx_{max}$ number of times, but it is not possible to describe the length of all arrays by this product. If idx_{max} is larger than the length of the array, it will result in an out of bounds memory access. To prevent this, an if statement can be used to prevent read and write if $idx > n$ array elements. Another way to circumvent the problem is to omit the "if statement" and rather increase the array to the size of idx_{max}. One might ask, will this result in extra calculations and extra calculation time? Because all the cores which are designated by the SM have to run the exact same kernel, it does not matter if they take a break or not as long as one of the cores will finish at a later time, they will have to wait. Figure 2.11 gives a visual example on how the threads are related to the blocks.

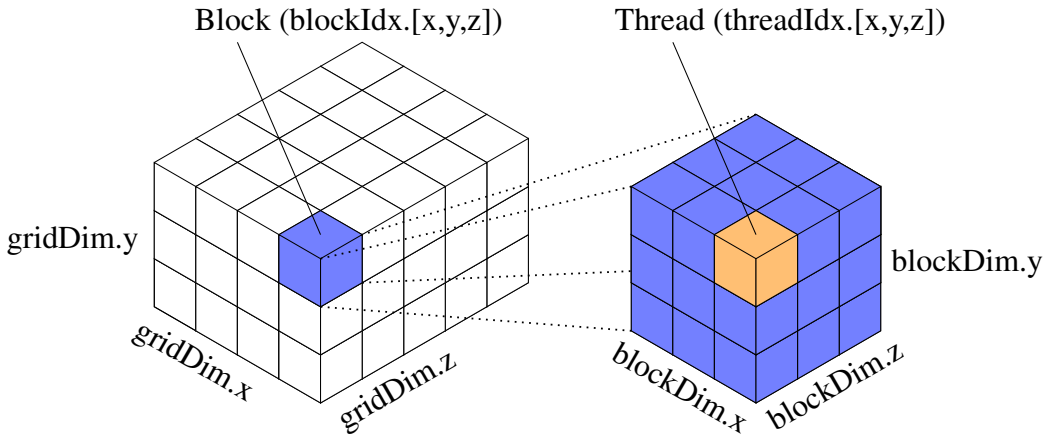


Figure 2.11: *A grid of blocks (left) and a block of threads (right).*

2.5.3 Memory management

The GPU can be used for parallel calculations to increase the speed of the application according to Amdahl's law, but it is important to take all the serial parts into account. When considering computation time reduction in parallel computing, it is very important to take the transfer time of the data to the GPU memory into account. This transfer time can in some cases account for a significant portion of the program execution time, and can in some cases be larger than the actual calculation (Gregg & Hazelwood, 2011). If it is possible to use a data block of known size, use of pinned memory reserves the block and increases the transfer speed. Depending on the hardware, the transfer rates can be several Giga Bytes per second (GB/s).

For developers of regular computer programs in C or C++, allocation and deallocation of memory is one of the most common tasks that must be handled. Faulty memory management is maybe one of the most common reasons for bugs, and can sometimes be hard to track down. But even in a multi-threaded CPU application, the complexity of memory management does not increase significantly. All the memory is still accessed in a similar fashion, but it is important to avoid race conditions such as multiple writes or write/reads at the same location. The GPU memory is structured in a different manner than the CPU memory. The three main categories are structured as following, where each type is faster than the previous:

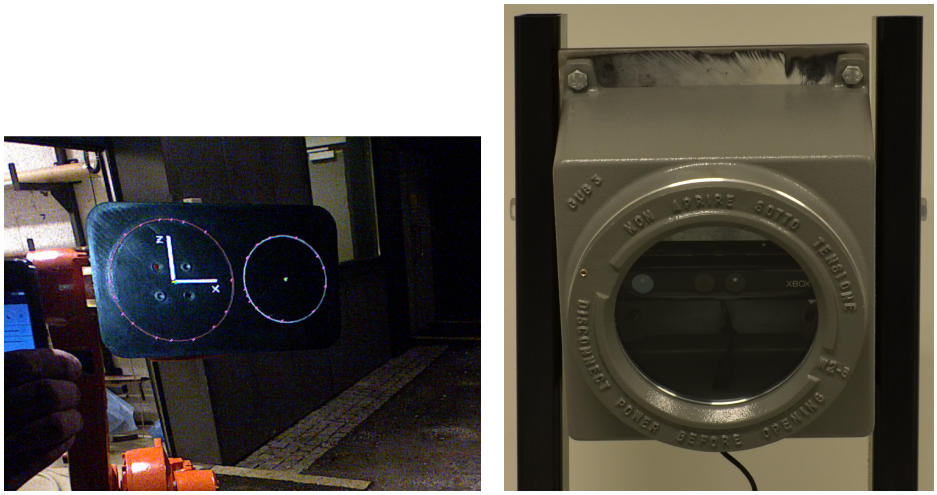
1. Global memory is available to:
    (a) All streaming processors
    (b) All threads
2. Shared memory available to:
    (a) All threads in a block
3. Local memory available to:
    (a) An individual thread

Each new CUDA architecture introduces additional functionality and changes in hardware. Because the hardware more or less dictates how the developer should

structure the program to utilise the GPU to maximise for speed, it is important to be familiar with the architecture that is targeted. Key references on this topic are (Wilt, 2013) and (NVIDIA, 2013)

2.6 Sensor Calibration

This section describes the process used for calibrating the position and orientation of the Kinect sensor with regards to the robot base. Recalling section 1.5.2, this is equivalent of finding the transformation from the robot base frame to the camera frame.



(a) (b)

Figure 2.12: *(a) Snapshot from the GUI where the calibration plate is fixed on the robot wrist. (b) The Kinect sensor mounted inside an ATEX certified casing.*

A calibration plate containing two circles is mounted onto the wrist. The two circles are parallel to the wrist plane and the left circle in figure 2.12 is centred in the wrist. By utilising classes in the Microsoft Kinect SDK, position of each point in the depth data can be obtained in world coordinates with regards to the optical sensor frame. The accuracy of the obtained Kinect sensor data varies but has a mean accuracy close to 0 mm for all three axes (Khoshelham & Elberink, 2012) thus, by acquiring more than one point to describe the centre of tool is expected to give a mean accuracy approaching 0 mm. By calculating the centre point on the plane, it

is possible to find a transformation matrix from the Kinect sensors' optical frame to the calibration plate left centre, and from there to the robot wrist. The points on the circles are assigned manually by the use of a graphical user interface. The manual assignment of points should be possible to fully automate, but this has at the time of writing not been implemented. The following algorithm generates a set of centre points $\boldsymbol{p}_c^{(i)}$ based on unique sets of points $(\boldsymbol{p}_u, \boldsymbol{p}_v, \boldsymbol{p}_w)$ located at the circle of the calibration plate.

$$\begin{aligned} \boldsymbol{p}_c^{(i)} = f_c(\boldsymbol{p}_u, \boldsymbol{p}_v, \boldsymbol{p}_w) \quad & \forall u \in \{1,2,...,N-2\}, \\ & \forall v \in \{v, v+1, ..., N-1\}, \\ & \forall w \in \{w, w+1, ..., N\} \end{aligned}$$

where $f_c(\boldsymbol{p}_u, \boldsymbol{p}_v, \boldsymbol{p}_w) = \boldsymbol{p}_c$ is a function which calculates the centre point by using Barycentric coordinates from cross- and dot-products and is calculated as follows,

$$p_c = \alpha \boldsymbol{p_u} + \beta \boldsymbol{p_v} + \gamma \boldsymbol{p_w} \tag{2.31}$$

where r is the circle radius. Calculation of α, β and γ is as follows,

$$\alpha = \frac{|\boldsymbol{p_v} - \boldsymbol{p_w}|^2 (\boldsymbol{p_u} - \boldsymbol{p_v}) \cdot (\boldsymbol{p_u} - \boldsymbol{p_w})}{2|(\boldsymbol{p_u} - \boldsymbol{p_v}) \times (\boldsymbol{p_v} - \boldsymbol{p_w})|^2} \tag{2.32}$$

$$\beta = \frac{|\boldsymbol{p_u} - \boldsymbol{p_w}|^2 (\boldsymbol{p_v} - \boldsymbol{p_u}) \cdot (\boldsymbol{p_v} - \boldsymbol{p_w})}{2|(\boldsymbol{p_u} - \boldsymbol{p_v}) \times (\boldsymbol{p_v} - \boldsymbol{p_w})|^2} \tag{2.33}$$

$$\gamma = \frac{|\boldsymbol{p_u} - \boldsymbol{p_v}|^2 (\boldsymbol{p_w} - \boldsymbol{p_u}) \cdot (\boldsymbol{p_w} - \boldsymbol{p_v})}{2|(\boldsymbol{p_u} - \boldsymbol{p_v}) \times (\boldsymbol{p_v} - \boldsymbol{p_w})|^2} \tag{2.34}$$

$$\boldsymbol{p}_{cm} = \frac{1}{M} \sum_{i=1}^{M} \boldsymbol{p}_c^{(i)} \quad \forall i \in \{1,2,...,M\}, \tag{2.35}$$

where $\boldsymbol{p}_{cm}$ is the mean centre point and M is the number of centre points in $\boldsymbol{p}_c^{(i)}$.

A unit vector representing the x-axis is calculated from the left centre point to the right centre point on the calibration plate. The z-axis is the plane normal vector, and the y-axis is the cross product between the x-axis and the z-axis. By using the centre point of the left circle $\boldsymbol{p}_{cm}$ and the calculated axis, all parameters are available for the frame describing the calibration plate with regards to the Kinect sensor. Then, by using the kinematics of the robot, a transformation from the end effector to the robot base can be performed. Since the position of the robot base with regards to the Kinect sensor is known, the position of the robot links in the Kinect sensor frame can be determined.

2.7 Collision Avoidance on Industrial Robots

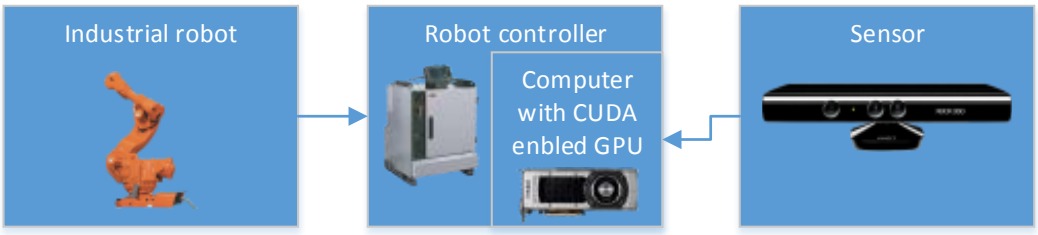


Figure 2.13: *The collision avoidance setup.*

To implement a collision avoidance system that is easy to interface with an industrial robot can be a challenge, particularly if this implementation should be easy to integrate for existing robot programs. For the collision avoidance systems described in this thesis, equipment external to the robot controller is needed. This equipment includes sensors and computers. The sensors in particular have to be mounted somewhere such that they can observe the work cell of the robot, where the mounting location can be on a structure or on the robot itself. The computer can on the other hand be modified to fit inside the control cabinet.

For new solutions to be adapted by industry, plug-and-play solutions are an advantage. A typical ABB robot move instruction and the corresponding proposed

collision avoidance instruction is,

$$\text{MoveL}[\text{pos1}, \text{ori}, \text{cfg}, \text{eAxNone}], \text{vel}, \text{zone}, \text{t1} \backslash \text{WObj} := \text{w1}; \tag{2.36}$$

$$\text{KMoveL}[\text{pos2}, \text{ori}, \text{cfg}, \text{eAxNone}], \text{vel}, \text{zone}, \text{t1} \backslash \text{WObj} := \text{w1}; \tag{2.37}$$

where (2.36) is a regular move instruction and (2.37) is the instruction for a move that incorporates collision avoidance. The algorithms for `KMoveL` are stored in a system module, which from the operators point of view, the only change that has to be made to an existing program in order to enable collision avoidance is to replace function name `MoveL` with `KMoveL`.

Chapter 3

Concluding Remarks

3.1 Conclusions

The work described in this thesis demonstrates collision handling for industrial robots in harsh environments. The first part of the work studies collision detection while the second part studies collision avoidance.

The ES used for collision detection shows fast execution rates suitable for real-time systems. Even though manual map creation can be time-consuming, it is possible to generate maps automatically from a model or sensor data.

Using HMMs for collision detection can be time consuming, but an advantage is that it can be trained off-line from CAD models. An HMM approach trained offline from CAD models is particularly suitable for oil and gas applications where there is lots of available time between commissioning and maintenance.

In the conducted work on collision detection using ES or HMM, the CPU was always taking part in processing the sensor data. Processing this data on the GPU would enable faster algorithm execution rates compared to CPU processing. By using GPU processing, more sensors can be interfaced with the same system and still provide desired performance.

Collision detection for sensors attached to the robot has provided useful results. Even though the robot base position was stationary, it could be envisioned that the robot will move on a track and therefore benefit from a sensor mounted on its structure. The laser scanner mounted on the robot is certified for outdoor use, but it is

not certified for potentially explosive environments. One of the challenges with a sensor that is not certified is that it has to be mounted inside an EX-certified casing. It is expected that an EX-certified casing for a laser scanner would limit the sensors' field of view. In addition, the size of the sensor casing needed to encapsulate the sensor might make it undesirable to mount it on the robot. On the other hand, a sensor which is fixed to a structure can provide good results as long as the robot kinematics is considered in the filtration algorithm. Again, a fixed sensor can be a challenge if the robot is moving on a track, because the track can position the robot outside the sensors field of view.

Collision avoidance is similar to collision detection in the sense that the problem can be made highly parallelisable. A disadvantage with collision avoidance is that it usually requires significant computing resources, especially if 3D depth data is used. This can be handled with filtration and simplifications. The work described in this thesis addressed the problem by shifting many of the calculations from the CPU to the GPU. The approach provided real-time collision avoidance while still using a rather detailed representation of the robot and the environment. This was, to the author's knowledge, the first time the GPU was used for collision avoidance using a real robot.

One of the more general depth sensor related challenges is the limited volume they can observe. For instance if the sensor is facing an obstacle, the surface on the opposite side is not visible. Reliable collision handling applications can address this problem by using multiple sensors, but more sensors come at the cost of computing resources. Since the collision detection algorithm is less complex than the collision avoidance algorithm, it is expected to better handle a higher number of sensors and still operate in real-time.

The work in this thesis studies methods for collision detection and collision avoidance. Solutions for both categories have been demonstrated for industrial robots. These solutions contribute towards offshore robotic co-workers working in potentially explosive environments.

3.2 Future Work

For developing more trustworthy systems, more outdoor tests should be conducted. Aspects such as fog, sunlight, rain and snow are conditions which can affect the sensor data. To address the sunlight, a light measuring sensor can be mounted side by side with the sensor which observes the environment. Then, based on thresholds in the data from the light sensor, a decision regarding the validity of the sensor data can be made.

The work on the GPU and CUDA started at approximately the same time as collision avoidance. Because this was the natural step after the work on collision detection, proper GPU algorithms were never made specifically for collision detection. Such algorithms can be part of future work and should also be tested with more than one sensor.

The developed algorithms for collision avoidance in CUDA C++ were made specifically to calculate the artificial potential field for a 6-DoF and 7-DoF manipulator. It will be beneficial if these algorithms are made more general such that they can be reused for different types of manipulators or other equipment requiring collision avoidance.

Even though GPU programming is not limited to NVIDIA graphics cards and CUDA, only GPUs from NVIDA were programmed. AMD is another large company producing graphic cards for GPU programming. Traditionally, the graphic cards from these manufacturers have been quite similar in performance. As shown in Nishikawa et al. (2013), the AMD Radeon HD 7970 graphic card has a higher throughput than the competing NVIDIA GTX 680. On the other hand, newer graphics cards have now been released by both manufacturers and the reference Nishikawa et al. (2013) may not be up to date.

Sensors are an important part of collision avoidance and collision detection. Mainly two different sensors have been used, the SICK laser scanner and the Microsoft Kinect sensor. It would be interesting to look at other sensors such as profile sensor. By mounting a profile sensor on the robot, it would be possible to scan the environment. The sensor can be used to scan a volume which require more detail. For example if the object which should be manipulated can have different poses, it can be scanned for precise positioning of the tool before engaging manipulation

and as such potentially avoid collision. This type of scanner can also be used to scan the working area, and the data can be stored e.g. for use as a supplementary environment model for collision avoidance or collision detection.

Regular RGB cameras were not considered because they do not provide depth data. It is of course possible to use a stereo camera setup, but it does require additional calibration as opposed to a purposely built depth sensor. Even though a single camera does not provide depth information, it can be used to provide environment information which the depth camera lacks. The use of a RGB camera as a supplement to a collision handling application can be worth further studies.

Building on the lessons learnt from the work described in this thesis, there are many available paths open for further studies and research.

REFERENCES

Amdahl, G. M. (1967). Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), (pp. 483–485)., New York, NY, USA. ACM.

Centrica Energy (2010). *Centrica Energy Upstream asset book*. Berkshire, United Kingdom: Centrica Energy.

Ching, W. K. & Ng, M. K. (2006). *Markov Chains*. Springer.

Craig, J. J. (2005). *Introduction to robotics: Mechanics and Control. 2005*. Prentice Hall, Englewood Cliffs, NJ.

De Luca, A. & Flacco, F. (2012). Integrated control for phri: Collision avoidance, detection, reaction and collaboration. In *4th IEEE RAS EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob)*, (pp. 288–295).

Ferrein, A., Niemueller, T., Schiffer, S., & Lakemeyer, G. (2013). Lessons Learnt from Developing the Embodied AI Platform Caesar for Domestic Service Robotics. In *AAAI Spring Symposium, Designing Intelligent Robots: Reintegrating AI*.

Fink, G. A. (2008). Markov models for pattern recognition. *p127*.

Flacco, F., Kroger, T., De Luca, A., & Khatib, O. (2012). A depth space approach to human-robot collision avoidance. In *Robotics and Automation (ICRA), 2012 IEEE International Conference*, (pp. 338 –345).

Fuchs, S., Haddadin, S., Keller, M., Parusel, S., Kolb, A., & Suppa, M. (2010). Cooperative bin-picking with time-of-flight camera and impedance controlled dlr lightweight robot iii. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (pp. 4862–4867).

Gregg, C. & Hazelwood, K. (2011). Where is the data? why you cannot debate cpu vs. gpu performance without the answer. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, (pp. 134 –144).

Haddadin, S., Belder, R., & Albu-Schäffer, A. (2011). Dynamic motion planning for robots in partially unknown environments. In *IFAC World Congress (IFAC2011)*, Milano, Italy.

Haddadin, S., Suppa, M., Fuchs, S., Bodenmller, T., Albu-Schffer, A., & Hirzinger, G. (2011). Towards the robotic co-worker. In C. Pradalier, R. Siegwart, & G. Hirzinger (Eds.), *Robotics Research*, volume 70 of *Springer Tracts in Advanced Robotics* (pp. 261–282). Springer Berlin Heidelberg.

Hu, G., Tay, W. P., & Wen, Y. (2012). Cloud robotics: architecture, challenges and applications. *Network, IEEE*, *26*(3), 21–28.

Jackson, P. (1990). *Introduction to expert systems*. Addison-Wesley Longman Publishing Co., Inc.

Jazar, R. N. (2010). *Theory of applied robotics*. Springer.

Kaldellis, J. & Kapsali, M. (2013). Shifting towards offshore wind energyrecent activity and future development. *Energy Policy*, *53*, 136–148.

Kaldestad, K. B., Hovland, G., & Anisi, D. A. (2012). Obstacle Detection in an Unstructured Industrial Robotic System: Comparison of Hidden Markov Model and Expert System. In *10th IFAC International Symposiums on Robot Control*, Dubrovnik, Croatia.

Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, *5*(1), 90–98.

Khoshelham, K. & Elberink, S. O. (2012). Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications. *Sensors*, *12*(2), 1437–1454.

Leroux, P. (2007). New regulations and rules for atex directives. *IEEE Industry Applications Magazine*, *13*(1), 43–51.

Meagher, D. (1982). Geometric modeling using octree encoding. *Computer graphics and image processing*, *19*(2), 129–147.

Nishikawa, N., Iwai, K., Tanaka, H., & Kurokawa, T. (2013). Throughput and power efficiency evaluations of block ciphers on kepler and gcn gpus. In *First International Symposium on Computing and Networking (CANDAR)*, (pp. 366–372).

NVIDIA (2013). CUDA C Programming Guide.

Pichler, A., Barattini, P., Morand, C., Almajai, I., Robertson, N., Hopgood, J., Ferrara, P., Bonasso, M., Strassmair, C., Rottenbacher, M., et al. (2013). Tailor made robot co workers based on a plug&produce framework. In *Robotics in Smart Manufacturing* (pp. 113–126). Springer.

Plasch, M., Pichler, A., Bauer, H., Rooker, M., & Ebenhofer, G. (2012). A plug & produce approach to design robot assistants in a sustainable manufacturing environment. In *22nd International Conference on Flexible Automation and Intelligent Manufacturing (FAIM 2012), Helsinki, Finland*.

Rabiner, L. (1989). A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, *77*(2), 257 –286.

Ross, P. (2008). Why CPU Frequency Stalled. *IEEE Spectrum*, *45*(4), 72.

Sanders, J. & Kandrot, E. (2010). *CUDA by Example: An Introduction to General-Purpose GPU Programming* (1st ed.). Addison-Wesley Professional.

Siciliano, B. & Khatib, O. (2008). *Springer handbook of robotics*. Springer.

Siegwart, R., Nourbakhsh, I. R., & Scaramuzza, D. (2011). *Introduction to autonomous mobile robots*. MIT press.

Spong, M. W., Hutchinson, S., & Vidyasagar, M. (2006). *Robot modeling and control*. John Wiley & Sons New York.

Wilt, N. (2013). *The cuda handbook: A comprehensive guide to gpu programming*. Pearson Education.

Yang, Z. R. (2010). *Machine learning approaches to bioinformatics*, volume 4. World Scientific.

Zhang, Z. (2012). Microsoft Kinect Sensor and Its Effect. *IEEE MultiMedia*, *19*(2), 4–10.

Paper A

Obstacle Detection in an Unstructured Industrial Robotic System: Comparison of Hidden Markov Model and Expert System

Knut B. Kaldestad, Geir Hovland and David A. Anisi

This paper has been published as:

Kaldestad, K., Hovland, G., and Anisi, D. Obstacle Detection in an Unstructured Industrial Robotic System: Comparison of Hidden Markov Model and Expert System. *In 10th IFAC Intl. Symposiums on Robot Control.* Dubrovnik, Croatia, 2012b.

Obstacle Detection in an Unstructured Industrial Robotic System: Comparison of Hidden Markov Model and Expert System

Knut B. Kaldestad*, Geir Hovland* and **David A. Anisi
*Department of Engineering
Faculty of Engineering and Science, University of Agder
Jon Lilletunsvei 9, 4879 Grimstad, Norway.
**Department of Technology & Innovation
Division of Process Automation, ABB
Norway

Abstract — **This paper presents a comparison of two approaches for detecting unknown obstacles inside the workspace of an industrial robot using a laser rangefinder for 2-D measurements. The two approaches are based on Expert System (ES) and Hidden Markov Model (HMM). The results presented in the paper demonstrate that both approaches are able to correctly detect and classify unknown objects. The ES is characterised by low computational requirements and an easy setup when relatively few known objects are to be included inside the workspace. HMMs are characterised by a higher flexibility and the ability to handle a larger amount of known objects inside the workspace. Another significant benefit of the HMM approach taken in this paper, in contrast to voice recognition, is the fact that the learnt parameters of the HMMs have physical meaningful geometrical interpretations.**

Keywords — Collision Detection, Industrial Robot, Hidden Markov Model, Expert System.

1 Introduction

Traditionally, industrial robots have been used in structured indoor environments, such as manufacturing plants or foundries. In such environments, objects inside the

workspace of the robot are most often either fixed at a known location or moving according to *a priori* determined and known patterns. Hence, the robot programs can be written such that these objects are avoided. Recently, there have been several initiatives to use industrial robots in unstructured outdoor environments. As an illustrative example, consider the case of robot automation in the oil and gas industry, (Anisi et al., 2010, 2011). In such environments, due to mismatch between existing world model (CAD) and the exact location of process components, the location of potential obstacles inside the workspace is partly unknown and potential collisions cannot be avoided solely by offline programming. To avoid collisions in such settings, online, sensor-based collision detection and avoidance methods must be adopted. Collision detection considers the problem of indicating the presence of unmodelled obstacles while collision avoidance aims at finding an alternative path around discovered obstacles leading to the target. To this end, use of external sensors, *e.g.*, laser-, ultrasound- and vision sensors is necessary and the robot programs must be able to process these data and adjust accordingly online.

The work presented in this paper describes two different approaches for obstacle detection in an unstructured environment. The two approaches are based on Hidden Markov Model (HMM), (Fink, 2007) and Expert System (ES), (Aarts and Marzano, 2003). HMMs are based on the probabilities of a sequence of observations while an ES calculates the state based on initial knowledge about the system. The paper compares the two methods based on criteria such as recognition rates, limitations, benefits, implementation time and practical use. As far as hardware is considered, the experiments in this paper are based on a laser rangefinder sensor and an industrial robot. The choice of sensor was important, due to strict regulations in the oil and gas industry hence using laser, the equipment must be explosion certified or certifiable, withstand tough weather conditions and be weather protected (Ingression Protection IP67 or higher). In this paper, as a first step, the performance of the HMM and ES algorithms are verified in an indoor environment, although the selected laser rangefinder is certified for outdoor conditions. Most industrial robots are designed for indoor usage, but additional outdoor protection can sometimes be optionally delivered by supplier.

There are not many articles in the literature describing the combination of obstacle

detection and a laser rangefinder in an industrial robotics environment. In particular the HMM approach has, to the authors' knowledge, seldom been applied to obstacle detection in robotics. Most of the related research in this area has focused on mobile robots and different applications such as localisation and mapping. An example is the research described by (Wolf et al., 2005) where the authors investigate a method for mobile robots to detect the state of the terrain. The robots are equipped with a laser rangefinder at an angle of 35^o and 40^o relative to the ground and are then able to produce a map in 3D when driving forward. To be able to determine whether parts of the scanned area belong to one of two states: flat terrain or rougher terrain, the authors use a HMM approach to estimate the state. The estimation of the state is based on comparison of successive scans. In contrast to (Wolf et al., 2005), the approach in this paper focuses on obstacle detection and not mapping. In addition, the presented method requires only one scan to determine the state sequence, but the probability of the observations of many models are calculated and evaluated.

(Zhu, 1990) presents a stochastic HMM-based algorithm for describing motion of obstacles. The work is similar to this paper by working in 2D-space. It differs by using image data instead of laser data. Further it calculates the probability of an object to be in a particular position, as opposed to the object being in a position that is not allowed.

Using a laser rangefinder, (Fulgenzi et al., 2009) describes the navigation of a vehicle from an initial position (A) to a destination position (B). HMMs are used to predict pedestrians movements such that the vehicle can continuously calculate a path with highest probability for success. The work differs from this paper in that the car is allowed to move to the destination following any possible path, as long as collision is avoided. The work described in this paper is restricted to follow a predetermined path, and does not allow any deviation.

(Hovland and McCarragher, 1999) presents a HMM approach for classifying discrete states in a force-controlled robotic assembly operation. The work is similar to this paper by the fact that several HMMs are used to represent different scenarios or workspace locations. In this paper only a single HMM is used at a given time, while in (Hovland and McCarragher, 1999) all the HMMs were compared continuously and the HMM with the highest score represented the actual situation.

The paper is organised as follows: Section 2 presents the problem formulation, Section 3 explains the ES while Section 4 introduces the HMM approach. Finally, Section 5 presents the experimental results and Section 6 concludes the work.

2 Problem Formulation

The problem setup (Fig. A.1) considered in this paper is as follows. An industrial robot (J) is conducting a manipulation operation. While doing so, a laser range finder (E) scans for objects inside the workspace of the robot. If an object is detected, it could either be a part of the normal situation or it could be an object that is not expected to be inside the workspace, *i.e.*, an obstacle to be avoided. As an example, an object that is part of the normal situation could be tool stands, work tables or vises with fixed and known positions. Upon detection of these known objects, the robot operation should proceed as normal. On the other hand, the detection of unknown obstacles should immediately raise a signal to the controller.

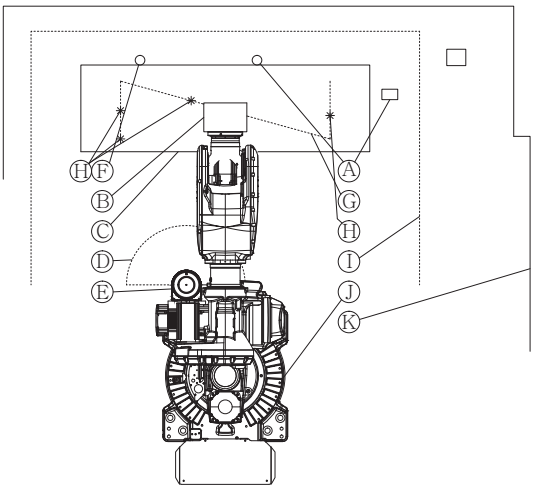


Figure A.1: *Experiment setup.*

The robot tool (B) moves along an offline-generated path (G) while manipulating an item on the work bench (C). The laser range finder (E) continuously scans the plane, physically limited by the walls (K), in the range of 180^o (D), where it detects the presence of known objects (A), and unknown obstacle (F). Finally, four arbitrary

tool locations (H) have been chosen for system comparison between the ES and the HMM approaches. Unknown obstacles are allowed to be present at any location in the workspace, not only on the work table (C).

3 Expert System

The ES used in this paper is an intuitive, low complexity approach which is described by the algorithm in Fig. A.2. The algorithm uses robot joints and the tool position to calculate the relative position between the laser and the tool. Since the tool holder is always parallel to the work bench, the tool-tip will change angle relative to the laser, and thus prevent a straight forward implementation of the algorithm. The laser measurements of the tool and the objects that are part of the normal situation are filtered out. Further, the complex environment could be represented by a CAD model, but in the described ES experiments, the environment is removed by the boundary filter (I) in Fig. A.1. The boundary filter is recalculated continuously to account for laser offset and rotation. Fig.A.3 (a) shows an example of the ES in use. The blue line shows the filter, and any measurements that are on the sensor-side of the line, which is the workspace of the robot, is reported to the robot controller. The tool is continuously filtered out from the laser measurements based on its current position.

4 Hidden Markov Models

Hidden Markov Model (HMMs) are used as a statistical approach to solve the problem described in Section 2. The HMMs will give a probability measure on how likely it is that an obstacle is inside the workspace of the robot. HMMs are based on state transition probabilities and the probability of being in a state given an observation or a sequence of observations. The interested reader is referred to (Rabiner, 1989) for a rigorous explanation. The observation vector $\boldsymbol{O}$ and the individual states $\boldsymbol{S}$ are denoted as,

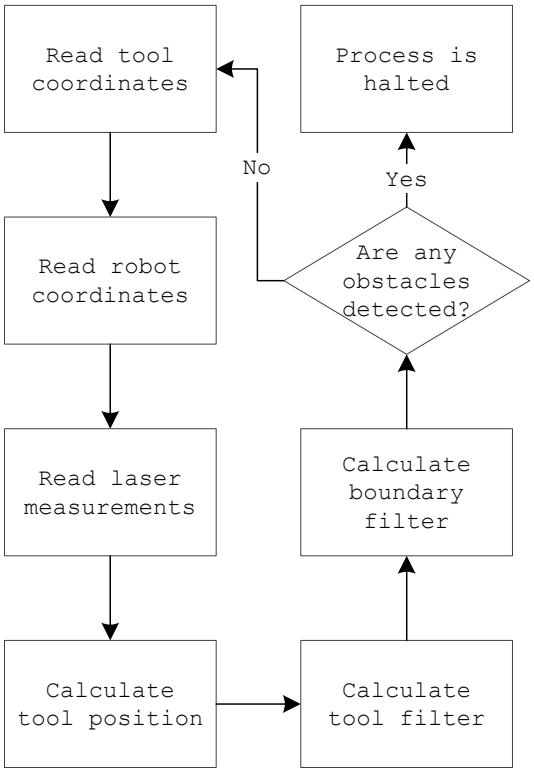


Figure A.2: *Expert System: Algorithm.*

$$\boldsymbol{O_t} = O_1, O_2, \cdots, O_T, \; \boldsymbol{S_n} = S_1, S_2, \cdots, S_{N_S}$$

where $t \in \{1, 2 \cdots, T\}$ and T is the number of observation time steps. N_S is the number of states in the HMM. In this paper a 12-by-12 transmission matrix $\boldsymbol{A}$ is represented by Fig. A.4. Even if the model has 12 states, only a total of five transitions for each observation sequence is allowed. The observations, see Fig. A.5, are divided into 30^o segments. Each segment containing 60 distance measurements, one measurement for every 0.5^o. Every distance measurement has a corresponding mean value (μ) and variance (C).

The probability of being in a state given the observation $\boldsymbol{O_t}$ is given by the observation probability $b_j(\boldsymbol{O_t})$ which is a Gaussian distribution

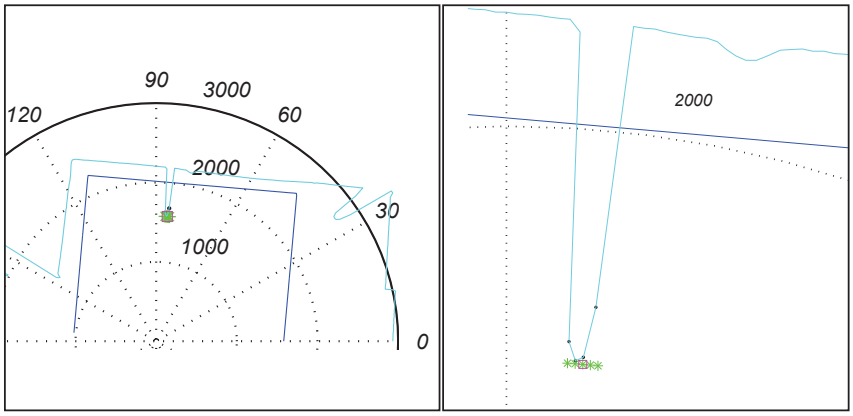


Figure A.3: *Expert System: Illustration of filtering.*

$$b_j(\boldsymbol{O_t}) = \mathcal{N}(\boldsymbol{O_t}|\boldsymbol{\mu}_j, \boldsymbol{C}_j) \tag{A.1}$$

The observation vector $\boldsymbol{O_t}$ is equal in size to the 60 element $\boldsymbol{\mu}_j$ and $\boldsymbol{C}_j$ vectors. For each 10 mm absolute movement along the manipulation path, measurements are conducted and a new model λ_k is created, where $k \in \{1, 2 \cdots, K\}$ and K is the total number of models. Each model is related to a tool position coordinate. If a model score $P(O|\lambda_k)$ is below a predefined threshold, the model does not represent the normal situation. In other words, an unknown object is present in the workspace.

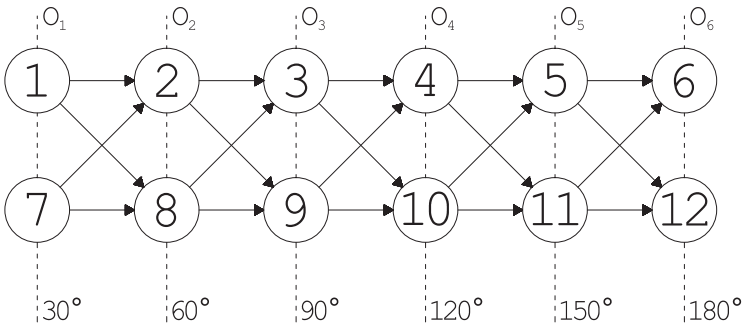


Figure A.4: *HMM transition matrix.*

The hidden states 1-12 in Fig. A.4 describe the normal situation of the workspace.

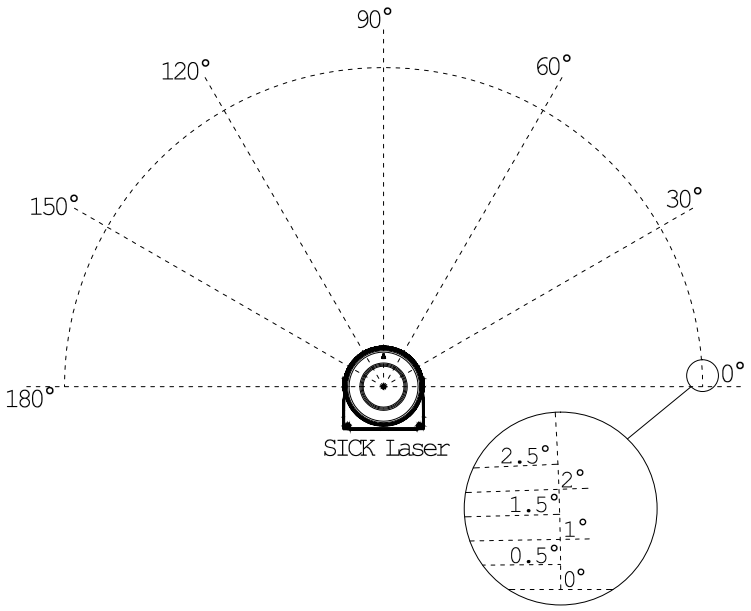


Figure A.5: *The* 30^o *segmentation of the laser rangefinder measurements.*

Each hidden state can describe that no object is present, or it can describe the presence of a known object. The sensor data for each hidden state consists of 60 measurements in a segment of 30^o. The HMM is trained on a pre-classified training set. This means that all the different scenarios defining the normal situation are included in the training data. For example, if the normal situation for the first 30^o segment consists of two cases: A) no objects, and B) one single known object, then state 1 could represent the "no objects" case, while state 7 could represent the "known object" case. If the sensor data for the five remaining 30^o segments contain no further known objects, the hidden state sequence 2-6 after state 7 would typically get a high score.

The particular parallel structure of the HMM with two rows of states, shown in Fig. A.4, was chosen to allow for a representation of two main cases: A) no known objects and B) one known object in each of the six sensor segments as shown in Fig. A.5. If several different known objects are to be placed inside a single sensor segment or the location/size of the objects can vary inside a single sensor segment, then the HMM should be expanded with additional parallel branches.

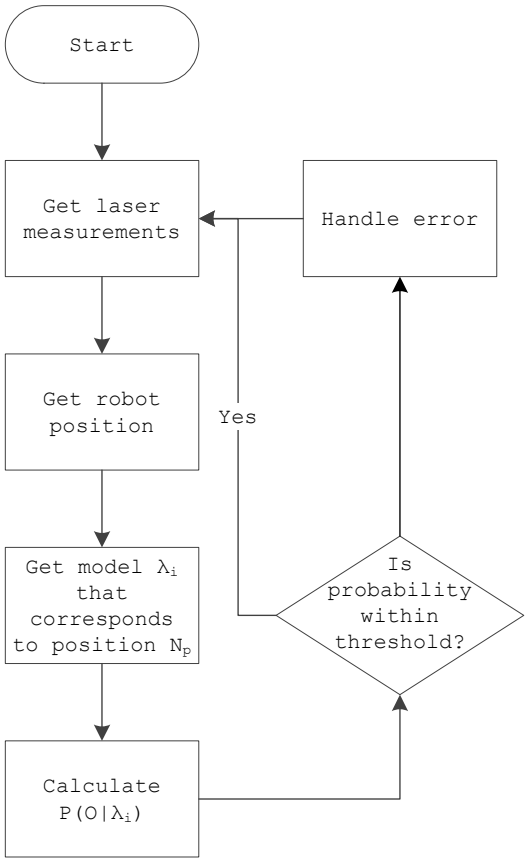


Figure A.6: *HMM decoding algorithm.*

4.1 HMM Decoding

The purpose of HMM decoding is to calculate the probability of the HMM given a measurement sequence from the laser rangefinder. If the score is high, then the measurements correspond well to the training data. Hence, the measurements in this case indicate a normal situation. If, on the other hand, the score is low, the measurements do not correspond well with the training data and hence it is assumed that one or more unknown obstacles have entered the workspace.

The employed decoding algorithm does not find the probability of the best state sequence through the model in Fig. A.4. The decoding is only concerned with the discovery of laser measurements that do not match the training data, such that this information can be used by the robot controller to take appropriate action, see algorithm Fig.A.6. However, if the setup or application should require identification

of the segment in which the error occurred, the decoding method could be expanded such that the probability contribution for each segment is stored and evaluated.

5 Experimental results

A laser rangefinder sensor LMS133-10100 from SICK is used for 2D length measurements outputting a maximum length of $20m$ $\pm$30mm associated with an angle in a range of 180^o. Combined with the HMM and ES methodology, the rangefinder is set up to detect any object that is not supposed to be in the workspace of the robot. In the experiments described in this paper, an unknown object at a random point in time enters the workspace. The experimental setup is shown in Fig. A.7 and a typical measurement from the rangefinder is shown in Fig. A.8. The range from 120^o to 180^o is where the robot cell wall ends, and the laser distance measurements are reflections from objects and walls in the laboratory.

5.1 Expert System

Fig. A.9 shows the workspace with the different objects. The two objects to the right and the tool have been removed by the filter. The red stars represent an unexpected obstacle. Because of potential system errors the filter removes additional points in the range $\pm 0.5^o$. Further, because the accuracy of the rangefinder is approximately $\pm 30mm$, the ES creates the filter at the object distance subtracted by the rangefinder accuracy of $30mm$. An example of filtering is shown in Fig. A.3 (b). The black dots in the figure represent the object, the green stars the filter and the pink square the middle of the filter (for illustration purposes only). Table A.1 shows the results obtained with the ES. The 12 scenarios are: i) No objects, ii) known objects A and iii) the combination of known objects (A) and unknown (F) obstacles inside the workspace. The tests are repeated for the 4 different robot positions along the tool path shown in Fig. A.1 and the ES is able to correctly detect unknown objects in all the 12 test cases.

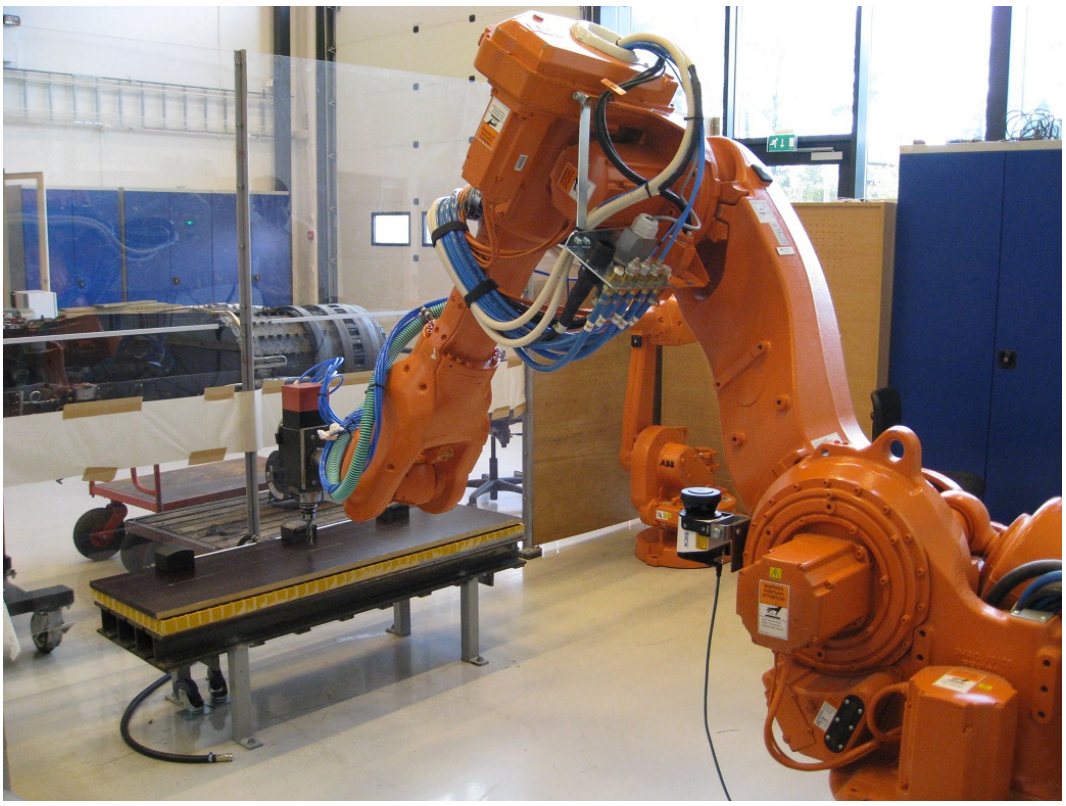


Figure A.7: *Experimental setup, consisting of an ABB IRB6600-175kg-2.55m robot, obstacles and a laser range finder.*

5.2 Hidden Markov Model

Since there is one HMM for each position of the robot along the path in Fig.A.1, the notation λ_i is introduced, where $i \in \{1, \cdots, N_p\}$ and N_p is the total number of positions along the path. In the experiments, a total of $N_p = 167$ positions are used. In each position i, 192 observations ($\boldsymbol{O_t}$, $t \in \{1, 2 \cdots, T\}$) are used to train the corresponding HMM λ_i for that robot position. In addition, 12 observations in each position are used as a test set to verify the scores of the HMMs. Each model has its own threshold value for model match decision. Tables A.2-A.5 show the model probabilities for the test sets in the 4 different robot positions. For each robot position, only the corresponding λ_i is evaluated. Note that the $\log[\cdot]$ scores of $P(\boldsymbol{O}|\lambda_i)$ are used, since the probabilities become small when the HMMs use large observation vectors, in these experiments the observation vector $\boldsymbol{O_t}$ contains 61 elements. The higher the score $\log[P(\boldsymbol{O}|\lambda_i)]$, the better is the match between

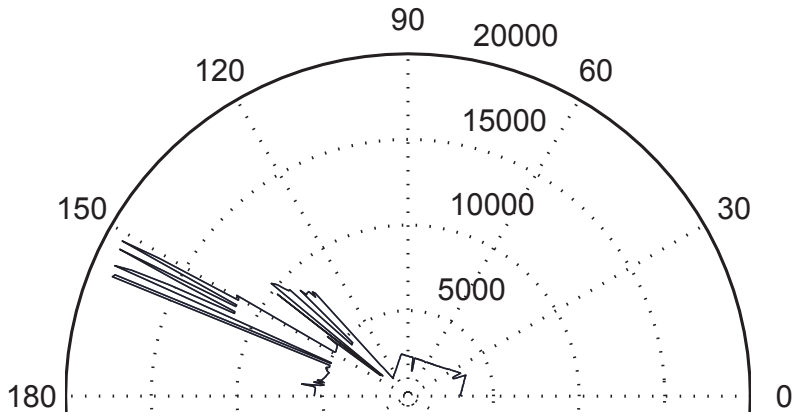


Figure A.8: *Typical measurement from the rangefinder in polar coordinates. The area of interest is close to the origin and with angles less than* 120^o.

Table A.1: *Expert system performance in the 4 test positions, in total* 12 *test cases.*

Test set	Pos 1, λ_1	Pos 2, λ_2	Pos 3, λ_3	Pos 4, λ_4
No object	✓	✓	✓	✓
Objects A	✓	✓	✓	✓
Object $A+F$	✓	✓	✓	✓

the model and the experiments. Table A.2 shows that the model score is in the approximate range from -72 to -69 when the test data is similar to the training data. When objects $A+F$ are present in the workspace, the test data are different from the training data, hence the model scores drop to the approximate range -103 to -105. Hence, a threshold value for detection of unknown objects at position 1 should lie in the range -80 to -90, depending on the acceptable level of detection and false alarms. For example, if the HMM threshold is defined at -80, the method has a high chance of detecting changes from the training set (unwanted objects), but the probability of false alarms could be large. However, if the threshold is defined higher, the method may not detect all unexpected objects, while the probability of false alarms will be reduced. The final selection about threshold values must be defined by the operator, case-by-case.

Tables A.3-A.5 for three other robot positions show similar results compared to

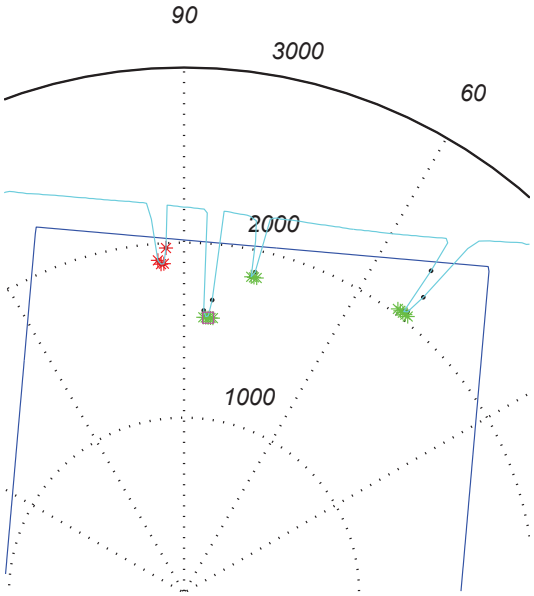


Figure A.9: *Expert System: Filtering of known objects (green) and detection of (red) obstacle.*

Table A.2: *Scores* $\log[P(\mathbf{O}_{i,j}|\lambda_1)]$ *with 12 test sets* $i \in \{1,\cdots,4\}$ *and* $j \in \{1,\cdots,3\}$.

Test set i	1	2	3	4
No object, $\mathbf{O}_{i,1}$	-71.70	-71.70	-71.78	-71.81
Objects A, $\mathbf{O}_{i,2}$	-69.44	-69.43	-69.60	-69.57
Object $A+F$, $\mathbf{O}_{i,3}$	-103.51	-103.21	-104.43	-103.77

Table A.2, but the threshold values must be selected differently. For Table A.3 the threshold values should lie in the approximate range -105 to -115, for Table A.4 in the approximate range -70 to -90 and for Table A.5 in the approximate range -77 to -83.

As shown in the experimental results, the HMM approach is able to correctly classify all the different scenarios in the 12 test set observations. As seen in Tables A.2-A.5, the threshold values for detecting unwanted objects must be set differently for each robot position. In the first three robot positions, there is a relatively large gap between the model scores for known objects and unknown obstacles (Tables A.2-A.4), and it is therefore easy to define the model score thresholds. The fourth po-

Table A.3: *Scores* $\log[P(\mathbf{O}_{i,j}|\lambda_2)]$ *with 12 test sets* $i \in \{1,\cdots,4\}$ *and* $j \in \{1,\cdots,3\}$.

Test set i	1	2	3	4
No object, $\mathbf{O}_{i,1}$	-91.23	-95.95	-95.98	-95.96
Objects A, $\mathbf{O}_{i,2}$	-100.18	-100.25	-100.24	-100.32
Object $A+F$, $\mathbf{O}_{i,3}$	-124.67	-127.37	-125.17	-126.41

Table A.4: *Scores* $\log[P(\mathbf{O}_{i,j}|\lambda_3)]$ *with 12 test sets* $i \in \{1,\cdots,4\}$ *and* $j \in \{1,\cdots,3\}$.

Test set i	1	2	3	4
No object, $\mathbf{O}_{i,1}$	-60.73	-60.64	-60.77	-60.73
Objects A, $\mathbf{O}_{i,2}$	-63.69	-63.65	-63.55	-63.64
Object $A+F$, $\mathbf{O}_{i,3}$	-101.03	-100.42	-101.99	-101.18

sition (Table A.5), however, has a narrower gap between the model scores which indicates that it is more difficult to distinguish between known and unknown objects being present in the workspace.

Fig. A.10 shows all the learnt mean-distance parameters, $\boldsymbol{\mu}_j$, inside the HMM for 6 state transitions, keeping in mind that each of these states defines a 30^o range of the laser measurements. Note that there are two different sets of mean-distance parameters shown in Fig. A.10. These correspond to two different state transition paths in the HMM, one corresponding to no objects and the other corresponding to known objects. One significant benefit of defining the HMM states, transitions and observations as in this paper, is the fact that the learnt parameters after re-estimation with the Baum-Welch algorithm have physically meaningful interpretations such as illustrated in Fig. A.10.

Table A.5: *Scores* $\log[P(\mathbf{O}_{i,j}|\lambda_4)]$ *with 12 test sets* $i \in \{1,\cdots,4\}$ *and* $j \in \{1,\cdots,3\}$.

Test set i	1	2	3	4
No object, $\mathbf{O}_{i,1}$	-72.27	-72.33	-72.18	-72.28
Objects A, $\mathbf{O}_{i,2}$	-74.82	-74.77	-74.73	-74.58
Object $A+F$, $\mathbf{O}_{i,3}$	-85.20	-85.64	-86.20	-85.72

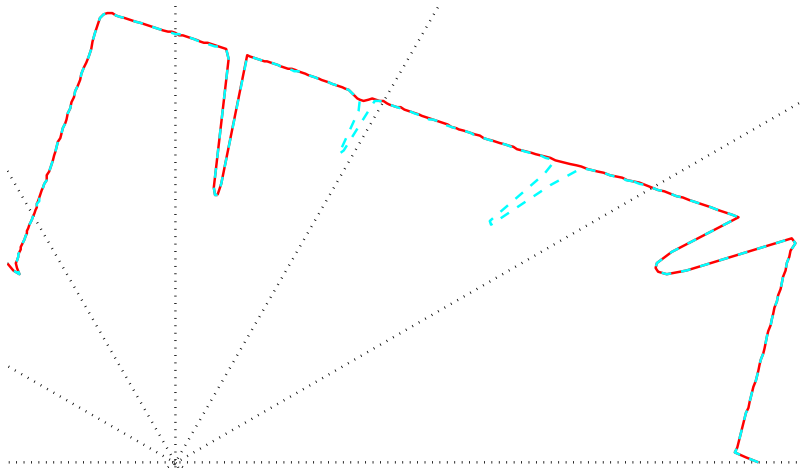


Figure A.10: *Mean distance after re-estimation. Red: Tool and no-objects, Cyan: Tool and objects.*

6 Conclusions and Future Work

This paper has presented a comparison between two approaches (Expert System and Hidden Markov Models) for detecting unknown objects inside the workspace of a robot. The ES quickly and accurately calculates any deviation to the normal situation as long as the model of the environment is held at a reasonable complexity level, but the method is sensitive to small adjustments in the environment. To manually input the object positions and angles relative to the laser can turn out to be a time-consuming task. Such is the case when many objects are placed inside the workspace and manual input of these objects could turn out to be unfeasible.

These are some of the reasons why HMM is considered: The HMM setup is less sensitive to application setup, therefore sensors can be positioned without knowledge of the distance to robot centre or angle relative to the robot centre. With the flexibility in position and angles, relatively short implementation time could be achieved. The implimentation would require thresholds provided by the operator and an auto generated HMM. If a change has been made to the environment, new HMMs must be trained to account for the changes. Since an HMM is providing a confidence level, the robot speed can be decreased if the score approaches the predefined threshold level. If the HMM score drops below the threshold, a signal is raised and sent to the robot controller. Another significant benefit of HMMs is the

fact that the learnt parameters provide physical meaningful results, as demonstrated with the experiments in this paper. The main drawbacks of the HMM approach are perhaps the relatively high development time required to implement and verify the algorithms as well as the high calculation time compared to the ES solution.
One could argue that the problem defined in this paper could be solved simply by comparing laser rangefinder measurements with the same measurements taken in the normal situation. If these measurements differ more than by the accuracy of the sensor, an unknown object would be detected. However, such a solution would only work if the normal situation could be described by a single measurement. In most industrial scenarios, the normal situation consists of a number of different objects in different sensor segments. In such cases the HMM approach provides a flexible and robust framework which could also be easier to implement compared to an ES.
One possible extension of the work presented in this paper could be a combination of the ES with CAD model and the HMM approach in order to reduce the amount of training data required by HMMs. Another possible extension could be to interpolate HMM data between robot positions to further reduce the amount of measured laser data. Finally, outdoor tests will be conducted in order to verify the performance of the algorithms when exposed to different environmental conditions, such as outdoor lighting, rain, fog, snow, etc.

Acknowledgment

Knut Berg Kaldestad acknowledges funding from ABB and the Norwegian Research Council through project number 193411/S60.

REFERENCES

Aarts, E. H. L. and Marzano, S. (2003). *The New Everyday: Views on Ambient Intelligence*. 010 Publishers, Rotterdam.

Anisi, D., Gunnar, J., Lillehagen, T., and Skourup, C. (2010). Robot Automation in Oil and Gas Facilities: Indoor and Onsite Demonstrations. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 4729–4734, Taipei, Taiwan.

Anisi, D., Persson, E., Heyer, C., and Skourup, C. (2011). Real-World Demonstration of Sensor-Based Robotic Automation in Oil & Gas Facilities. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, San Francisco, California.

Fink, G. A. (2007). *Markov Models for Pattern Recognition: From Theory to Applications*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Fulgenzi, C., Spalanzani, A., and Laugier, C. (2009). Probabilistic motion planning among moving obstacles following typical motion patterns. In *Proc. 2009 IEEE/RSJ Intl. Conf. Intelligent Robots and Systems*, pages 4027 –4033.

Hovland, G. E. and McCarragher, B. J. (1999). Hidden Markov Models as a Process Monitor in Robotic Assembly. *Modeling, Identification and Control*, 20(4):201–223.

Rabiner, L. (1989). A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, 77(2):257 –286.

Wolf, D., Sukhatme, G., Fox, D., and Burgard, W. (2005). Autonomous Terrain Mapping and Classification Using Hidden Markov Models. In *Proc. 2005 IEEE Intl. Conf. Robotics and Automation*, pages 2026 – 2031.

Zhu, Q. (1990). A Stochastic Algorithm for Obstacle Motion Prediction in Visual Guidance of Robot Motion. In *Systems Engineering, 1990., IEEE Intl. Conf. on*, pages 216 –219.

Paper B

CAD-Based Training of an Expert System and a Hidden Markov Model for Obstacle Detection in an Industrial Robot Environment

Knut B. Kaldestad, Geir Hovland and David A. Anisi

This paper has been published as:

Kaldestad, K., Hovland, G., and Anisi, D. CAD-Based Training of an Expert System and a Hidden Markov Model for Obstacle Detection in an Industrial Robot Environment. *In IFAC Intl. Conf. on Automatic Control in Offshore Oil and Gas Production.* Trondheim, Norway, 2012a.

CAD-Based Training of an Expert System and a Hidden Markov Model for Obstacle Detection in an Industrial Robot Environment

Knut B. Kaldestad*, Geir Hovland* and **David A. Anisi
*Department of Engineering
Faculty of Engineering and Science, University of Agder
Jon Lilletunsvei 9, 4879 Grimstad, Norway.
**Department of Technology & Innovation
Division of Process Automation, ABB
Norway

Abstract — **Deploying industrial robots in harsh outdoor environments require additional functionalities not currently provided. For instance, movement of standard industrial robots are pre-programmed to avoid collision. In dynamic and less structured environments, however, the need for online detection and avoidance of unmodelled objects arises. This paper focus on online obstacle detection using a laser sensor by proposing three different approaches, namely a CAD-based Expert System (ES) and two probabilistic methods based on a Hidden Markov Model (HMM) which requires observation based training. In addition, this paper contributes by providing a comparison between the CAD-based ES and the two versions of the HMM, one trained with real sensor data, and one where virtual sensor data has been extracted from the CAD-model and used during the training phase.**

Keywords — Obstacle Detection, Industrial Robots, Expert System, Hidden Markov Model.

1 Introduction

Industrial robots have been used since the 1960s for solving repetitive, routine, heavy and dangerous tasks, such as coating, painting, pick and place, welding, as-

sembly and inspection (Nof, 1999). The traditional industrial robot works in structured, indoor environments and does not stop its process unless its safety switch circuit is broken or it's stopped by the operator. Standard industrial robots are preprogrammed such that the robot path avoids any obstacles in its vicinity, and does therefore not need to know or have any awareness of these objects' locations.

Historically, the main driver for using robots within manufacturing industries has been to achieve better quality and productivity by increased automation. In most industries, this is still true today. Recently, however, the need for deploying industrial robots in rather unstructured outdoor environment has arisen. Within the oil and gas industry, for instance, (Anisi et al., 2010, 2011) the applications generally stand out from other industries as the main driver has been to automate tasks that have been difficult or even impossible for people to undertake based on Health Safety and Environment (HSE) issues. Applying robotics in this way has resulted in an improvement in HSE, but often with an associated dip in production. Although this is contradictory to the general goal of automation, work is now being done towards maintaining focus on HSE and at the same time improving the efficiency and profitability of the facilities.

Today, industrial robots are not able to work independently in harsh and unstructured outdoor environments, which involves detection and avoidance of unmodeled objects. The work presented in this paper takes a step in that direction by focusing on online obstacle detection using two different approaches. The first approach is a CAD-based Expert System (ES). The ES has initial knowledge about the environment, and therefore knows what objects are allowed and at which positions they are allowed. The second approach is using a Hidden Markov Model (HMM). This is a probabilistic method which relies on training of observation of the environment. Alternative methods could, for example, be Bayesian decision networks and support vector machines. The alternative methods are however beyond the scope of this paper, hence, the focus is on performance and limitations of the ES and HMM methods.

Related work describing the combination of online obstacle detection and a laser rangefinder in an industrial robotics environment are hard to find. Most of the related research has focused on mobile robots and different applications such as lo-

calisation and mapping. In research described by (Wolf et al., 2005), the authors investigate a method for mobile robots to detect the state of the terrain. The robots are equipped with a laser range scanner and able to produce a map in 3D when driving forward. To be able to determine whether parts of the scanned area belong to one of two states – flat terrain or rougher terrain – the authors use an HMM approach. The state estimation is based on comparison of successive scans. The approach in (Wolf et al., 2005) differs from this paper in a few ways. The approach in this paper focuses on obstacle detection and not mapping. In addition, the presented method requires only one scan to determine the state, a larger number of states is used, and the probability of the observations of many models are calculated and evaluated.

The Little Helper described by (Hvilshøj et al., 2009) is an industrial robot mounted on a mobile robot platform and provides another point of reference. The project purpose is to devise and develop an industrially usable mobile robot concept. The robot operates in a semi-structured indoor environment, where it picks up objects that are placed at different positions and are located by vision sensors. One of the outlined scenarios: A work station calls on the robot, the robot moves to the workstation and performs a manipulation task with the robot arm. When the task is complete, the robot releases the task and moves away from the work station. This approach is similar to the approach described in this paper in the way of handling environments that are not fully structured, and using a industrial robotic arm for manipulation tasks. The work presented in this paper goes beyond the one conducted for the Little helper by focusing on online obstacle detection and in particular comparing CAD-model-based ES with HMM.

The work presented in this paper is an extension of the work described in (Kaldestad et al., 2012). The main extension is the inclusion of a CAD-file to define the ES. The HMM is then trained on generated data from the ES, which are indirectly generated from the CAD-file. If accurate maps of the environment are available, the presented method eliminates the need for measurement based training of the HMM.

The remaining of this paper is organized as follows: Section 2 presents the problem formulation, Section 3 explains the ES while Section 4 introduces the two HMM approaches. Finally, Section 5 presents the experimental results and Section 6 concludes the work.

2 Problem Formulation

In industrial environments, the robot movement is typically restricted to its pre-programmed trajectories in combination with static or temporary world zones preventing the robot's tool center point (TCP) to either leave or enter the manually defined world zones. However, as the demand for handling more dynamic environments increases, so does the demand for planning trajectories dynamically. To this end, this paper focuses on the problem of on-line detection of unknown and unmodeled objects which constitutes the initial part of dynamical trajectory planning.

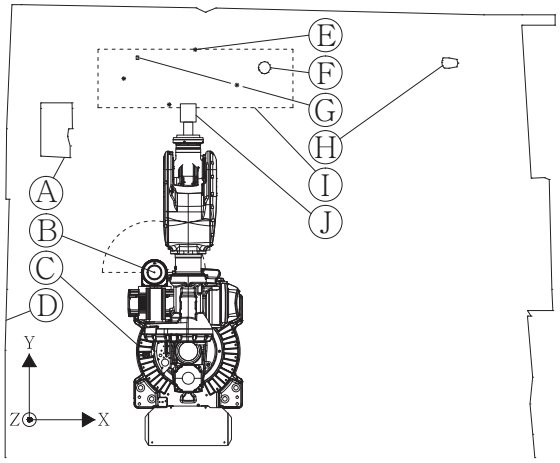


Figure B.1: *A schematic overview of the robot cell.*

The laboratory setup depicted in Fig. B.1 gives a schematical overview of the robot cell. The robot (C) is manipulating objects on the work bench (I) with a tool located in the tool holder (J). Objects (A) and (H) are static objects in the robot cell and together with the wall (D) these are a part of the CAD-file map. Furthermore, objects (F) and (G) are objects that are allowed in the robot cell at these positions and are part of the CAD-file map that includes objects. The laser (B) continuously scans $0^o - 180^o$ in the x-y plane at a height z. While manipulating, robot movement will rotate and translate the laser which is mounted on the robot's first axis. The four black dots on the workbench such as (E), are the positions where laser data have been collected for the work described in this paper.

In the work described in this paper, two different approaches for on-line obstacle detection will be investigated, where the Expert System will be described in the following section.

3 Expert System

The Expert System (ES) described in this paper relies upon a description of the environment (map and objects) in terms of a CAD-model. When a CAD-model of the environment exists, the distances from the centre of the laser sensor to any shape described by the model can be calculated using geometrical relations as will be detailed below. Then by comparing this to the distances measured on-line by the laser, it is possible to detect unmodeled obstacles entering the robot workspace. However, a straightforward comparison between the measured and the modeled distance would lead to an unacceptably high rate of false object detection alarms. To remedy this behaviour, systematic sensor measurement errors ($\pm$30 mm), as well as inaccuracies in the provided CAD-model will be taken into account before triggering alarms.

More precisely, with an estimated CAD-model accuracy of $\pm$10 mm, the laser measurements are allowed to deviate within the threshold of $\varepsilon = \pm 10$ mm) before triggering the alarm indicating obstacle detection. Despite this threshold, the cases when the laser measurements deviate more than ε is observed typically when the laser beam hits near the edge of an object. In this paper, such measurements are recognized by being considerably greater than that of the modeled distance to the object. This non-predictable behaviour of the laser measurements, was handled by extending the functionality of the ES in accordance with the bottom two boxes in Fig. B.2, which shows the overall program design.

As previously mentioned, the ES needs to calculate the modeled distances from the laser centre to the environment described by the CAD-model. For this purpose, the orientation of the robot's first axis (upon which the laser sensor is mounted) and the position and orientation of the robot tool need to be known. This data is readily available on most industrial robot controllers of today. However, the ES also requires knowledge about the position of the laser sensor relative to the robot base

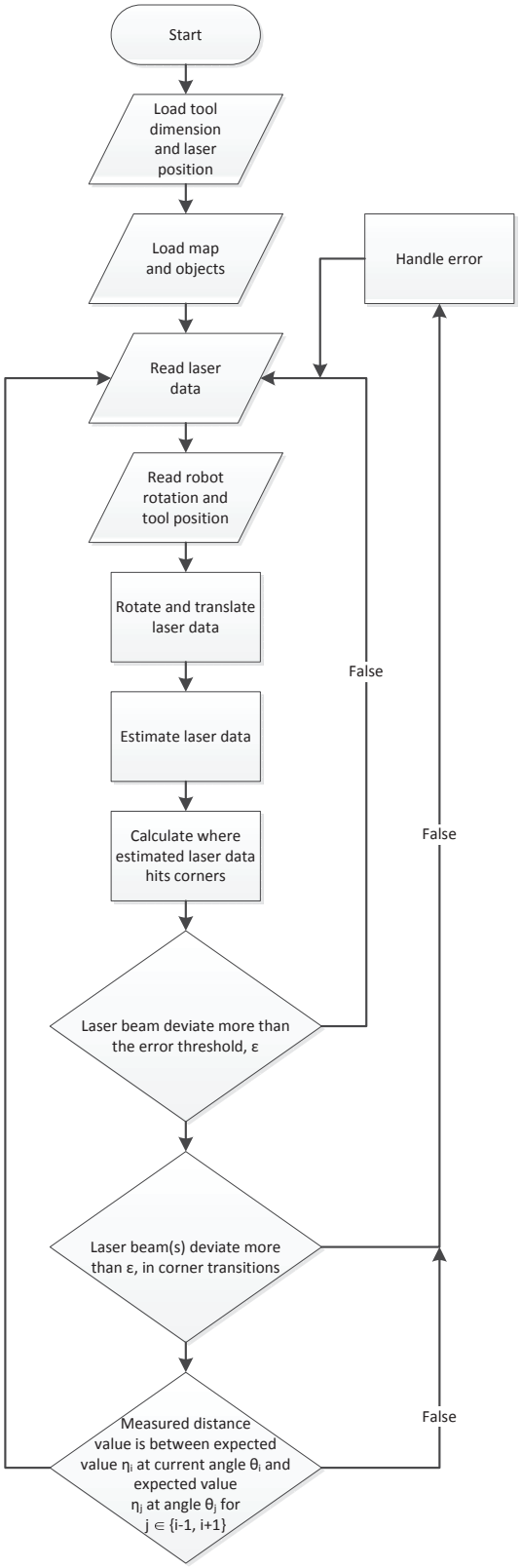


Figure B.2: *The ES program design.*

Figure B.3: *FARO laser tracker used to calibrate the position of the laser scanner relative to the base of the robot.*

coordinate system. In practice, this quantity is most often not known or measureable directly with sufficient accuracy. Therefore, finding the position of the laser relative to the robot base is performed as follows.

To allow accurate calculation of the relative position between the laser scanner and the robot base, a FARO laser tracker was utilized. It measures points in 3D space with a worst case accuracy of 18 µm + 3 µm/m. As an example, the accuracy 5 m out from the device will be 18 µm + 3 µm/m$\times$5 m = 33 µm. The laser tracker was positioned in the front-right of the robot (see Fig. B.3), and a local coordinate system was created, with x-axis and y-axis direction in accordance with Fig. B.1, and z-axis perpendicular to the x-y-plane to define a right-handed coordinate system. Next, a laser tracker target which reflects the laser beam was centrally aligned on top of the laser sensor. Then, the first axis of the robot was rotated at angles θ_1 and θ_2 degrees. Measurements were conducted by the FARO laser tracker at each of the points, enabling us to extract the radius defining a circle centered at the origin of the robot base coordinate system, and passing through the measurement points. As shown in Fig. B.1, this radius equals the distance between the laser to the centre of

the robot. The angle offset from the centre of the robot to the laser was found by maximizing the y-distance; this is when the laser position will be 90^o relative to the robot centre. At the point where y was maximized, the angle rotation of the robot's first axis was read from the robot controller. This equals the negative angle offset of the laser.

Having calculated the exact position of the laser sensor in the environment, the distance from the sensor to the closest point in the environment is calculated as follows. The CAD-model describing the environment (map and obstacles) consists of a number of corner points associated with each object. Each two consecutive points belonging to the same object, are then used to define a line. Then, for each ray centred at the laser position and defined by the angle $\theta \in [0, 180]$ degrees from the x-axis, the intersection point between the ray and all these line segments are calculated. To finally arrive at the distance to the closest point in the environment, the intersection with the lowest length value is recognized. The method is repeated until a closest intersection for all angles in $\{0, 0.5, \cdots, 180\}$ are extracted.

4 Hidden Markov Model

The following notation will be in accordance with (Rabiner, 1989). The HMM approach provides a confidence value for laser length data matching the model $\lambda = (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\pi})$. Here, $\boldsymbol{A}$ is a 12×12 transition matrix (see Fig. B.5), $\boldsymbol{B}$ is the observation matrix consisting of the mean and variance – both of size 12×61. Finally $\boldsymbol{\pi}$ is the 12×1 initial state distribution vector. The observation matrix $\boldsymbol{B}$ contains six sets of measured data, one for each 30^o segment. The arrows in Fig. B.5 show the allowed state transitions.

Two different sensor tranining data sets are used to estimate the HMM, λ; one uses measured laser data, the other uses estimated (virtual) data extracted from the CAD-model (cf. Fig. B.2). The second approach will be an advantage if real sensor data is not available before the system is implemented. It would not only allow for smart scheduling of the computation time in advance of system deployment, but it will also allow the system to instantly begin operating without making numerous measurement of the normal situation where the robot is located. To be able to

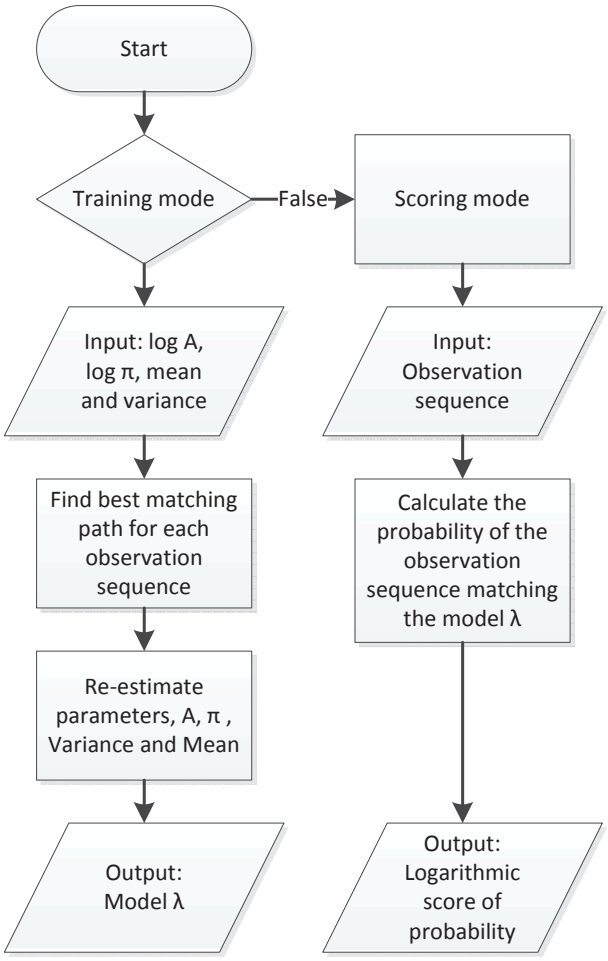


Figure B.4: *Flowchart of the HMM training and scoring algorithm.*

represent laser measurements, noise that is similar to the standard deviation of the laser sensor is applied to the virtual data. Further, the signal mean and variance are calculated for use in the $\boldsymbol{B}$ matrix.

5 Experimental Results

5.1 Expert system

The ES, being an extension of the one presented in (Kaldestad et al., 2012), has increased complexity by adding a map of the environment where previously a filter

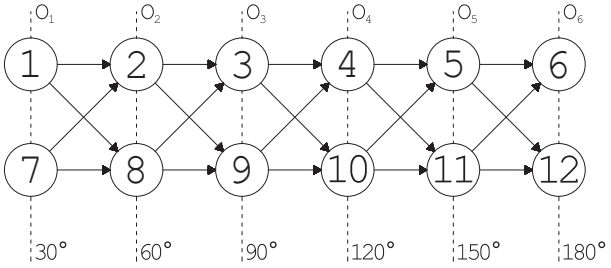


Figure B.5: *Illustration of the HMM transition matrix,* $\boldsymbol{A}$.

was used to filter out anything but the tool. If any unfiltered object entered the area, an error would be thrown. As a result of adding a map, the system is more error sensitive. This sensitivity is especially evident when the laser beam hits in a corner location of an object. This situation often leads to a wrong distance measurement by the laser. There is currently no known method of accurately correct for these situations and as described earlier, beams in corner regions are filtered out. In the experiments the filtering angle θ is 1^o, and the systematic error of the system, η, is set to 40 mm. Fig. B.6 shows the ES detecting an object that is not a part of the map, and the system responds by throwing an error.

Table B.1 shows the results of the 12 test cases in the experimental studies with the ES. It is notable that the ES has classified all scenarios correctly in all four positions. The "no object" and "objects" columns constitute the normal scenarios, i.e., cases when no unexpected obstacle is present. The objects in question are (G) and (F) from Fig. B.1. The "objects + obstacle" column represents the abnormal situation when an arbitrary obstacle is placed in an arbitrary location in the robot cell.

Table B.1: *Expert system performed succesfully in all four test positions, in total 12 test cases.*

Test set	No object	Objects	Objects + obstacle
Pos 1, λ_1	✓	✓	✓
Pos 2, λ_2	✓	✓	✓
Pos 3, λ_3	✓	✓	✓
Pos 4, λ_4	✓	✓	✓

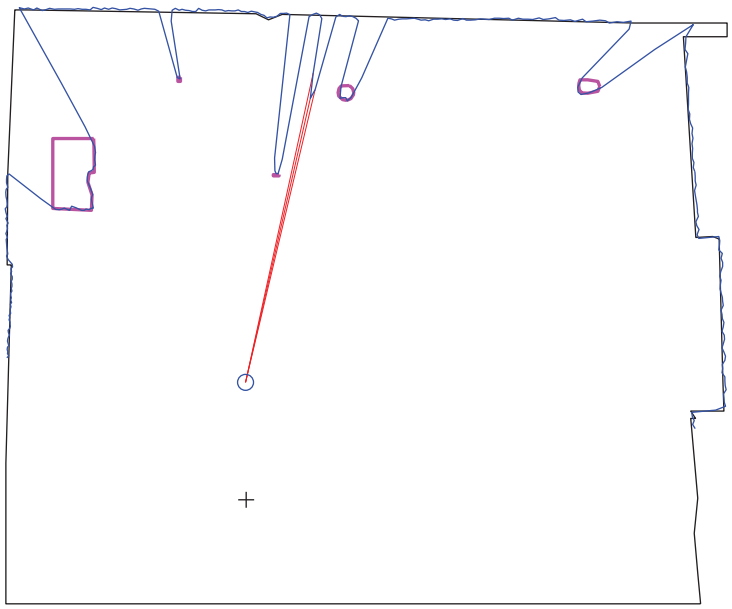


Figure B.6: *The blue line is reflections from a laser measurement, an object which is not a part of the CAD-model is detected and is represented by red beams.*

5.2 Hidden Markov Models

The HMM results can be found in Table B.2–B.5 where each table presents the scores based on twelve different observations at one of the four positions marked (E) in Fig. B.1. From the tables, the log score for the "no object" and "objects" scenarios, which constitute the normal training cases, are quite similar in all the four tables. As for the abnormal "objects + obstacle" case goes, since the obstacle is not a part of the model λ, it is expected that a model score with lower probability is expected. The results show that the score for "objects + obstacle" is significantly higher than the trained models ("no object" and "objects").

For the virtual data to be able to function as a means for training the HMM, the parameter systematic error must be tuned. Fig. B.7 shows that the laser measurements are not spread uniformly across the systematic error of ± 30 mm. The measurements are rather taking a normal distributive shape. Table B.6 shows the standard deviation of the four different measurements presented in Fig. B.7.

The algorithm for training the HMM creates random normal distributed values for the estimated measurement. The reason for the random variables is because the

Table B.2: *Scores* $-\log[P(\mathbf{O}_{i,j}|\lambda_1)]$ *with 12 test sets* $i \in \{1,\cdots,4\}$ *and* $j \in \{1,\cdots,3\}$.

Test set	No object	Objects	Objects + obstacle
1	1868	1862	2005
2	1867	1862	2005
3	1868	1863	2005
4	1868	1862	2006

Table B.3: *Scores* $-\log[P(\mathbf{O}_{i,j}|\lambda_2)]$ *with 12 test sets* $i \in \{1,\cdots,4\}$ *and* $j \in \{1,\cdots,3\}$.

Test set	No object	Objects	Objects + obstacle
1	1909	1912	2046
2	1909	1912	2048
3	1909	1912	2046
4	1909	1912	2047

HMM training must have a variety of observations to perform the training on. The effect of changing the systematic error is shown in Fig. B.8. The top part of the figure show that by increasing the systematic error the probability of a HMM trained on virtual data increases for a laser observation with object in the robot cell, while it is more or less steady for a virtual data observation. The top part of the figure also shows that the difference between a laser observation with object and a laser observation with object and obstacle gets smaller as the value of the systematic

Table B.4: *Scores* $-\log[P(\mathbf{O}_{i,j}|\lambda_3)]$ *with 12 test sets* $i \in \{1,\cdots,4\}$ *and* $j \in \{1,\cdots,3\}$.

Test set	No object	Objects	Objects + obstacle
1	1870	1889	2066
2	1870	1890	2066
3	1870	1890	2069
4	1870	1889	2067

Table B.5: *Scores* $-\log[P(\mathbf{O}_{i,j}|\lambda_4)]$ *with 12 test sets* $i \in \{1,\cdots,4\}$ *and* $j \in \{1,\cdots,3\}$.

Test set	No object	Objects	Objects + obstacle
1	1865	1868	2672
2	1864	1869	2664
3	1865	1869	2678
4	1865	1868	2665

Table B.6: *Standard deviations from Fig. B.7.*

Upper left	Upper right	Lower left	Lower right
7.527	7.562	8.346	9.4244

error increases. This difference is clearer in the bottom part of the figure, and is shown by the red line. It is important to have this difference sufficiently high, as this difference tells how well an accepted HMM score (no obstacle in the robot cell) can be differentiated from an HMM score with obstacle in the robot cell. The black vertical line shows the chosen systematic error at 8.55 mm, which corresponds to the mean of the standard deviation from Table B.6.

Table B.7: *Scores based on virtual data* $-\log[P(\mathbf{O}_{1,j}|\lambda_k)]$ *with 20 test sets* $j \in \{1,\cdots,3\}$ *and* $k \in \{1,\cdots,4\}$.

Test set	No object	Objects	Objects + obstacle
λ_1 (VD)	1881	1880	
λ_1 (LD)	11646	11930	12451
λ_2 (VD)	1883	1882	
λ_2 (LD)	11472	12110	12530
λ_3 (VD)	1882	1883	
λ_3 (LD)	11121	11689	12573
λ_4 (VD)	1885	1885	
λ_4 (LD)	11751	12239	12427

Table B.7 shows the probability scores for virtual data and laser data for the models

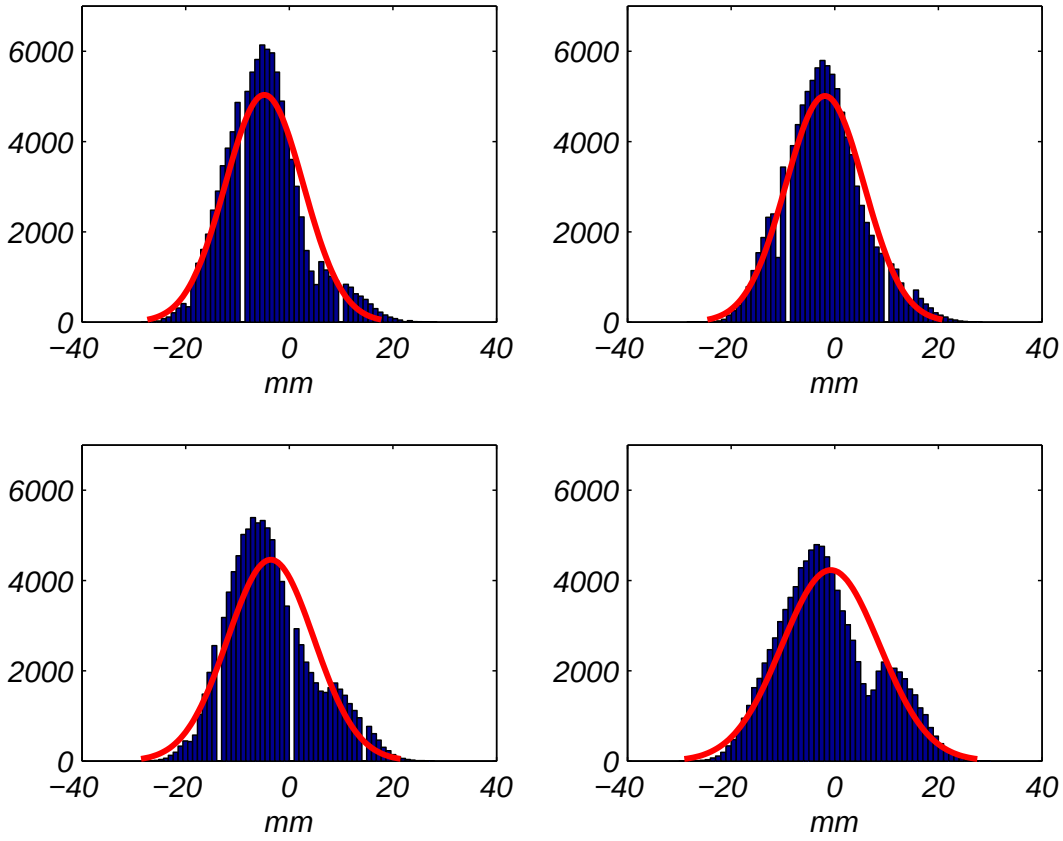


Figure B.7: *Systematic error of the laser, for four arbitrary angles. Each histogram is created from 100 000 laser measurements.*

λ_k that are trained from virtual data. The result shows that it is possible to distinguish the two allowed situations ("no object" and "objects") from a situation where an obstacle is present.

6 Conclusions and Future Work

The ES algorithm is computationally efficient, and is suitable in real time applications. Furthermore, it could provide the system with coordinate location of the obstacle.

The ES is highly dependent on accurate input values (such as a map and the position of the robot), and there are many of these values that can be improved. First, the map of the environment could be measured more accurately by use of, e.g., the

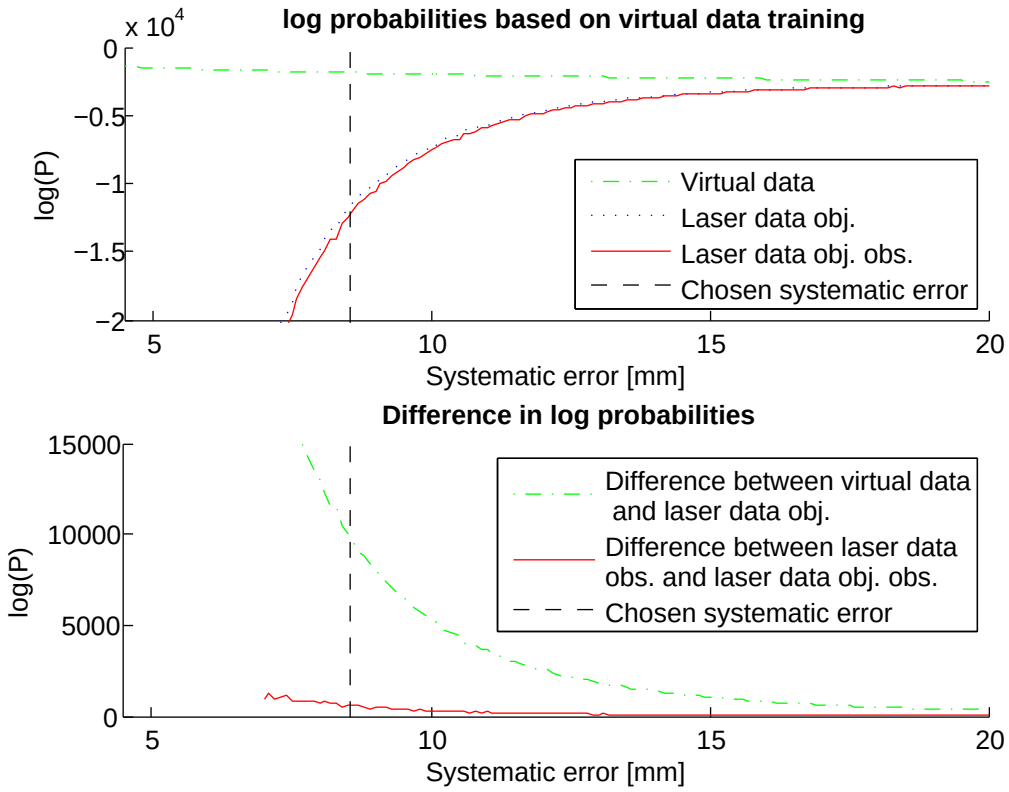


Figure B.8: *Model score versus systematic error.*

FARO laser tracker seen in Fig. B.3. Second, a small misalignment of the sensor could have large impact on the measurements, in particular at long distances. This might cause false classification, especially in the CAD-based ES.

An improvement of the problem experienced when measuring corners could be to incorporate a second laser, positioned at another location on the robot.

The possibility of training the HMM directly from the virtual data is an advantage; time can be saved and there is no need for acquiring large amounts of data after deployment of the robot. It is possible to calculate the HMMs in advance, enabling the system to be up and running at the time of deployment.

One limitation of the HMM is the computational requirements for the training and in the current version of the algorithm, it could not provide information of the obstacle location.

There are apparently some challenges with applying a CAD-based map of an un-

structured environment, one is how to update a map of an environment that continuously changes. The idea behind the approach presented in this paper, is that large parts of the map will be static. Some parts, however, will be dynamic, but the update frequency of the environment will be so low that on-line updates of the map would be feasible. Dynamic mapping is, however, beyond the scope of this paper and would be a direction of future research.

Acknowledgment

Knut Berg Kaldestad acknowledges funding from ABB and the Norwegian Research Council through project number 193411/S60.

REFERENCES

Anisi, D., Gunnar, J., Lillehagen, T., and Skourup, C. (2010). Robot Automation in Oil and Gas Facilities: Indoor and Onsite Demonstrations. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 4729–4734, Taipei, Taiwan.

Anisi, D., Persson, E., Heyer, C., and Skourup, C. (2011). Real-World Demonstration of Sensor-Based Robotic Automation in Oil & Gas Facilities. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, San Francisco, California.

Hvilshøj, M., Bøgh, S., Madsen, O., and Kristiansen, M. (2009). The mobile robot little helper: Concepts, ideas and working principles. In *Emerging Technologies Factory Automation, 2009. ETFA 2009. IEEE Conference on*, pages 1 –4.

Kaldestad, K., Hovland, G., and Anisi, D. (2012). Obstacle Detection in an Unstructured Industrial Robotic System: Comparison of Hidden Markov Model and Expert System. In *10th IFAC Intl. Symposiums on Robot Control, Submitted*, Dubrovnik, Croatia.

Nof, S. (1999). *Handbook of industrial robotics*. Number v. 1 in Electrical and electronic engineering. John Wiley.

Rabiner, L. (1989). A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, 77(2):257 –286.

Wolf, D., Sukhatme, G., Fox, D., and Burgard, W. (2005). Autonomous Terrain Mapping and Classification Using Hidden Markov Models. In *Proc. 2005 IEEE Intl. Conf. Robotics and Automation*, pages 2026 – 2031.

Paper C

3D Sensor-Based Obstacle Detection Comparing Octrees and Point clouds Using CUDA

Knut B. Kaldestad, Geir Hovland and David A. Anisi

This paper has been published as:

Kaldestad, K., Hovland, G., and Anisi, D. 3 D Sensor-Based Obstacle Detection Comparing Octrees and Point clouds Using CUDA. *In Modeling, Identification and Control.* 33.4 (2012): 123-130.

3D Sensor-Based Obstacle Detection Comparing Octrees and Point clouds Using CUDA

Knut B. Kaldestad*, Geir Hovland* and **David A. Anisi
*Department of Engineering
Faculty of Engineering and Science, University of Agder
Jon Lilletunsvei 9, 4879 Grimstad, Norway.
**Department of Technology & Innovation
Division of Process Automation, ABB
Norway

Abstract — **This paper presents adaptable methods for achieving fast collision detection using the GPU and Nvidia CUDA together with Octrees. Earlier related work have focused on serial methods, while this paper presents a parallel solution which shows that there is a great increase in time if the number of operations is large. Two different models of the environment and the industrial robot are presented, the first is Octrees at different resolutions, the second is a point cloud representation. The relative merits of the two different world model representations are shown. In particular, the experimental results show the potential of adapting the resolution of the robot and environment models to the task at hand.**

Keywords — Collision Detection, Industrial Robot, Hidden Markov Model, Expert System.

1 Introduction

Traditionally, the main industrial impact of robot technology has lied in domains containing repetitive routine tasks, in particular; manufacturing and automotive industries. In such environments, objects inside the workspace of the robot are most often either fixed at a known location or moving according to *a priori* determined and known patterns. Hence, the robot programs can be written such that these objects are avoided. Recently, there have been several initiatives to use industrial

robots in un-, or semi-structured outdoor environments. As an illustrative example, consider the case of robot automation in the Oil and Gas (O&G) industry, (Anisi et al., 2010, 2011). As a general trend within that industry, most of the easy accessible oil and gas fields have already been exploited, leaving the more remote and geopolitically challenging reserves for future exploration. Given the importance and focus of the oil and gas industry related to safety, environmental impact, cost efficiency and increased production, the potential for more extensive use of automation in general, and robot technology in particular, is evident.

Within the oil and gas industry, the use of robotics has so far been limited to special functions such as sub-sea and pipeline intervention and inspections using Remotely Operated Vehicles (ROVs), automation of drilling operations and well tractors. The degree of autonomy in these applications has however been very low, meaning that the robots are either remotely controlled, or that the robot follows *a priori* determined routes and hence has no ability to cope with environmental variations and uncertainty. The non-determinism mainly is due to mismatch between existing world model and the exact location of process components in the physical world. As presence of unexpected obstacles can not be ruled out in semi- structured environments typically found in the oil and gas industry, collision-free route cannot be guaranteed solely by offline programming. To avoid collisions in such settings, online, sensor-based collision detection and avoidance methods must be adopted. Collision detection considers the problem of indicating the presence of unmodelled obstacles while collision avoidance aims at finding an alternative path around discovered obstacles leading to the target. To this end, use of external sensors, *e.g.*, laser-, ultrasound- and vision sensors is necessary and the robot programs must be able to process these data and adjust accordingly online.

There are not many papers describing mapping of the environment in combination with collision detection in an industrial robotic setup. The work presented in (Henrich et al., 1998) shows the concept of modeling the robot and the environment, and finding the shortest distance to the obstacles. However, the approach in (Henrich et al., 1998) is based on simple models of both the robot and of the environment, and the models are created offline. In contrast, the work presented in this paper uses a more detailed offline generated model of the robot, which in real life applications

will not change during the time of the operation, but as apposed to (Henrich et al., 1998) it presents an online mapping of the environment.

Ramisa uses the Microsoft Kinect together with a robotic manipulator for grasping wrinkled cloth (Ramisa et al., 2012). As with this paper, the Kinect is used to gather depth data. As apposed to this paper, the Kinect is mounted on a structure external to the robot and not on one of the robot axes.

Dziegielewski's work on accurate mapping, (von Dziegielewski et al., 2012), shows that combining mapping and CUDA is not a new teqnique, even though the use of the CUDA cores are different in this paper. The mapping presented in this paper is different from (von Dziegielewski et al., 2012) in that the presented approach uses point clouds rather than a mesh based approach. The approach based on a point cloud is faster, but in some cases less informative.

This paper does not discuss different techniques for fast nearest neighbour search (NNS) such as NNS via k-d trees (Elseberg et al., 2012), because all the points are a part of the calculation, such methods will not improve the calculation time. The focus for this paper is to present how the calculation time in some cases could be decreased in collision detection applications by using the GPU, the result is based on our own algorithms developed for CUDA and CPU together with with open source CPU-based algorithms. CPU-based algorithms distance calculation which are performed on single core would perform better on multiple cores, this is not considered in this paper.

The remaining of this paper is organized as follows. Section 2 provides the problem formulation. Following that, the details of the system setup are given in Section 3. The experimental study conducted in this work is presented in Section 4 while Section 5 gives thorough discussions of the results. Finally, Section 6 provides concluding remarks and discussions on future work.

2 Problem Formulation

For industrial robots executing pre-planned tasks within a workcell, the concept of introducing new objects to the work cell without explicitly re-programming the robot controller is a research topic deserving further attention and development. The

common thing to do if a new object is added to the robot work cell is to stop the production. If the new object does not occlude the robot path while it manipulates any of the objects in the environment, the process could be started again without implications. On the other hand, if the new objects obstructs the path, a new path must be created by the operator. In the near future, it could be desirable to operate the robot in an industrial location with a reduced amount of protective fences. In such setups, it is important that the robot system is capable of detecting new objects and their locations. Hence, the robot must be part of a more adaptive system, with the possibility to dynamically introduce new objects to the scene. This raises the question about how the information about the objects should be made available to the robot controller. As described in previous work (Kaldestad et al., 2012b) and (Kaldestad et al., 2012a), one of these techniques could be manual input of a map. Depending on the application, the approach of manual input could have significant drawbacks. In addition, there are at least two drawbacks with respect to time. A manual human input of a map takes time and reduces performance. The second issue is the time aspect related to the resources of the human, which may be better utilised with other tasks. Because of these drawbacks, autonomous map generation using an external sensor is advocated. Next, the placement of the sensor is not trivial, and since the industrial robots to be used should potentially be capable to move on linear tracks along the floor or the ceiling, it is often desirable to mount the sensor on the robot.

To avoid excessive amounts of 3D sensor data, it is desirable to compress or reduce the data to increase performance. One approach that ensures that the data in the map does not exceed predefined space limits is the Octree approach.

Using robot manipulators in combination with Octrees has been done before (Faverjon, 1984), (Hayward, 1986) and there already exist methods for collision detection, based on overlapping Octrees. In most cases however, it is not desirable to wait until an actual collision before giving a warning, therefore using another approach such as representing the robot by an oversized Octree solves the problem with having the warning at time of impact, but it does not address the issue of generating a message when the robot approaches an object.

One of the advantages of using Octrees, is that the 3D position information is

equally distributed in the Cartesian space. This approach has of course both its advantages and drawbacks, and for that reason we have also compared it to using point clouds directly. One of the main advantages of using an Octree representation of both the robot and the environment, is especially when a large number of points are present within a small volume, and often these points may not give additional valuable information to act on. In such settings, all of these points which belong to their respective place in the Octree, will be reduced to one cube.

The trend is to explore more and more complex methods, but it is a necessity to simplify in order to stay within reasonable time limits. Still, a few years back, in late 2005 the computer frequency of the Intel and, a bit later, the AMD CPUs' stalled (Ross, 2008). Due to the heat dissipated and the power consumed by increasing the clock, it was found more reasonable to increase the number of processing areas on one CPU. This was followed by Nvidia's hardware and software architecture Compute Unified Device Architecture (CUD, 2006) (CUDA) in 2006. Because the programmer would benefit by Moore's law, a doubling of the transistor count every two years, which resulted in a doubling of the CPU frequency every second year up to 2005, today, other technologies could be used to continue improving application execution time. The CUDA architecture is one excellent technology to use when there are large parallel computational problems, such as calculating distances between points.

In this paper two main questions are addressed. First, how well will an Octree structure behave in terms of computational efficiency, compared to a pure point cloud. Second, what will be the fastest way to detect a collision or near collision, if all the known location on the robot and in the environment are compared, given the previously mentioned representations, using the CPU or CUDA (GPU).

3 System Setup

The setup depicted in Fig. C.3 consists of an ABB IRB1600-1.45 robot, an IRC5 industrial controller and a Microsoft Kinect sensor mounted on the robot's 4^{th} link, see Fig. C.1. The larger environment Fig. C.2, contains a second robot, an object on a rotary axis and protective walls inside the reachable workspace of the IRB1600

robot. Except the IRB1600 and the Kinect, all the other objects are static and part of the environment. Communication with the robot is done in a network connection

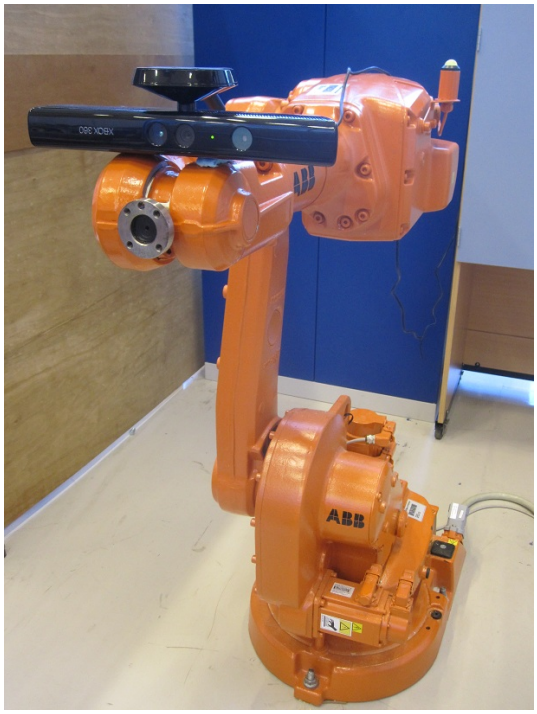


Figure C.1: *ABB IRB1600-1.45 robot with Microsoft Kinect sensor mounted on the 4th link.*

with a separate computer (PC_{rob}). A second computer in the network (PC_{obs}) with Nvidia 280 GTX CUDA-Enabled graphics card, was used for managing 3D-point data, which includes different models of the robot and the environment. The environment map is updated by gathering the Kinect distance data and storing it to the map. Further, robot motor angles are received from PC_{rob}, and used to transform the environment Kinect data in addition to transforming a point model of the robot. Before the Kinect-data is inserted to an Octree or point cloud, it is filtered using a distance filter, and it is also run through a statistical outliers removal to remove noisy depth readings. The transformed environment map is then used to calculate the distance from the robot to the environment.

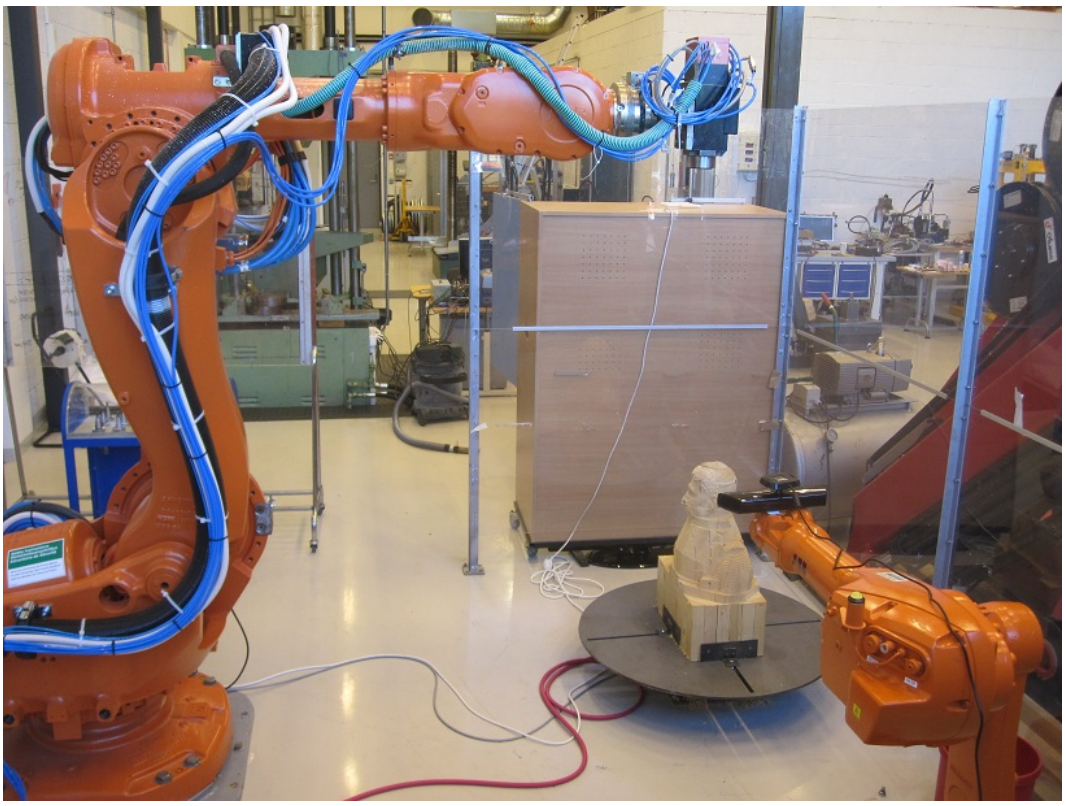


Figure C.2: *ABB IRB1600-1.45 robot and surrounding environment.*

4 Experiments

Given one or several concatenated point clouds of the environment as shown for example in Fig. C.4, an Octree representation can be generated. In this paper the Octree class in the Point PCL library (Rusu and Cousins, 2011) (PCL) is used. Six different Octree implementations have been compared with an approach working directly on a point cloud. The approaches were compared with respect to computational time and the number of Octree elements. Different sidelengths of the Octree cubes were used (1mm, 10mm and 100mm). In addition, the representation of the IRB1600 robot and the environment could be with different cube sidelengths. Fig. C.5 shows a representation where 100mm cube sidelengths were used for the IRB1600 robot and 1mm sidelengths were used to represent the environment. Notice that only the centre points of the cubes are shown in the figure. The representation of the robot was generated from a CAD file, while the representation of

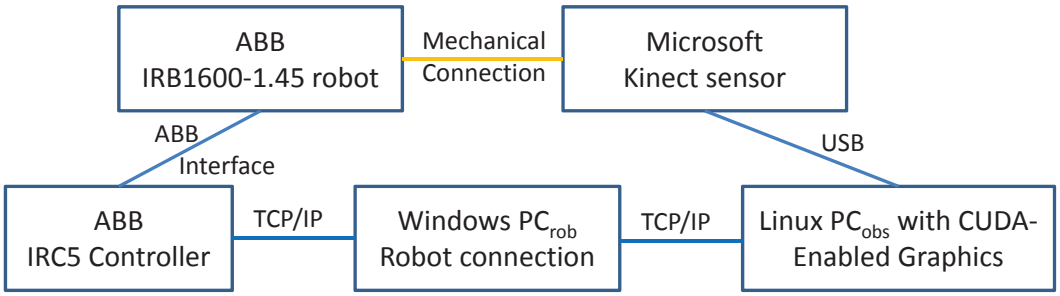


Figure C.3: *Overview of system components.*

the environment was generated using the Kinect sensor. Fig. C.6 shows another representation where 10mm side lengths were used for both the robot and the environment, here the cubes are shown.

The different approaches were compared based on computational time, which was found when calculating the smallest distance L_{min} between all the robot points R_i and all the environment points E_j, ie.

$$L_{min} = \min_{i,j} \|R_i - E_j\| \tag{C.1}$$

When using a point cloud, all the robot points generated from the CAD file were compared with all the points generated by the Kinect sensor. When using an Octree, the centre points in all the robot cubes were compared with the centre points in all the environment cubes.

The minimum collision distance for an Octree representation of both the environment and the manipulator is given by

$$L_{min} < \|\mathbf{c}_s\| \tag{C.2}$$

while for an uncompressed and un-approximated point cloud it is given by

$$L_{min} = 0 \tag{C.3}$$

where $\mathbf{c}_s$ is the vector pointing from the cube centre to one of its corners.

Table C.1 shows the experimental results for a relatively small point cloud. Eq. (C.1)

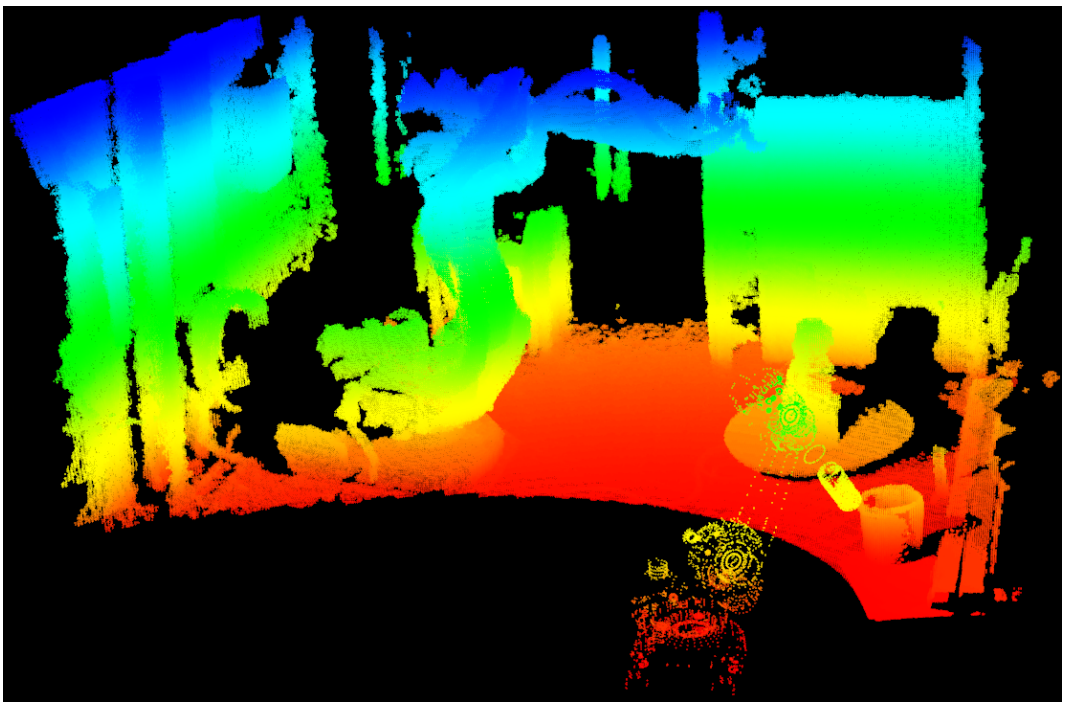


Figure C.4: *Example of concatenated point cloud of environment generated with Kinect sensor.*

was parallelised and run on a graphics card (GPU) with 240 cores using the CUDA library. Specifically the CUDA time includes the transfer of data to the GPU memory (two float arrays), the execution of two kernels; Where the first calculates the distance between tow coordinates in $\mathbb{R}^3$ and the second kernel finds the minimum value of the previous distances. Finally the minimum value is returned to the CPU. In addition, the computational time of Eq. (C.1) was tested on the CPU (Intel Core-i5 3.3 GHz) without parallelisation. As expected, the computational time depends on the number of point-to-point distance calculations. When using the largest Octree representation (100mm side lengths for both robot and environment) the required calculation time is 0.070 and 0.008 seconds respectively for the GPU and the CPU. It is interesting to notice that for this coarse representation the CPU calculation is actually faster than the GPU calculations, most likely caused by the initial setup time required for the GPU calculations. For all the other representations (smaller Octree cubes and directly on point cloud), the CUDA-based GPU calculations are faster than the CPU calculations. For example for the Octree

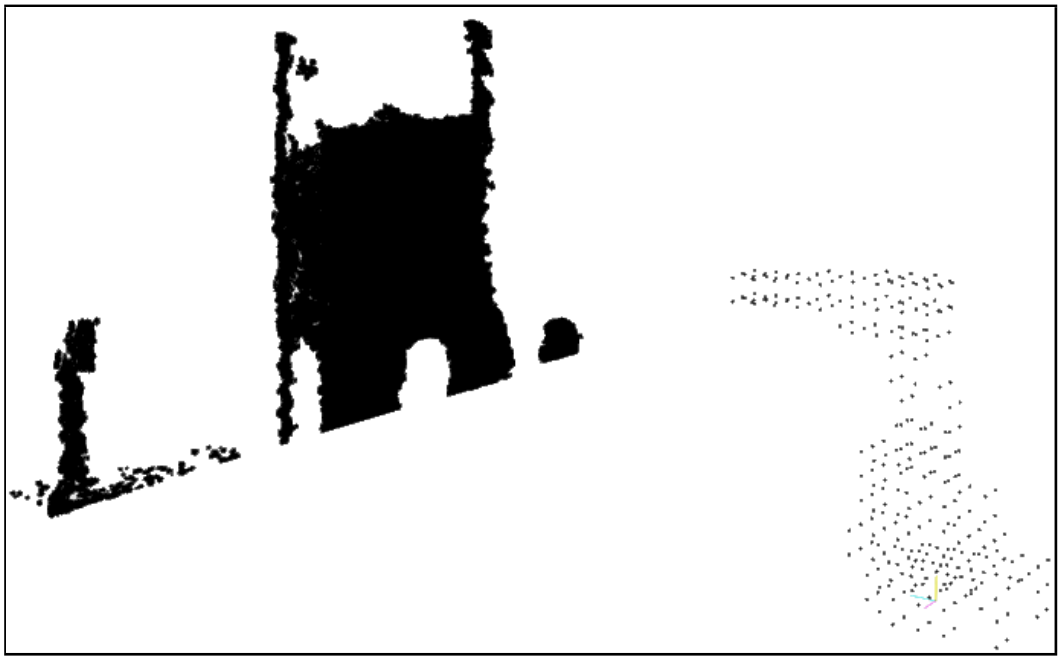


Figure C.5: *Illustration of centre points in Octree. 100mm sidelengths were used for the robot and 1mm sidelengths for the environment.*

10+1mm representation in Table C.1 the GPU calculation takes 4.647 seconds compared to the CPU calculation of 18.276 seconds which makes the GPU calculation approximately 4 times faster. Table C.2 shows the same result for a point cloud approximately 5 times larger. The same speed comparison for the Octree 10+1mm representation in this case shows that the GPU calculation is more than 20 times faster than the CPU calculation. Fig. C.7 illustrate the computational time vs. the number of point-to-point calculations for both the GPU and the CPU. The number of point-to-point calculations is given by the multiplication of robot and environment points. The most time consuming operation was when applying eq. (C.1) to the point cloud. For the smallest point cloud set, Table C.1 which requires 3.7 billion operations, the GPU was approximately 20 times faster than the CPU. The difference in speed increased even more for the largest point cloud in Table C.2, consisting of 18 billion operations, in this case the CUDA approach was 32 times faster than the CPU. Fig. C.7 summarises the number of elements with respect to time. The table shows that for a small number of operations, the CPU calculation

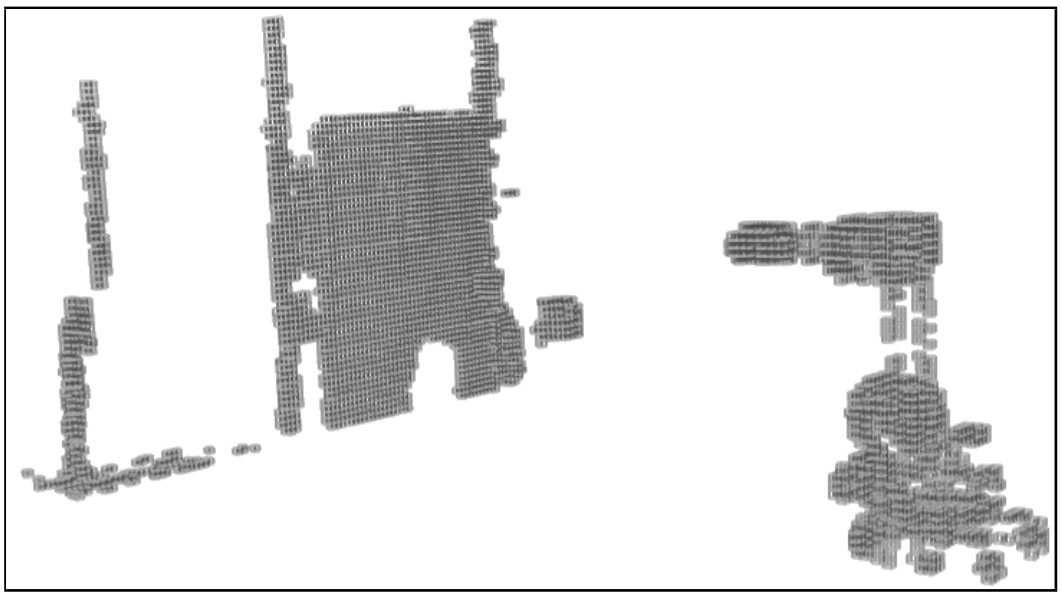


Figure C.6: *Illustration of both centre points and cubes in Octree with 10mm side-lengths for both the robot and the environment.*

time is lower than the CUDA time, but somewhere between 250,000 to 1,700,000 operations, the GPU is faster than the CPU.

5 Discussion

From the results in Table C.1 and C.2 it can be seen that for a smaller number of operations with 303 robot elements and 800 environment elements totaling $303 \cdot 800 = 242400$ operations, the CPU is 8.75 times faster than the GPU. Here, one operation is defined as one distance calculation. On the other hand, for a bit finer resolution of the Octree, 100mm for the robot and 10mm for the environment, there are 10,073,235 operations, but in this case the graphics card is 2.37 times faster than the CPU. The results presented in this paper, shows that for a smaller number of operations, it would be faster to let the CPU handle the calculations. The tipping point comes at around 10 million operations, when it is faster to let the GPU handle the calculations.

The graphics card used in this research, the Nvidia GTX 280 with 240 cores, was a card introduced in 2008. In comparison, state of the art today, a Nvidia GTX 690

Method	CUDA Time	CPU Time	Robot Elements	Environment Elements
Point cloud	9.694	197.455	19 221	192 953
Octree 1+1mm	3.369	42.567	10 809	107 213
Octree 10+1mm	4.647	18.276	4 397	111 395
Octree 100+1mm	0.443	1.254	303	112 639
Octree 10+10mm	2.049	5.494	4 397	33 574
Octree 100+10mm	0.156	0.369	303	33 245
Octree 100+100mm	0.070	0.008	303	800

Table C.1: *Experimental results with a relatively small point cloud of environment. Octree X+Ymm means cubes with X mm sidelength for the robot and Y mm sidelength for the environment.*

Method	CUDA Time	CPU Time	Robot Elements	Environment Elements
Point cloud	25.268	833.228	19 221	958 021
Octree 1+1mm	5.29	124.213	10 809	308 069
Octree 10+1mm	2.450	52.897	4 397	322 638
Octree 100+1mm	0.754	4.101	303	370 194
Octree 10+10mm	1.459	13.639	4 397	85 277
Octree 100+10mm	0.246	1.053	303	94 670
Octree 100+100mm	0.068	0.008	303	814

Table C.2: *Experimental results with a relatively large point cloud of environment.*

introduced in 2012 has core count 12.8 times larger, with 3072 cores total. Based on the number of cores, not taken the difference in clock speed into consideration, the Nvidia GTX 690 should perform several factors better than the GTX 280. But this speed-up would not just depend on the number of cores directly. As discussed in (Gregg and Hazelwood, 2011), for cases where huge amounts of data have to be copied to the graphics cards memory, which in this case is the number of elements in the point cloud / Octree, the total time could be increased drastically.

When comparing the pros and cons of the point cloud and the Octree as two different means to present the environment, it was found that that the Octree is ideal for setups where the exact coordinate location is not needed. If the exact measurement information is needed the uncompressed and un-approximate point cloud is the best

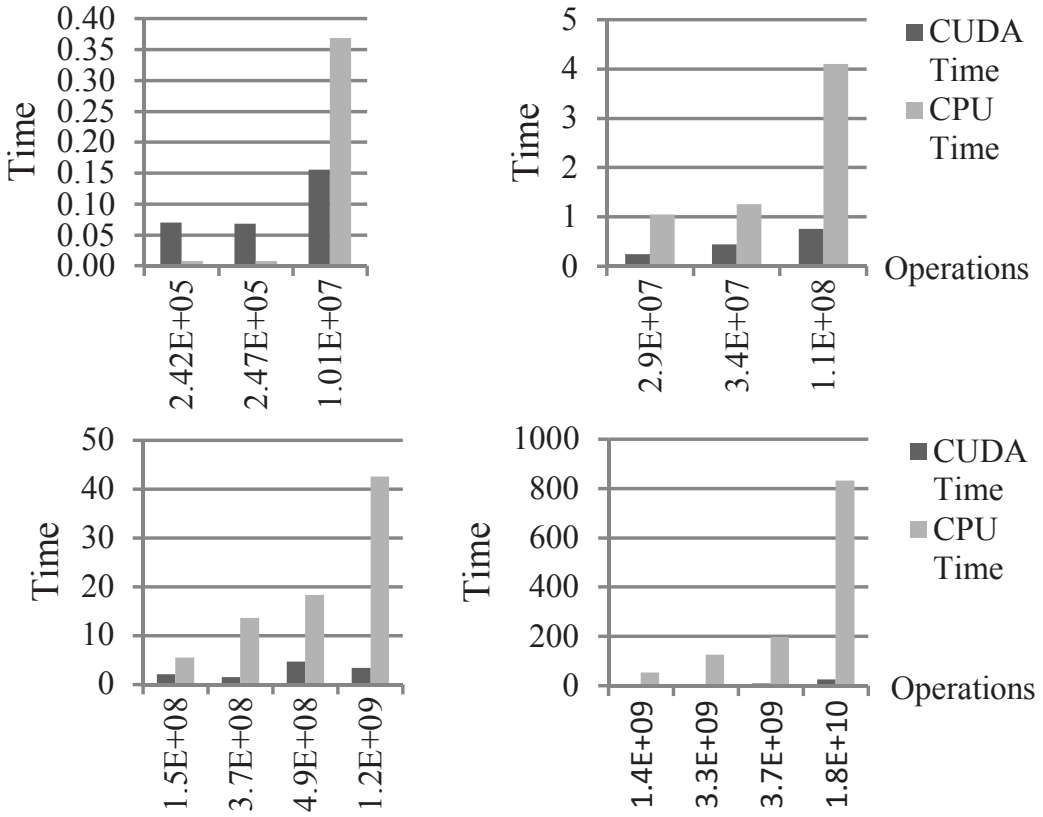


Figure C.7: *Computational time vs. number of point-to-point operations.*

approach. It is notable that by increasing the resolution of the Octree to the floating point representation, one may match the accuracy of the point cloud representation. However, this special case may be giving the system much more total overhead than using the point cloud directly. Another positive thing with the Octree is the inherent compression of the sensor data, which is distributed by steps of one cube side. Varying the cube size, gives a density and hence different compression, which in turn reduces the number of elements and therefore also the calculation time.

By expanding the collision detection algorithms, it is possible to tell on which link, and at which location there is soon to be a collision. This information could be very valuable to an adaptive system, because the robot representation could be changed on the fly to change the resolution of the robot model and there are two main reasons to do so. First, by having a lower resolution of the robot when it is far away from other objects will decrease the calculation time, which in some cases have the

desirable outcome that the robot could follow a path at higher velocity. Second, if the robot is approaching to do a part manipulation, the resolution of e.g. the wrist could be increased such that no collision is reported by passing objects on a very short range, an example of this is shown in C.8. To be able to choose when the links should change resolution will require a modification of (C.3)

$$L_{min} - RC_{dist} < \|\mathbf{c}_s\|$$
$$RC_{dist} < L_{min} + \|\mathbf{c}_s\|$$

where RC_{dist} is the user defined resolution change parameter such that PC_{obs} increases the resolution of a particular joint or a collection of joints if the corresponding part is greater. For a new resolution a new value for RC_{dist} has to be chosen. Of course the values of RC_{dist} would be chosen before robot task program execution, to make the system more independent from human input at execution time.
In this paper, all the points in the scene were checked, but another approach could be to check only the points which the robot could reach in the next step (plus a safety margin). This could be done by doing a quick conservative calculation of where each robot joint could be positioned at the next time step and make the calculations only in the surrounding volume.
A challenge when the robot should adapt to the environment is to differentiate between the physical changes that are expected and should happen, and the changes that not are supposed to happen. One example of such expected changes could be when the robot manipulates one or more objects in the environment. As this is a part of the normal operation, it should not raise a collision detection, which again means that the robot should be able to interact, or put in another words, "collide" with the object. To address this, one may associate an attribute with each object. As a simplistic example, let these attributes have the following values: "green", "orange" and "red". The elements of the static part of the scene, i.e., the areas that should not change during the time of operation and the robot is not allowed to interact with it, are marked "orange". Objects that the robot should interact with, should be marked with "green". Finally, objects introduced which are new to the scene should be marked as "red". Further, it should also be associated a distance to each of the three attributes. The new unknown objects in the room, marked with "red",

are expected to have higher uncertainty associated with them, compared the original environment, and hence, the associated collision detection distance should therefore be larger than that for the static environment. The static environment would then warn about collision at closer distance than for the new objects. Finally, the objects to be manipulated would have zero distance or no distance, which means that interaction with these objects should not result in a collision detection.

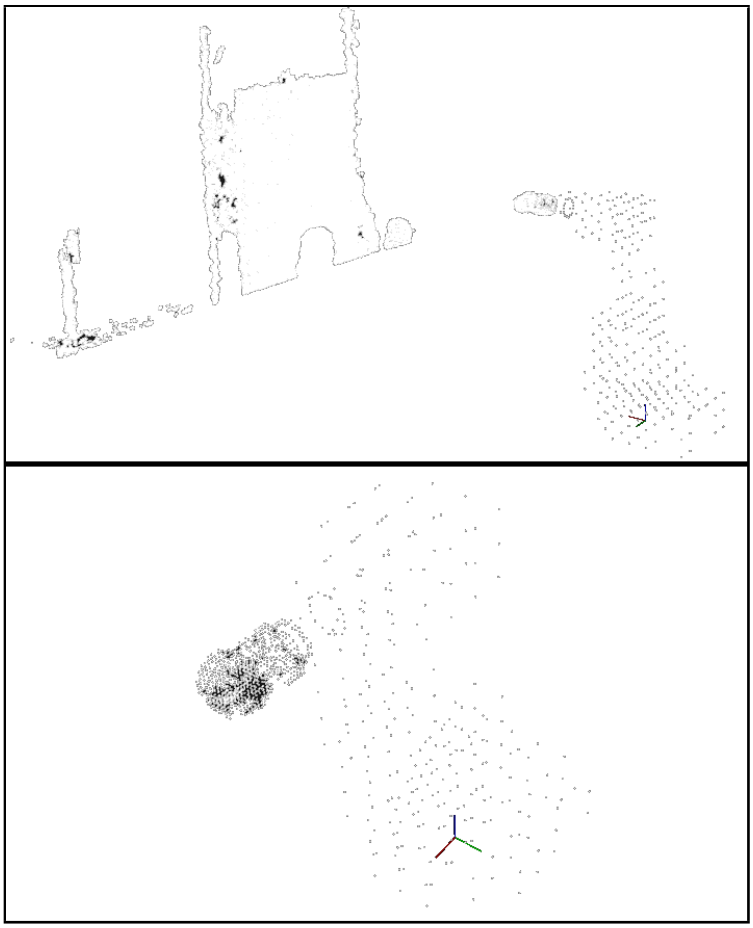


Figure C.8: *Example of different Octree resolutions on the robot.*

6 Conclusions And Future Work

This paper shows that the traditional approach with intersecting Octree cubes could be challenged by the emerging massively parallel GPU technology, bringing ad-

ditional information such as near collision. CUDA could reduce the calculation time significantly for a large number of operations, on the other hand, for a smaller number of calculations, the CPU is faster. In addition, this paper shows that it is beneficial to use an Octree representation of the environment if the computational time should be kept low. In addition, it is proposed to be using different resolution for the robot and the environment models, which will yield higher performance for an adaptable system. It has been shown that the developed CUDA algorithms in many cases outperforms the proposed CPU implementation and that GPU-based algorithms could be a favorable choice in real-time industrial robot collision detection applications.

Future work will focus on using and expanding the methods which have been developed in this paper. The next steps will focus on collision avoidance, which requires an accurate mapped environment and fast calculations to the nearest objects. In addition to this, developing methods for including newly discovered objects that have entered into the environment during operation, to the world map needs further attention.

Acknowledgment

Knut Berg Kaldestad acknowledges funding from ABB and the Norwegian Research Council through project number 193411/S60.

REFERENCES

(2006). *NVIDIA CUDA C - Programming Guide*.

Anisi, D., Gunnar, J., Lillehagen, T., and Skourup, C. (2010). Robot Automation in Oil and Gas Facilities: Indoor and Onsite Demonstrations. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 4729–4734, Taipei, Taiwan.

Anisi, D., Persson, E., Heyer, C., and Skourup, C. (2011). Real-World Demonstration of Sensor-Based Robotic Automation in Oil & Gas Facilities. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, San Francisco, California.

Elseberg, J., Magnenat, S., Siegwart, R., and Nüchter, A. (2012). Comparison of nearest-neighbor-search strategies and implementations for efficient shape registration. *Journal of Software Engineering for Robotics (JOSER)*, 3(1):2–12.

Faverjon, B. (1984). Obstacle avoidance using an octree in the configuration space of a manipulator. In *Robotics and Automation. Proc. IEEE Intl. Conf. on*, volume 1, pages 504 – 512.

Gregg, C. and Hazelwood, K. (2011). Where is the data? why you cannot debate cpu vs. gpu performance without the answer. In *Performance Analysis of Systems and Software (ISPASS), IEEE Intl. Symposium on*, pages 134 –144.

Hayward, V. (1986). Fast collision detection scheme by recursive decomposition of a manipulator workspace. In *Robotics and Automation. Proc. 1986 IEEE Intl. Conf. on*, volume 3, pages 1044 – 1049.

Henrich, D., Wurll, C., and Wörn, H. (1998). 6 dof path planning in dynamic environments-a parallel online approach. In *Robotics and Automation, Proc. IEEE Intl. Conf.*, volume 1, pages 330 –335 vol.1.

Kaldestad, K., Hovland, G., and Anisi, D. (2012a). CAD-Based Training of an Expert System and a Hidden Markov Model for Obstacle Detection in an Industrial Robot Environment. In *IFAC Intl. Conf. on Automatic Control in Offshore Oil and Gas Production*, Trondheim, Norway.

Kaldestad, K., Hovland, G., and Anisi, D. (2012b). Obstacle Detection in an Unstructured Industrial Robotic System: Comparison of Hidden Markov Model and Expert System. In *10th IFAC Intl. Symposiums on Robot Control*, Dubrovnik, Croatia.

Ramisa, A., Alenya, G., Moreno-Noguer, F., and Torras, C. (2012). Using depth and appearance features for informed robot grasping of highly wrinkled clothes. In *Robotics and Automation (ICRA), IEEE Intl. Conf. on*, pages 1703 –1708.

Ross, P. (2008). Why cpu frequency stalled. *Spectrum, IEEE*, 45(4):72.

Rusu, R. B. and Cousins, S. (2011). 3D is here: Point Cloud Library (PCL). In *IEEE Intel. Conf. on Robotics and Automation (ICRA)*, Shanghai, China.

von Dziegielewski, A., Hemmer, M., and Schomer, E. (2012). High quality conservative surface mesh generation for swept volumes. In *Robotics and Automation (ICRA), IEEE Intl. Conf.*, pages 764 –769.

Paper D

Collision Avoidance with Potential Fields Based on Parallel Processing of 3D-Point Cloud Data on the GPU

Knut B. Kaldestad, Sami Haddadin, Rico Belder, Geir Hovland and David A. Anisi

This paper has been published as:

Kaldestad, K., Haddadin, S., Belder, R., Hovland, G., and Anisi, D. Collision Avoidance with Potential Fields Based on Parallel Processing of 3D-Point Cloud Data on the GPU. *In IEEE International Conference on Robotics and Automation* Hong Kong, China, 2014.

Collision Avoidance with Potential Fields Based on Parallel Processing of 3D-Point Cloud Data on the GPU

Knut B. Kaldestad*, Geir Hovland* and **David A. Anisi
*Department of Engineering
Faculty of Engineering and Science, University of Agder
Jon Lilletunsvei 9, 4879 Grimstad, Norway.
**Department of Technology & Innovation
Division of Process Automation, ABB
Norway

Abstract — **In this paper we present an experimental study on real-time collision avoidance with potential fields that are based on 3D point cloud data and processed on the Graphics Processing Unit (GPU). The virtual forces from the potential fields serve two purposes. First, they are used for changing the reference trajectory. Second they are projected to and applied on torque control level for generating according nullspace behavior together with a Cartesian impedance main control loop. The GPU algorithm creates a map representation that is quickly accessible. In addition, outliers and the robot structure are efficiently removed from the data, and the resolution of the representation can be easily adjusted. Based on the 3D robot representation and the remaining 3D environment data, the virtual forces that are fed to the trajectory planning and torque controller are calculated. The algorithm is experimentally verified with a 7-Degree of Freedom (DoF) torque controlled KUKA/DLR Lightweight Robot for static and dynamic environmental conditions. To the authors knowledge, this is the first time that collision avoidance is demonstrated in real-time on a real robot using parallel GPU processing.**

1 Introduction & State Of The Art

1.1 Problem Statement

Over the last decades robots carried out various autonomous operations in the real world and were certainly a game changer in automation as we know it today. They perform repetitive and laborious work that requires high precision and large payloads. However, the success of manipulating robots in the real-world was so far limited to pre-planned tasks that require no online re-planning on trajectory nor task level. In particular, no technology was mature enough yet to let robots do quick and safe low-level decision making even on collision avoidance level. The recent trend of enabling robots to physically interact with humans in co-worker settings, however, enforces the need for viable solutions to the problem. Also the need to let larger industrial robots carry out more flexible tasks in other domains than manufacturing in rather uncertain and potentially changing environments increases the need even further. Both from a safety as well as task completion perspective, sensor based dynamic motion planning, for both in- and outdoor robots, could avoid causing damage in an unplanned event where an obstacle comes to block the original robot path.

1.2 State of the art

The state of the art is divided into three parts: applications, sensing and collision avoidance. The focus is on two significant application fields, namely robotic co-workers with light-weight robots, where the main concerns are high-performance physical human-robot interaction (pHRI) and safety, as well as new applications for larger industrial robots in harsh environments.

1.2.1 Applications

The KUKA/DLR Lightweight Robot (see Figure D.1) was initially developed by DLR (Albu-Schäffer et al., 2007) and has its roots in the space mission project ROTEX (Hirzinger et al., May). Later developments of the robot, after version

Figure D.1: *The KUKA/DLR Lightweight Robot equipped with a chuck-tool, which enables the robot to conduct different operations equipping e.g. an umbraco bit or a bore bit.*

III onwards, have been done in cooperation with the robot manufacturer KUKA (Bischoff et al., June). Major design considerations regarding the 7-DoF robot was the intention to let the robot support and intuitively interact with humans in industrial manufacturing domains. There has been considerable studies with the LWR in the pHRI context (e.g. (Haddadin et al., 2010; De Luca and Flacco, June)). In the second work e.g. the human hand is allowed to interact with the robot. A Kinect depth sensor is used to observe the scene, a hand gesture initializes interaction, and the hand is filtered out. The robot actively avoids all parts of the scene, which are still present in sensor data.

Other work presented in the field of collision avoidance is the reactive real-time motion generator by (Haddadin et al., 2010). More sophisticated path planning algorithms need considerable time for path calculation. Reactive motion generators are typically subject to getting stuck in local minima. The reactive algorithm in (Haddadin et al., 2010) runs at the inner control loop at 1kHz, it can generate smooth motions, while maintaining desired velocity profiles and ensure smooth human contact through velocity profiling. The algorithm demonstrated its performance by experiments for both statical and dynamical obstacles. The robot was able to circumvent obstacles and reach the respective goal. Also when an external force was applied to the robot, such that it deviated from its original path, it converged to

the goal when the external contact force was removed. In addition, the algorithm was validated by using different types of sensors, such as laser scanner and tracking system for keeping track of the wrist.

The circular fields methods use an analogy to electromagnetic fields and was recently extended in (Haddadin et al., 2011). In contrast to potential fields, circular fields associate magnetic fields to environmental obstacles and take into account the robot velocity as well. However, at the same time circular fields methods require some additional knowledge of the environment, such as surface normals. Important to notice is that the algorithm is not prone to local minima, which is demonstrated in (Haddadin et al., 2011) by simulating different well-known trap scenarios. Furthermore, the paper demonstrates dynamical obstacle avoidance with complex 3D geometry.

Figure D.2: *A robot located in a potentially explosive environment, among pipes carrying gas. This shows a state of the art petrochemical related application for industrial robots.*

Industrial robots are exposed to more demanding environments than ever before (see Figure D.2), performing on-site process inspection and manipulation while being remotely operated. Within the oil and gas industry, the strict regulations these robots have to comply with, such as ATEX (Leroux, 2007) (French for “atmospheres

explosibles") certified equipment and high reliability pose a real challenge when researching real-time collision avoidance. The work by (Anisi et al., 2011) presents a robotic valve manipulation application and showcases the according demands. The robot is located outdoor at a running hydrocarbon process facility doing valve manipulation with sensor based online trajectory planning. Two other applications are presented in (Anisi et al., 2010), the first is an indoor vision based valve manipulation with two collaborating robots. The first robot determines the orientation and the exact position of the valve, using an end-effector mounted network camera and a gradient based optimization algorithm. The second robot picks up the tool from a tool change holder and moves over to the valve and conducts the manipulation. The second presented application is a semi-automated scraper handling task, where a pipe is cleaned by sending the pressure driven scraper, from one location to the receiving destination. At the receiving end the robot opens the door to a depressurized chamber and locates the scraper by a proximity switch mounted at the tool. The scraper is extracted and the door is closed.
The literature referenced above demonstrates a few applications in an industry with strict regulations, where safety for humans and equipment is of highest importance. This shows that the oil and gas industry is starting to look at the opportunities that off-the-shelf industrial robots can provide. One of the challenges in this field is to best utilize already certified equipment and introduce new solutions to improve operations and safety.

1.2.2 Kinect, 3D Points, Graphics Card

Since the depth camera Kinect was released in November 2010, a vast amount of research has been done in relation to the device, such as 3rd party drivers from OpenKinect and OpenNI, Microsoft's own SDK for Windows and libraries for image and point cloud processing such as Point Cloud Library (PCL) (Rusu and Cousins, 2011). PCL version 1.6 has focus on CPU based algorithms, while algorithms for the GPU is under development. The work in (Neumann et al., 2011) uses Kinect data and processes it on the GPU, for point registration purposes. One of the described advantages of the GPU utilization, is the transformation of the Kinect

data to 3D points. Each pixel in the 640x480 RGB data is associated with a depth value. The transformation of each pixel to its corresponding 3D coordinate is highly suitable for parallel processing.

While there is significant research being done on the GPU, it has still not been a real alternative to the CPU in the majority of robotics applications. The reason is that the calculations need to be parallel and of a certain size before the GPU will outperform the CPU. Another factor has been development time and increased complexity, when comparing the complexity of C or C++ code for a CPU with CUDA C code for a GPU.

1.2.3 Collision avoidance

Robot collision avoidance could be described as a set of instructions sent to the robot such that it avoids unintended interaction with objects while moving to its desired position. Such instructions could be generated from models of the environment (Henrich et al., 1998) or sensor data (Borenstein et al., 1991) (or a combination of both). The pioneering work in real-time robot manipulator collision avoidance and potential fields started with (Khatib, 1984) and (Khatib, 1986). The principle of the method is that a distance dependent repulsive force is generated as a function of the distance that is either fed as control input or modifies the path of a stable desirably attractive dynamical system, this in turn, generates a suitable reference trajectory. For obstacle avoidance the force will be repulsive, and is e.g. formulated as a polynomial function. This typical, very broad class of functions could give application desired characteristics such as the force increasing polynomially the closer the manipulator gets to the obstacle. Even though the field of obstacle avoidance is not new, it is still being researched heavily today. At the time of writing, a viable solution for collision avoidance in this field has not been demonstrated. Some of the more recent work on improving potential field like methods is the significant extension in (Haddadin et al., 2011) of the original circular fields approach developed in (Singh et al., 1996).

The work presented in this paper is related to (Flacco et al., 2012), however in this paper the robot is represented as a vertex model as opposed to spheres. As such, this

gives a more realistic representation of the robot. The volume map is represented as voxels with a user defined resolution. The calculations are performed using both the GPU and CPU, which is one of the main contributions in this paper, in contrast to previous work focusing only CPU implementation.

2 Algorithm

2.2 Overview of Approach

In our framework, collision avoidance behavior is incorporated on two levels:

1. on trajectory deformation level, i.e. $\boldsymbol{x}_d(t)$ responds to the virtual forces
2. on torque control level, i.e. a control input $\boldsymbol{\tau}_v$ causes causes avoidance joint torques

In order for the first level to respond to virtual forces, the trajectory generation needs to establish a dynamical system with physical motivation (essentially a virtual impedance behavior) such as the one presented in (Haddadin et al., 2010). The virtual dynamics generate a reference trajectory that responds to disturbance wrenches on operational task level and is then fed to a Cartesian impedance controller. Clearly, this scheme can only ensure end-effector collision retraction/avoidance. However, for kinematically redundant manipulators such as the LWR, this does not cover appropriate nullspace reactions. Therefore, a nullspace collision avoidance controller $\boldsymbol{\tau}_v$ is designed to implement according behavior. For this, the virtual forces $\mathscr{F}_{v,n}$ that act on each link n are projected via the respective sub-Jacobians J_n to joint space, followed by a suitable nullspace projector $\mathscr{N}(\boldsymbol{q})$. This ensures that $\boldsymbol{\tau}_v$ does not interfere with the primary impedance task. The overall controller can be written as

$$\boldsymbol{\tau}_d = J^T(K_x\tilde{\boldsymbol{x}} + D_x\dot{\tilde{\boldsymbol{x}}}) + \boldsymbol{g}(\boldsymbol{q}) + \boldsymbol{\tau}_v \tag{D.1}$$

$$= J^T(K_x\tilde{\boldsymbol{x}} + D_x\dot{\tilde{\boldsymbol{x}}}) + \boldsymbol{g}(\boldsymbol{q}) + \mathscr{N}(\boldsymbol{q})\sum_{n=1}^{N} J_n^T\mathscr{F}_{v,n}, \tag{D.2}$$

2.1 Notation Used

Table D.1: *For ease of use our notation is summarised below.*

Symbol	Explanation
$\boldsymbol{q}$	Robot joint angle vector
$\boldsymbol{\tau}_d$	Desired control input
$\boldsymbol{x}$	Real end-effector positon
$\boldsymbol{x}_d$	Desired end-effector position
$\tilde{\boldsymbol{x}}$	Difference between real and desired positions
D_x	Cartesian damping matrix
K_x	Cartesian stiffness matrix
$\boldsymbol{g}(\boldsymbol{q})$	Gravity compensation torque in joints
$J(\boldsymbol{q})$	Robot Jacobian
$\mathscr{N}(\boldsymbol{q})$	Nullspace projector
$\boldsymbol{\tau}_v$	Virtual torques
$\boldsymbol{f}_n, \boldsymbol{m}_n$	Link force/moment vector
$F(\boldsymbol{d})$	Reactive force
$\mathscr{F}_{v,n}$	Virtual forces/moments
$T_l(\boldsymbol{q})$	Transformation matrices, one for each link: robot base to link frame as a function of $\boldsymbol{q}$
$\boldsymbol{p}_{in}$	Depth data from Kinect sensor
$\boldsymbol{r}_{in}$	Robot vertices
$\boldsymbol{r}_j$	Transformed vertices in robot model
d	Distance between robot and env. points
d_c	Distance from an environment point to the center of rotation for a robot link
r_{max}	Threshold force distance
s_l	Voxel side length
v_s	Voxel size
x_0, y_0, z_0	Offsets applied to measured points
x_w, y_h, z_d	Edge lengths of camera volume

where the notation is consistent with Table D.1. The virtual disturbance wrenches $\mathscr{F}_{v,n}$ for each link n are typically generated from sensory data and/or geometric knowledge from the environment. Environment 3D data is gathered using a Microsoft Kinect. Applying Algorithm 1 with 3D environmental point cloud data and a 3D vertex model of the robot, the set of virtual forces/moments $\mathscr{F}_{v,n}$ is generated as explained next.

2.3 Calculation of reactive forces

The calculation of the reactive forces are well suited for parallel processing, because the distance between each point represented by the robot model and each point in the environment can be calculated separately. The distance $\boldsymbol{d}$ would then be used, such that the force is a function of the distance, $F(\boldsymbol{d})$. The force function could take many forms, where as in our case, we use a second order polynomial function. To avoid any forces from objects located at a predefined distance farther from the robot, a threshold force distance r_{max} is set such that the robot only reacts to the objects located at $\|d\| \leq r_{max}$. These forces are then summed to create a force vector and a moment vector for each link and the end effector.

$$\boldsymbol{f}_n = \sum F(\boldsymbol{d}) \tag{D.3}$$

$$\boldsymbol{m}_n = \sum F(\boldsymbol{d}) \times \boldsymbol{d}_c \tag{D.4}$$

where $\boldsymbol{f}_n$ and $\boldsymbol{m}_n$ are the forces and moments for each respective link $\boldsymbol{n}$, and $\boldsymbol{d}_c$ is the distance from the environment point to the center of rotation for the robot link.

2.4 Voxel map creation

When creating the voxel map, a small footprint and a parallel scheme of the data is important. To reduce the size of the map, a simple compression method is proposed, which is highly parallel and easily deployable on the GPU.

A regular voxel map for 3D Cartesian space representation could be represented by

Algorithm 1: Reactive Force/Moment Calculation

Input Map:
Depth data $\boldsymbol{p}_{in}$
Voxel size v_s
Bounds x_0, y_0, z_0, x_w, y_h, z_d
Neighbours n
Input Robot:
Vertices $\boldsymbol{r}_{in}$ for all n links
Joint angles $\boldsymbol{q}$
Transformation matrices $T_l(\boldsymbol{q})$
Input Potential field:
Pointer to force function $F(\boldsymbol{d})$
Threshold force distance r_{max}

Voxel insertion:
for *each point* $\boldsymbol{p}$ *in* $\boldsymbol{p}_{in}$ *within bounds* **do**
  if $\boldsymbol{p}$ *within VoxelMap* **then**
    VoxelMap $\leftarrow \boldsymbol{p}$
  end
end

Remove robot from VoxelMap:
for *each vertex* $\boldsymbol{r}$ *in* $\boldsymbol{r}_{in}$ **do**
  $\boldsymbol{r}_j \leftarrow T_l(\boldsymbol{q}) \times \boldsymbol{r}$
  if *VoxelMap contains* $\boldsymbol{r}$ *in* $\boldsymbol{r}_j$ *as voxel* **then**
    Remove $\boldsymbol{r}_j$ from VoxelMap
  end
end

Remove Outliers:
for *each voxel* $\boldsymbol{v}$ *in VoxelMap* **do**
  if $\boldsymbol{v}$ *has less than n neighbours* **then**
    Remove $\boldsymbol{v}$ from VoxelMap
  end
end

Calculate forces and moments:
for *each link n* **do**
  $\hat{\boldsymbol{f}} \leftarrow \boldsymbol{0}$, $\hat{\boldsymbol{m}} \leftarrow \boldsymbol{0}$
  for *each voxel center* $\boldsymbol{v}_c$ *in VoxelMap* **do**
    for *each* $\boldsymbol{r}$ *in* $\boldsymbol{r}_j$ *belonging to link n* **do**
      $\boldsymbol{d} \leftarrow \boldsymbol{r} - \boldsymbol{v}_c$
      if $\|\boldsymbol{d}\| < r_{max}$ **then**
        $\hat{\boldsymbol{f}} \leftarrow \hat{\boldsymbol{f}} + F(\boldsymbol{d})$
        $\hat{\boldsymbol{m}} \leftarrow \hat{\boldsymbol{m}} + F(\boldsymbol{d}) \times \boldsymbol{d}_c$
      end
    end
  end
  $\mathscr{F}_{v,n} \leftarrow \hat{\boldsymbol{f}}, \hat{\boldsymbol{m}}$
end

return $\mathscr{F}_{v,n}$

points of type $\boldsymbol{P} = (x\ y\ z)^T$, and demonstrate its presence in a 3D array by

$$Map_3(x,y,z) = 1 \tag{D.5}$$

If the map is limited e.g. to 5 m x 5 m x 5 m with a resolution of 5 mm, the map would consist of 10^9 elements, or 1 gigabyte of memory allocated in an uncompressed state. Such an approach is clearly not space efficient. Methods such as *kd-trees*(Bentley, 1975) could now be used to reduce the memory footprint. There, the 3D points are structured in a 3-dimensional tree which allows for faster search. The time to generate the tree and calculate the k nearest neighbors (kNN) is, however, not fast enough for our demands. The work in (Arefin et al., 2012) provides, in addition to their own algorithm GPU-FS-kNN, a good overview of different kNN algorithms for both CPU and GPU. Due to the current speed limitations of these approaches, we chose to follow a different algorithmic path:

If the 3D-Sensor is located in a fixed position and orientation, the only input to the map is the (x,y,z)-location of the according measurements. If the z-coordinate changes, the map needs to be updated (we can not get multiple depth values for the same pixel) due to the fact that originally the data is $2\frac{1}{2}$D. Because of this limitation, the voxel map can be represented as a 2D array, where its implicit structure may be written as

$$Map_{2.5}(x,y) = z. \tag{D.6}$$

This representation is in fact very similar to the original one, except for the fact that it has now a clear structure and it is possible to index the points directly. This is done by creating the map with seven parameters: Offsets x_0, y_0, z_0, dimensions x_w, y_h and z_d and voxel size v_s. This results in the following relationship between sensor points $P_k = (x_k\ y_k\ z_k)$ and $Map_{2.5}(x_i\ y_i)$.

$$\hat{z} = z_k \tag{D.7}$$

$$\hat{x}_i = \text{floor}\left(\frac{x_k - x_0}{v_s}\right) \tag{D.8}$$

$$\hat{y}_i = \text{floor}\left(\frac{y_k - y_0}{v_s}\right) \tag{D.9}$$

From eqs. (D.8) and (D.9) it should be fairly straightforward to see that large values for v_S will lower the total resolution. The consequence is that not all points P_k will appear in the $Map_{2.5}$-representation. Averaging the depth value of the points could result in hallucinated distances in open space, e.g. the mean distance between an object that is close to the sensor, and an object farther away. If the location of the Kinect is chosen such that it is close to the volume that it shall observe, a reasonable choice is to insert the point with the lowest z-value, into the $Map_{2.5}$. If the depth camera were to provide a resolution of 1 mm, a worst case with a voxel side length of s_l would lead to a loss of $m = s_l^2 - 1$ points.
This loss is not of great importance as long as s_l is kept reasonably low compared to the size of the object surface area.
Some of the benefits of the $Map_{2.5}$ structure are:

- Omission of the z-dimension, for 3 m depth with resolution 0.01 m which results in a 300 times more compact structure than Map_3.

- Structured in a way, which enables fast localization and removal on the GPU. Example: The robot could be removed from the map without using any search algorithm. The robot is simply inserted into a separate voxel map, created with the same parameters as the environment. Then in parallel for all points, remove the points where the robot overlaps with the environment.

- No need to search for neighbors which can be indexed directly. Example: To find N neighbors for all points, located within radius r from point p, it is only necessary to check exactly those voxels within radius r of the point. This can be done for each point in parallel.

Well known problems with the depth data from the Kinect are noise and presence of outliers (Khoshelham and Elberink, 2012). Outliers could contribute to residual motion of the robot if it is located within the threshold range of the link. This could severely affect the potential field force acting on the robot, if it is located close to the link. An outlier could potentially increase the risk of collisions, jeopardizing

the entire purpose of collision avoidance. A simple and fast algorithm for outlier removal for the GPU is therefore proposed in algorithm 1.

Finally, the concrete robot (LWR) is represented by seven vertex models, one for each link, base and end-effector. The robot has two representations in the algorithm, the first is a voxel map ($voxel_{rob}$) used for robot removal. The second is an unstructured vertex model (see Figure D.3) used for calculating the forces. $voxel_{rob}$ is created with the same parameters as the environment map ($voxel_{env}$). For each iteration, a homogeneous transformation accordingly to the robot's (sub) forward kinematics are applied to each of the vertex models of the robot and inserted into a new map $voxel_{rob}$. The robot is then removed from the environment. Each voxel in $voxel_{rob}$ that corresponds to a voxel in $voxel_{env}$ (plus a tolerated offset) is thus removed from $voxel_{env}$.

The unstructured vertex model of the robot, which gives its proper 3D representation, is used for calculating the forces generated from the potential field. For each vertex on every link in the robot vertex model, a distance is calculated to all environment voxels. Each of these distances contributes to its link with three forces and three moments. The forces and moments are then added for every link. In the end, three total forces and moments act on each joint, respectively.

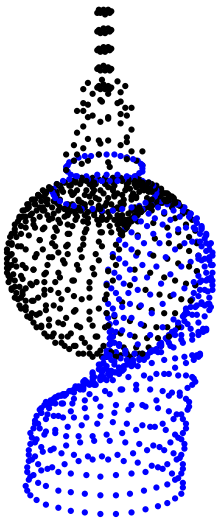


Figure D.3: *Vertices of the last arm segment, wrist, flange and tool.*

3 Experiments

3.1 System Setup

The experimental system consists of a KUKA LWR IV manipulator with 7-DoFs and the robot controller system *Beasty* (Parusel et al., May) running at 1000 Hz. The computer for the calculation of algorithm 1 is equipped with an Intel Xeon 3.3 GHz CPU, 8GB of memory and a NVIDIA Geforce GTX 680 graphics card.

3.2 Experiment Design

The experiments are carried out in an industrial setting, partly shown in Figures D.6 to D.9. The robot is located in the environment center and the Kinect depth sensor is placed such that it captures the robot in a side view (please see accompanying video for details).
For both experiments the voxels had a resolution of 10 mm, while the volume covered by the depth camera is x_{width}=2000 mm, y_{height}=2000 mm and z_{depth}=1800 mm. The robot model consists of two sets of 7 vertex models, which contain 2468 and 4701 vertices in total.

3.2.1 Experiment 1, Static Objects

In the static environment, the robot first runs the path simply generated by eq. (D.2) if no collision forces modify the path. The depth data is checked for changes at the same rate as the data becomes available. First, a box is placed in the environment to block the robot path. Different types of objects are then placed on top of the box, thus changing the environment. The robot actively avoids every object that partly blocks its free movement volume.

3.2.2 Experiment 2, Dynamic Objects

In this experiment the robot end-effector is set to reach a goal location. A human worker then enters the environment during the robot movement, causing the manipulator to deviate from its nominal path.

4 Results

The experimental evaluation indicates the performance of the algorithms for both experimental setups. The robot shows whole-arm collision avoidance, while still being able to reach its final goal position if the according volume is clear. The time-stamps in the screenshots (Figure D.6 – D.9) are all relative, the entire experiment was done in one shot.

4.1 Static Obstacles

Figure D.6 depicts the desired path if no obstacles obstruct the motion. The robot intends to move along this path for the entire experiment. While being in motion, a blue box is placed in the path of the robot, see Figure D.7. As can be seen from the figure, the robot actively avoided the box. To make it more difficult for the robot to reach its goal, the white obstacle is placed on top of the box, see Figure D.8. The white obstacle is moved to another position further away from the robot. The robot responds by avoiding the obstacle on the right side (not shown in the figures).

4.2 Dynamic Obstacles

The second phase of the experiment includes a dynamic obstacle, a person enters the workspace while the robot is in motion. As one can see, the robot is able to quickly avoid the human and prevent the collision. In the accompanying video, the dynamic response can be seen more clearly and it is shown that the robot converges to the goal again as soon as the human leaves the workspace.

4.3 Calculation Time

Since the map is recreated each time new sensor data becomes available, it is possible to calculate the potential forces in the mean time. Even though the environment update rate is restricted by the Kinect, the robot state can be retrieved at a rate of 1 kHz. In parallel to waiting for new depth data, the algorithm then calculates the current potential field forces based on the updated robot position. The potential

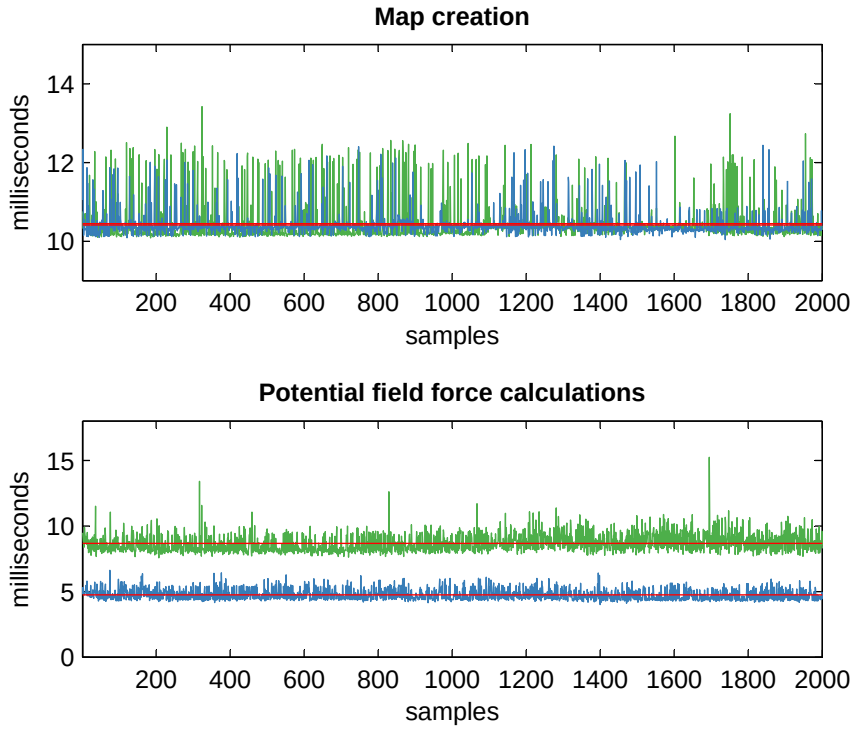


Figure D.4: *Algorithm performance, calculation time on the vertical axis in milliseconds for 2000 samples. The robot model consists of 2468 vertices. The green graph shows an environment voxel resolution of 5 mm, while the blue graph shows a voxel resolution of 10 mm. The red lines depicts the average measurement value.*

field force calculations continuously receive new robot joint angles, transforming the robot vertices accordingly, and calculating the respective potential field forces.

Figure D.4 depicts the calculation time for 2468 robot vertices with 10 mm and 5 mm resolution. The average time for the full map creation is 10.41 ms and 10.44 ms, respectively. The potential field calculations take on average 4.74 ms and 8.68 ms, respectively. Figure D.5 is generated with a robot consisting of 4701 vertices, and for two different voxel resolutions of 10 mm and 5 mm, respectively. The average time for the map creation was 10.45 ms and 10.51 ms, while the potential field calculation takes 5.70 ms and 11.29 ms, respectively.

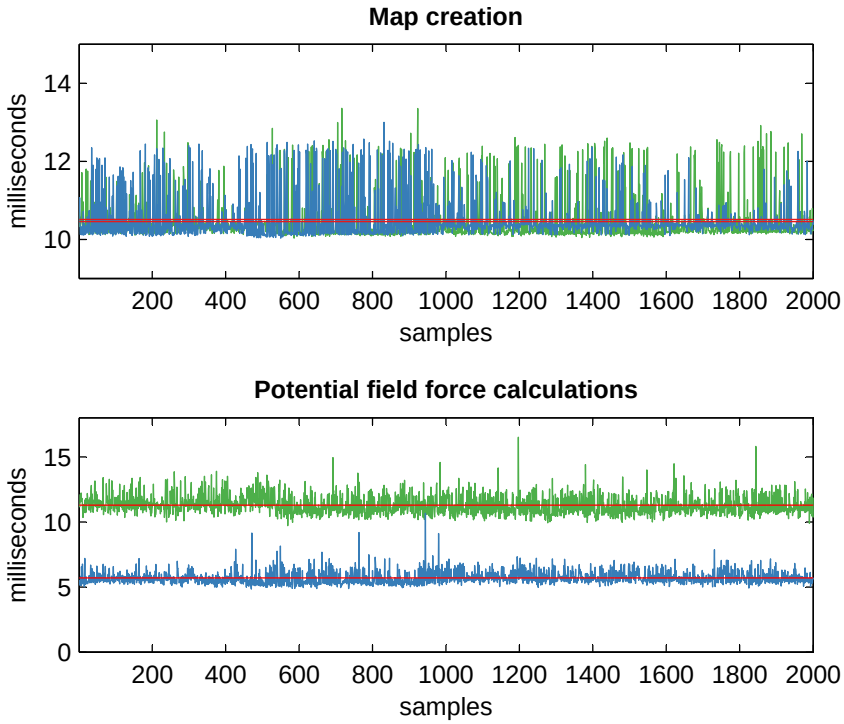


Figure D.5: *Algorithm performance, calculation time on the vertical axis in milliseconds for 2000 samples. The robot model consists of 4701 vertices. The green graph shows an environment voxel resolution of 5 mm, while the blue graph shows a voxel resolution of 10 mm. The red lines show the average measurement value.*

4.4 Comparison GPU and CPU speeds

The results in Table 4.4 show a significantly lower calculation time for the GPU algorithm. The performance increase varies from a factor of 68.63 to 402.07 depending on voxel map resolution and number of robot vertices. The significantly better performance on the GPU is the enabling factor allowing real-time collision avoidance.

5 Conclusion

In this paper, a study on GPU based collision avoidance with Potential Fields is presented, using 3D point cloud data converted to voxels as environmental representation. The virtual forces that are fed to the trajectory planning and torque

Architecture	Voxel map resolution	Robot vertices	Algorithm calculation time	Performance factor
CPU	10 mm	4701	1.426 s	129.63x
GPU	10 mm	4701	0.011 s	
CPU	5 mm	4701	5.692 s	402.07x
GPU	5 mm	4701	0.014 s	
CPU	10 mm	2468	0.755 s	68.63x
GPU	10 mm	2468	0.011 s	
CPU	5 mm	2468	3.009 s	273.54x
GPU	5 mm	2468	0.011 s	

Table D.2: *CPU vs GPU performance. In the performance calculation* x_{width} *and* y_{height} *are both 2000 mm,* z_{depth} *is 1800mm and* r_{max} *is 300 mm.*

control level are calculated in real-time. In fact, the proposed algorithm may even run significantly faster than the sensor frame rate. The experimental performance of the scheme showed good results with a 7-DoF KUKA/DLR Lightweight robot for various static and dynamic environmental conditions. In particular, the combination of trajectory deformation based on virtual dynamics that are affected by the virtual forces on Operational space level, together with the projection of the forces into the nullspace of the Cartesian impedance controller led to convincing whole body real-time collision avoidance responses. To the authors knowledge, this paper presents for the first time real-time collision avoidance using a real robot and parallel GPU processing.

Acknowledgment

This work was performed while the first author was visiting DLR, whose hosting is gratefully acknowledged. This work has been partially funded by the European Commission's Sixth Framework Programme as part of the project SAPHARI under grant no. 287513. The work is also funded by ABB and the Norwegian Research Council through project number 193411/S60.

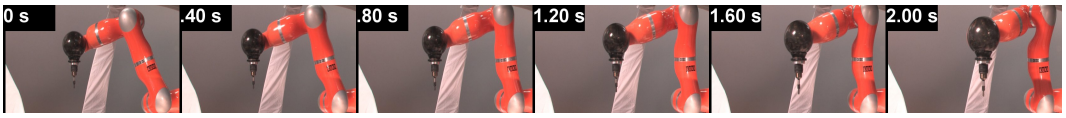


Figure D.6: *No obstacle: Showing the original robot path.*

Figure D.7: *Static obstacle: A box blocks the robot path.*

Figure D.8: *Static obstacle: Another object is placed in front of the robot, to make the free movement volume even smaller.*

Figure D.9: *Dynamic obstacle: A human worker enters the work area and the robot actively avoids him.*

REFERENCES

Albu-Schäffer, A., Haddadin, S., Ott, C., Stemmer, A., Wimböck, T., and Hirzinger, G. (2007). The DLR lightweight robot - lightweight design and soft robotics control concepts for robots in human environments. *Industrial Robot Journal*, 34(5):376–385.

Anisi, D., Gunnar, J., Lillehagen, T., and Skourup, C. (2010). Robot Automation in Oil and Gas Facilities: Indoor and Onsite Demonstrations. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 4729–4734, Taipei, Taiwan.

Anisi, D., Persson, E., Heyer, C., and Skourup, C. (2011). Real-World Demonstration of Sensor-Based Robotic Automation in Oil & Gas Facilities. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, San Francisco, California.

Arefin, A., Riveros, C., Berretta, R., and Moscato, P. (2012). GPU-FS-kNN: A Software Tool for Fast and Scalable kNN Computation Using GPUs. *PLoS One*, 7(8):e44000.

Bentley, J. L. (1975). Multidimensional Binary Search Trees Used for Associative Searching. *Commun. ACM*, 18(9):509–517.

Bischoff, R., Kurth, J., Schreiber, G., Koeppe, R., Albu-Schäffer, A., Beyer, A., Eiberger, O., Haddadin, S., Stemmer, A., Grunwald, G., and Hirzinger, G. (June). The KUKA-DLR Lightweight Robot arm - a new reference platform for robotics research and manufacturing. In *Robotics (ISR), 2010 41st Intl. Symp. on and 2010 6th German Conf. on Robotics (ROBOTIK)*, pages 1–8.

Borenstein, J., Koren, Y., and Member, S. (1991). The vector field histogram - fast obstacle avoidance for mobile robots. *IEEE Journal of Robotics and Automation*, 7:278–288.

De Luca, A. and Flacco, F. (June). Integrated control for pHRI: Collision avoidance, detection, reaction and collaboration. In *Biomedical Robotics and Biomechatronics (BioRob), 2012 4th IEEE RAS EMBS Intl. Conf.*, pages 288–295.

Flacco, F., Kroger, T., De Luca, A., and Khatib, O. (2012). A depth space approach to human-robot collision avoidance. In *Robotics and Automation (ICRA), 2012 IEEE Intl. Conf.*, pages 338 –345.

Haddadin, S., Belder, R., and Albu-Schäffer, A. (2011). Dynamic motion planning for robots in partially unknown environments. In *IFAC World Congress (IFAC2011)*, Milano, Italy.

Haddadin, S., Urbanek, H., Parusel, S., Burschka, D., Roßmann, J., Albu-Schäffer, A., and Hirzinger, G. (2010). Real-time reactive motion generation based on variable attractor dynamics and shaped velocities. In *IEEE Intl. Conf. on Intelligent Robots and Systems*.

Henrich, D., Wurll, C., and Wörn, H. (1998). 6 dof path planning in dynamic environments-a parallel online approach. In *Robotics and Automation, Proc. IEEE Intl. Conf.*, volume 1, pages 330 –335 vol.1.

Hirzinger, G., Brunner, B., Dietrich, J., and Heindl, J. (May). ROTEX-the first remotely controlled robot in space. In *Robotics and Automation, 1994. Proc. IEEE Intl. Conf.*, pages 2604–2611 vol.3.

Khatib, O. (1984). Real-time control of manipulators in operational space. In *Proc. of the 28th Annual Stanford Conf. American Society for Quality Control*, pages 9/1–9/7, Palo Alto, CA, USA.

Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *The Intl. Journal of Robotics Research*, 5(1):90–98.

Khoshelham, K. and Elberink, S. O. (2012). Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications. *Sensors*, 12(2):1437–1454.

Leroux, P. (2007). New regulations and rules for atex directives. *Industry Applications Magazine, IEEE*, 13(1):43–51.

Neumann, D., Lugauer, F., Bauer, S., Wasza, J., and Hornegger, J. (2011). Real-time RGB-D Mapping and 3-D Modeling on the GPU using the Random Ball Cover Data Structure. In Fossati, A., Gall, J., Grabner, H., Ren, X., and Konolige, K.,

editors, *IEEE Intl. Conf. on Computer Vision (ICCV) Workshops*, pages 1161–1167.

Parusel, S., Haddadin, S., and Albu-Schäffer, A. (May). Modular state-based behavior control for safe human-robot interaction: A lightweight control architecture for a lightweight robot. In *Robotics and Automation (ICRA), 2011 IEEE Intl. Conf. on*, pages 4298–4305.

Rusu, R. B. and Cousins, S. (2011). 3D is here: Point Cloud Library (PCL). In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Shanghai, China.

Singh, L., Stephanou, H., and Wen, J. (1996). Real-time robot motion control with circulatory fields. In *Robotics and Automation, 1996. Proc., 1996 IEEE Intl. Conf. on*, volume 3, pages 2737–2742 vol.3.

Paper E

Implementation of a Real-Time Collision Avoidance Method on a Standard Industrial Robot Controller

Knut B. Kaldestad, Geir Hovland and David A. Anisi

This paper has been submitted as:

Kaldestad, K., Hovland, G., and Anisi, D. Implementation of a Real-Time Collision Avoidance Method on a Standard Industrial Robot Controller. *In IEEE International Conference on Intelligent Robots and Systems.* Chicago, Illinois, 2014.

After the above-mentioned paper previously unpublished work demonstrating the developed method in an outdoor experiment is presented as an appendix.

Implementation of a Real-Time Collision Avoidance Method on a Standard Industrial Robot Controller

Knut B. Kaldestad*, Geir Hovland* and **David A. Anisi
*Department of Engineering
Faculty of Engineering and Science, University of Agder
Jon Lilletunsvei 9, 4879 Grimstad, Norway.
**Department of Technology & Innovation
Division of Process Automation, ABB
Norway

Abstract — **This paper presents, to the authors knowledge, for the first time real-time collision avoidance implemented on a standard industrial robot controller using point clouds and parallel GPU processing. The algorithms are developed in such a way that minimal modification of existing end-user programs are required to enable the features and benefits of real-time collision avoidance. The proposed approach is successfully demonstrated in an experimental setup consisting of an ABB IRB1600 industrial robot with an IRC5 controller, an enclosed Microsoft Kinect sensor for generating point cloud data and an external PC with a GPU for processing the point cloud and interfacing with the robot controller.**

Keywords — Collision Detection, Industrial Robot, Hidden Markov Model, Expert System.

1 Introduction

Real-time collision avoidance with dynamic obstacles has been demonstrated on experimental research platforms, for example (Kaldestad et al., 2014). With such systems algorithms for real-time avoidance can be implemented at low levels in the controller using advanced programming languages at fast update rates, typically 1ms, (Bischoff et al., June). With an industrial robot controller, however,

end-users usually do not have access to the low-level controller loops and the programming language is limited by the standard robotic application languages, which usually have smaller instruction sets and features compared to languages such as C/C++ and are often interpreted in run-time on the controller and hence slower than compiled programs. To the authors knowledge, real-time collision avoidance including dynamically measured point clouds in 3D and parallel GPU processing is demonstrated for the first time in this paper on a standard industrial robot, in this case an ABB IRB1600 robot with an IRC5 controller. A separate Linux PC with a GeForce GTX TITAN GPU with 2688 cores and an enclosed Microsoft Kinect sensor are used to generate point clouds of dynamic objects in the robot station. The point cloud data is converted to a force acting on the robot's tool and this force is interfaced with algorithms using the standard RAPID application programming language on the IRC5 controller. Although a separate Linux PC is required to process the point cloud data, the solution is implemented in such a way that minimal modification of existing end-user RAPID programs are required to take benefit of the collision avoidance features.

Similar to the work presented in this paper, (Winkler and Suchý, 2011) presented an approach to collision avoidance of industrial robots based on force fields. The force field was generated by virtual charges which were placed on obstacles. The positions of the obstacles were determined continuously by image processing using a camera. The method was implemented using two KUKA KR6/2 industrial robots, where one robot moved in a straight-line path while the second robot dynamically positioned a circular blue object in the workspace of the first robot. A colour filter, step edge detection and the Hough transform were used to process the images with a total computation time of between 200ms and 500ms. The method presented only worked if the colour of the obstacle was different from the surrounding environment, including the robots. Parallel computing (multi-core or GPU) was not attempted.

In (Csiszar et al., 2012) 2D collision avoidance based on forces from a virtual electric field approach was demonstrated. An experimental setup consisting of a KUKA KR500 industrial robot, KR C2 controller, two external PCs and a Sick laser scanner S3000 were used. The laser scanner observed the area around the robot in 2D

(XY) while the height of the obstacle was considered to be infinitely large. The Microsoft Kinect sensor was mentioned as an alternative for extension to 3D, but this sensor was not used in (Csiszar et al., 2012). The volume of the robot itself was not considered, but mentioned as a future extension of the work to improve the collision avoidance. The experimental results in the paper look good, but performance of the system in terms of for example force calculation update rates, communication speeds, etc. was not mentioned.

Apart from the two references above, most open literature related to collision avoidance of industrial robots is based on offline planning and optimisation, see for example (Rambau and Schwarz, 2010) where KUKASim was used to generate the testdata and the software cplex was used to solve mixed-integer optimisation problems for collision avoidance. Another example of offline-based collision avoidance is in (Asakawa and Kanjo, 2013) for welding of large structures. Collision avoidance was based on the application of potential force fields generated by CAD data of the robot and the surrounding environment.

The paper is organised as follows: Section II gives an overview of the proposed approach, Section III presents the developed algorithms in more detail, Section IV shows experimental results while Section V presents final conclusions.

2 Approach

An ABB industrial robot which is located in an industrial laboratory environment generates trajectories based on RAPID robtargets[1] and is modified by virtual forces generated from distances between a vertex model of the robot and observations of the environment. The environment is observed by a Microsoft Kinect sensor and the 3D point cloud (coordinates $[x,y,z]$) is sent to an algorithm which structures and compresses the data into a voxel map. A filtering algorithm is then applied to the data which removes outliers. Further a one-to-one vertex model of the robot is transformed to the position of the robot in the voxel map. This model is used to remove the robot from the point cloud such that only the environment without

[1]Rapid is an ABB propriarity robot programming language. A robtarget is an object containing information, such as position and orientation (quaternion), for trajectory planning and execution.

the robot remains. The transformed model of the robot is then used to calculate reactive forces and momentums on each link based on distances using a potential field algorithm. For more details, see (Kaldestad et al., 2014). As previously mentioned, this paper focuses on the implementation of the GPU/CUDA algorithm applied to a standard ABB industrial robot controller for real-time collision avoidance. With the oil and gas industry in mind, the Kinect sensor has been mounted inside a certified explosion proof casing and the chosen application is the opening and closing of an industrial-type valve. The main contributions of the paper are: 1) implementation and demonstration of a real-time collision avoidance method on a standard industrial robot controller previuously demonstrated only on experimental research systems and 2) the collision avoidance functionality can be added to existing end-user programs with a minimum of modifications - all `MoveL` instructions have to be replaced by `KMoveL` (Algorithm 1) - all parameters to the function call `MoveL/KMoveL` are the same.

3 Algorithms

Algorithm 1 shows the pseudo-code for the algorithm implemented on the IRC5 controller. It communicates with the external PC and interpolates long linear moves into smaller segments. Both position and orientation (quaternion) are interpolated. In Algorithm 1 $\otimes$ denotes multiplication of two quaternions. Fig. E.1a shows the flow-chart description of Algorithm 1.

Algorithm 2 shows the pseudo-code for the algortihm implemented on the CUDA enabled external PC. The GPU algorithm is documented in detail in (Kaldestad et al., 2014). The point cloud data from the Kinect is sampled at 30Hz. The compliance matrix k is 3×3 and is used to convert the virtual forces to adjusted Tool-Centre-Point (TCP) positions. The data exchange between the IRC5 controller and the PC depends on the programmed speed of the robot, since data is exchanged between the interpolated robot movements. Fig. E.1b shows the flow-chart description of Algorithm 2.

Algorithm 2: RAPID Language `KMoveL`: Robot Communication and Movement

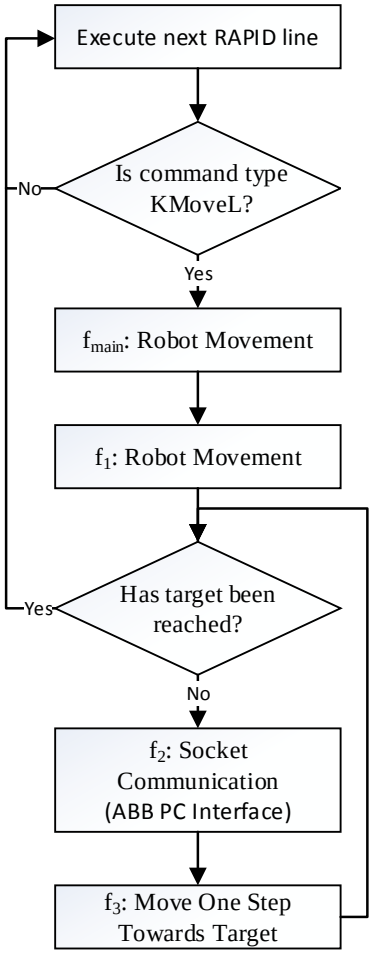
Input Robot:
Step size s (distance)

f_1: **Interpolate Orientation:**
Read current orientation quaternion $\boldsymbol{q}$
Read target orientation quaternion $\hat{\boldsymbol{q}}$
Calculate minimum n steps to target
$\tilde{\boldsymbol{q}}$ = rotQuaternion($\boldsymbol{q}$, $\hat{\boldsymbol{q}}$, n)

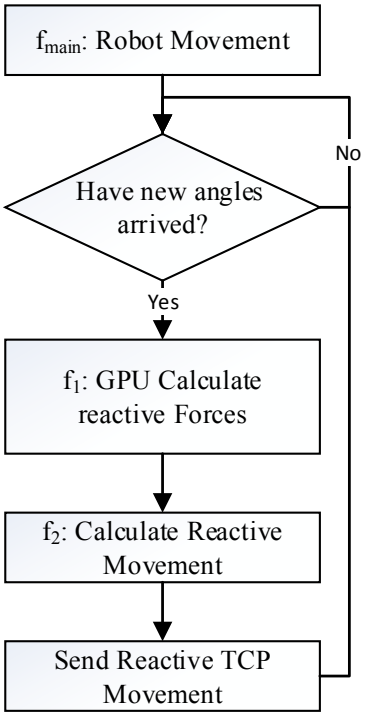
f_2: **Socket Communication (ABB PC Interface):**
Send joint angles $\boldsymbol{\theta}_{\text{out}}$
Receive reactive TCP movement $\boldsymbol{x}_{\text{in}}$

f_3: **Move One Step Towards Target:**
IF $n_{\text{actual}} < n$ **THEN** $\boldsymbol{q}_{\text{next}} = \boldsymbol{q} \otimes \tilde{\boldsymbol{q}}$
ELSE $\boldsymbol{q}_{\text{next}} = \hat{\boldsymbol{q}}$
Read current target position $\boldsymbol{d}$
Calculate unit vector $\boldsymbol{v}$ towards target position $\hat{\boldsymbol{d}}$
$\boldsymbol{d}_{\text{next}} = \boldsymbol{d} + \boldsymbol{v} \cdot s + \boldsymbol{x}_{\text{in}}$
MoveL $\boldsymbol{d}_{\text{next}}, \boldsymbol{q}_{\text{next}}$
$n_{\text{actual}} = n_{\text{actual}} + 1$

f_{main} : **Robot Movement:**
Execute f_1:**Interpolate Orientation**
WHILE $\hat{\boldsymbol{d}}$ not reached:
Execute f_2:**Socket Communication**
Execute f_3:**Move One Step Towards Target**



(a) Robot program as implemented in RAPID on the robot controller



(b) C++ and CUDA C program flow.

Figure E.1: Flowchart of Programs

Algorithm 3: CUDA Enabled Computer

Input:
Spring konstant k

f_1: **GPU Algorithm (Kaldestad et al., 2014)**
Calculate reactive forces $\boldsymbol{F}$

f_2:**Calculate reactive movement TCP**
Read forces acting on TCP $\boldsymbol{F}_{\text{TCP}}$
$\boldsymbol{x}_{\text{out}} = \boldsymbol{F}_{\text{TCP}} \cdot k$

f_{main} : **Reactive Movement:**
WHILE in execution state:
Receive joint angles $\boldsymbol{\theta}_{\text{in}}$
Execute f_1:**GPU Algorithm**
Execute f_2:**Calculate reactive movement** $\boldsymbol{x}_{\text{out}}$
Send reactive TCP movement $\boldsymbol{x}_{\text{out}}$

Figure E.2: *The working environment of the robot including a bottle which acts as an obstacle.*

Figure E.3: *The Microsoft Kinect monted inside an explosion proof casing.*

4 Experiments

Fig. E.2 shows the experimental setup. An ABB IRB1600 robot with a special tool to open and close a valve is used. The industrial-type valve is included for demonstration purposes and is hence not attached to any piping in this experiment. The bottle to the right in the picture is introduced as a dynamic obstacle. The coordinate system used in this paper is the standard global frame of ABB robots. The global X axis is on the floor and parallel with the upper arm of the robot in the home position shown in Fig. E.2. The global Z axis points from the floor to the ceiling and the y-axis equals $y = z \times x$.

Fig. E.3 shows the Microsoft Kinect sensor mounted inside an explosion-proof casing with a glass thickness of 9.6mm. The restricted space inside the casing is accommodated by modifying the Kinect by removing the foot and the casing (except for the front). Additionally cutting parts of the metal bracket including parts of the front casing on each side of the circuit board is necessary for mounting the Kinect such that the infrared projectors field of view (the left most lens in Fig. E.3) is kept to a maximum. The sensor casing allows the proposed solution to be used in ex-zones (for example in the oil & gas industry) as long as the robot and controller

themselves are also ex-proof.

Fig. E.4 (left) shows the robot as the TCP passes above the obstacle, while Fig. E.4 (right) shows the robot when it has reached the final target position and adjusts the valve. Fig. E.5 shows some of the measured positions in the adjusted toolpath (red circles). The blue arrow in Fig. E.5 shows an example of the virtual force vector calculated by the external CUDA-enabled PC.

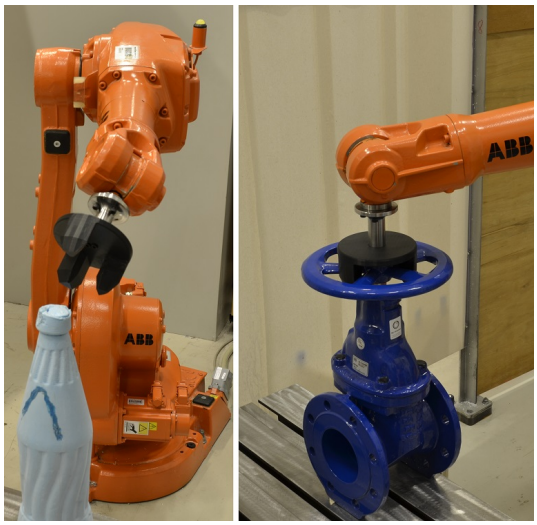


Figure E.4: *The picture on the left shows the robot as it avoids the obstacle. To the right, the robot manipulates a valve.*

Fig. E.6 is generated using rviz (Robot 3D Visualizer) in the Robot Operating System (ROS) (Quigley et al., 2009) and shows the 3D point cloud data generated by the Kinect sensor. The points to the right in the figure represent parts of the bottle, while the points closer to the TCP represent a human entering the robot station. The point cloud of the robot itself is removed before the virtual force is calculated. The green arrow in the figure shows the virtual force at the TCP which in turns generates a position adjustment calculated via a compliance matrix k.

Figs. E.7-E.9 show the 6 measured joint angles as well as the X,Y,Z TCP positions during the experiment. The TCP moves directly above the obstacle (bottle) at about $t = 25$ seconds. The measurements in Figs. E.7-E.9 were recorded by the IRC5 controller at about 100Hz.

Figs. E.10-E.12 show the virtual forces generated on the CUDA-enabled PC from

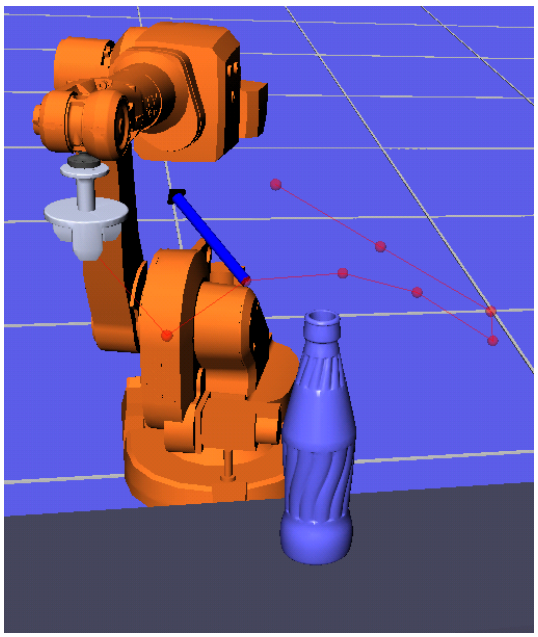


Figure E.5: *Illustration of the robot path during collision avoidance experiment. At the third last point in the figure, the force vector applied at the tool is shown (blue arrow).*

the Kinect's point cloud data. The force vector (blue colour) in Fig. E.5 occurs at about $t = 33$ seconds where the virtual force is negative in the X, Y directions and positive in the Z direction which is also the case in Figs. E.10-E.12. The virtual forces are calculated on the PC at a rate of about 100Hz, while sent to the IRC5 controller at a slower rate, typically at 5Hz to 25Hz, depending on the interpolated step size and the programmed speed.

5 Discussion & Conclusion

This paper has demonstrated real-time 3D collision avoidance on a standard industrial robot controller communicating with a CUDA-enabled external PC and a Microsoft Kinect sensor. Although the techniques used in the paper (artificial force fields generated from 3D point clouds) have been published before on experimental research control systems, to the authors knowledge, this is the first time that the same methods have been demonstrated using a standard industrial robot controller

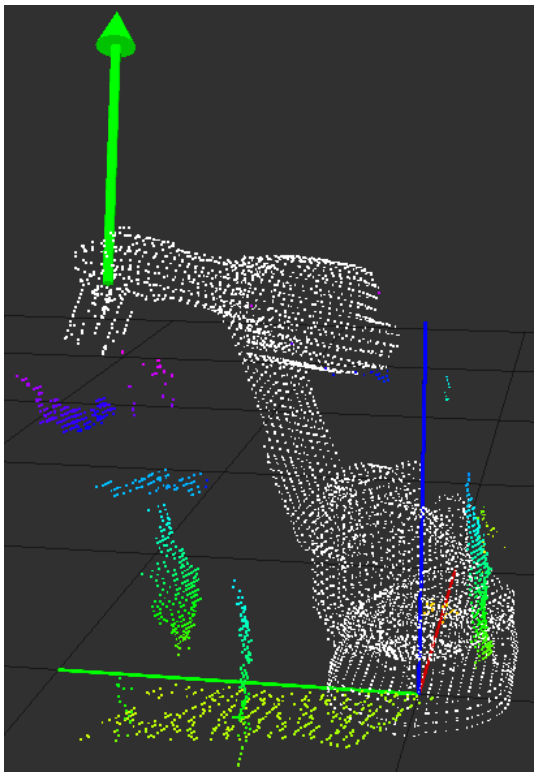


Figure E.6: *Example of virtul force at tool-centre-point generated by point cloud from obstacles.*

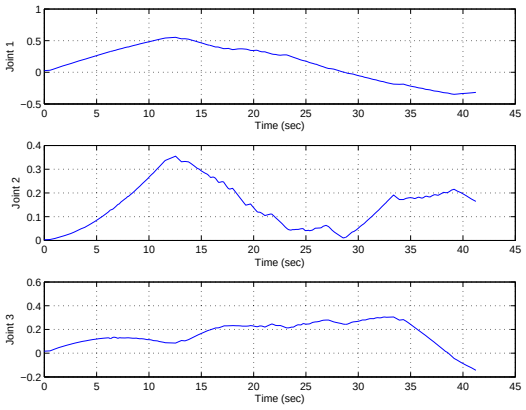


Figure E.7: A Circle

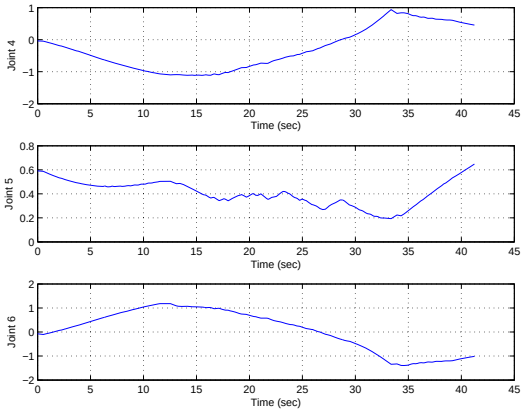


Figure E.8: A Rectangle

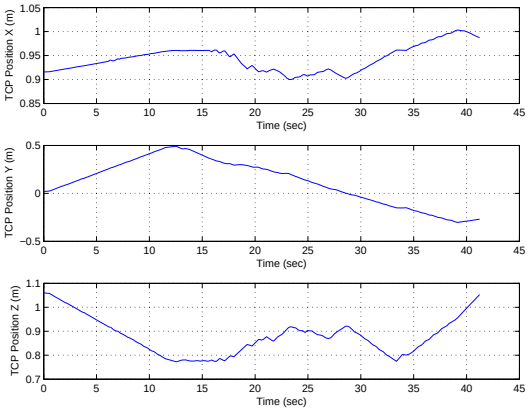


Figure E.9: *X,Y,Z Positions of TCP (m).*

without any special software or hardware modifications.

Of course, the proposed real-time collision avoidance approach could be improved if implemented in the low levels (fast sample rates) of the controller loops by the robot manufacturer. However, such a solution as presented in this paper is currently not available as a standard product. The proposed method in this paper allows current end-users of a standard industrial robot to implement and start using 3D real-time collision avoidance with relatively little effort. End-users simply have to replace all occurences of `MoveL` instructions in their ABB RAPID programs with a new function call `KMoveL` which is implemented as shown in Algorithm 1 and Fig. E.1a. The parameters to the new function `KMoveL` are by purpose kept identical to the parameters of the standard function `MoveL`.

All point cloud measurements in this paper were taken through the glass of the enclosure. The glass did not significantly distort the Kinect measurements, but the small sized enclosure did limit the horizontal field of view of the sensor on the side closest to the infrared projector. The small limitation of the view did not have an impact on this application. A study of the distortion on the point cloud is left to future work.

Future extension of the work presented in this paper will focus on the following: 1) experiments with the Kinect sensor and enclosed casing in an outdoor environment when the sensor is exposed to direct sunlight, rain, snow and ice, 2) handling and avoidance of the robot's wrist singularity and 3) prevent situations where the robot

arms can collide with the robot itself.

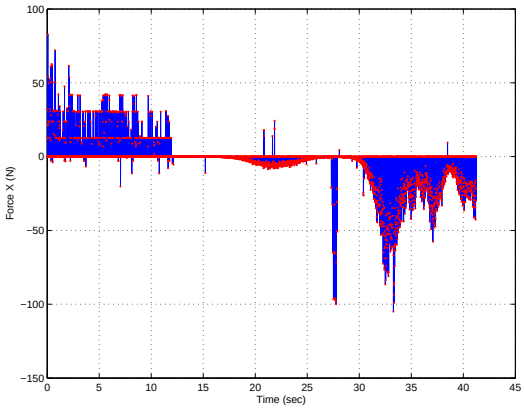


Figure E.10: *Virtual force (blue) and filtered force (red) at tool-centre-point in X-direction.*

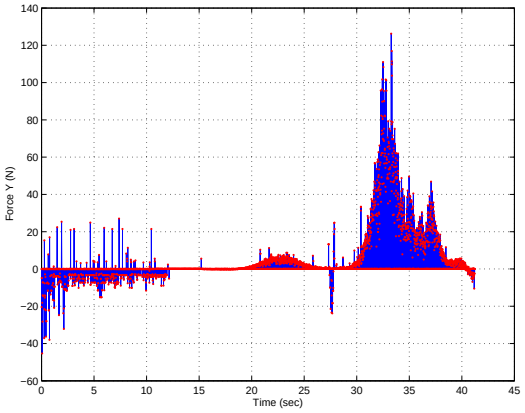


Figure E.11: *Virtual force (blue) and filtered force (red) at tool-centre-point in Y-direction.*

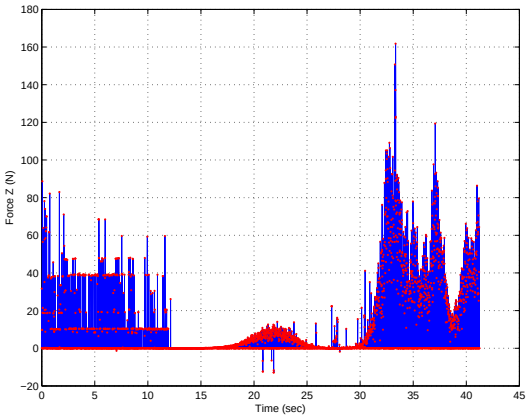


Figure E.12: *Virtual force (blue) and filtered force (red) at tool-centre-point in Z-direction.*

Acknowledgments

The work is funded by ABB and the Norwegian Research Council through project number 193411/S60.

REFERENCES

Asakawa, N. and Kanjo, Y. (2013). Collision Avoidance of a Welding Robot for a Large Structure (Application of Potential Field). *Intl. J. Automation Technology*, 7(2):190–195.

Bischoff, R., Kurth, J., Schreiber, G., Koeppe, R., Albu-Schäffer, A., Beyer, A., Eiberger, O., Haddadin, S., Stemmer, A., Grunwald, G., and Hirzinger, G. (June). The KUKA-DLR Lightweight Robot arm - a new reference platform for robotics research and manufacturing. In *Robotics (ISR), 2010 41st Intl. Symp. on and 2010 6th German Conf. on Robotics (ROBOTIK)*, pages 1–8.

Csiszar, A., Drust, M., Dietz, T., Verl, A., and Brisan, C. (2012). Dynamic and Interactive Path Planning and Collision Avoidance for an Industrial Robot Using Artificial Potential Field Based Method. In *Mechatronics: Recent Technological and Scientific Advances, Part IV*, pages 413–421, doi: 10.1007/978–3–642–23244–2_50, Warsaw.

Kaldestad, K. B., Haddadin, S., Belder, R., Hovland, G., and Anisi, D. (2014). Collision Avoidance with Potential Fields Based on Parallel Processing of 3D-Point Cloud Data on the GPU. In *IEEE Intl. Conf. on Robotics and Automation*, Hong Kong.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, volume 3.

Rambau, J. and Schwarz, C. (2010). How to avoid collisions in scheduling industrial robots? In *Universitätsbibliothek Bayreuth*, Bayreuth, Germany.

Winkler, A. and Suchý, J. (2011). Vision Based Collision Avoidance of Industrial Robots. In *18th IFAC World Congress*, Milano.

Appendix — Previously unpublished work

1 Contribution

The robot considered in this appendix is an ABB IRB 1600 industrial robot, with a standard IRC5 robot controller. In this system, two environments are considered, one indoor and one outdoor environment. The indoor industrial lab environment has good lighting, and consists of a work-bench, a valve and an obstacle (see figure E.2). The environment observer is a Microsoft Kinect which provides data of any obstacle within its view. Even though everything in its strictest sense is an obstacle to the robot, the robot is allowed to interact with objects such as the valve as long as it is the intention of the task. Motivated by the requirements of the oil and gas industry, the Kinect sensor is mounted inside a casing certified for use in explosive environments (see figure E.3). The particular robot model used for the experiments is not ATEX certified, but the collision avoidance methods which are described here, are easily adapted to such robots. The typical system that would benefit from a proper collision avoidance algorithm is any system which has to be able to react to an unforeseen change in the environment surrounding the robot. These unforeseen changes can be a surrounding structure having an unintended change of pose and blocks the path of the robot. But it can also be an intended change, such as a human that interacts with the environment close to the robot, or interacts with the robot itself. Whatever the reason for change in the environment is, a proper collision avoidance system would make any robot setup more safe and robust. The challenges in this type of setup are the quality of the data the sensors provide. One of the challenges with a camera and projector sensor, such as the Kinect, is the limited field of view (FOV) which may require additional sensors to cover the whole environment around the robot. Another problem can be sensor interference, where multiple projectors interfere with each others patterns. In the considered system setups, only one sensor is present. The amount of data that can be manipulated depends on algorithmic requirements, the speed that the processing units execute at and the number of processing units. In this work, highly parallel algorithms are used, and therefore a powerful graphics card is used to handle the

main computation load. Because a standard industrial robot controller is used with as few modifications as possible, the robot controller is the main bottleneck which limits the communication frequency between the controller and the computer which calculates the virtual forces. The second bottleneck is the speed at which the robot controller is able to modify the path and at the same time calculate a smooth trajectory with minimal jerk.

2 Methodology

2.1 Sensor calibration

The initial part of the system setup is to correctly calibrate the sensor with regards to the robot base. Without proper calibration of the sensor, the rest of the algorithm will not work in a predictable way. In order to filter away the robot from the Kinect data, the position of the robot with regards to the Kinect data must be determined. This information is described by a 4x4 transformation matrix, consisting of the position and orientation of the robot base with regards to the Kinect frame.
For the user who is not familiar with the Microsoft Kinect sensor, the front exposes one infrared projector and two sensors. One of the sensors is a RGB camera and the other is an infra-red (IR) depth sensor. There are drivers and libraries for multiple operating systems for acquiring and manipulating the Kinect data. A typical manipulation is to map pixels in the RGB data to a pixel in the IR depth data. The depth data can then be mapped to world coordinates [x,y,z], for further analysis or manipulation.
Figure E.13 shows the program flow of the calibration algorithm. The Kinect is placed in a position which gives sufficient information of the environment. It is important that the sensor remains in the same position with the same orientation after calibration, a slight deviation would require a re-calibration. A calibration plate is mounted on the robot (see figure E.14) and is then faced towards the sensor. The plate has two circles which are used to determine two centre points. The first centre of the plate is mounted in the centre of the tool flange and the other with a vertical offset. The vector between these two points represents the x-axis. The y-axis is normal to the plate, and finally the z-axis is determined from the cross

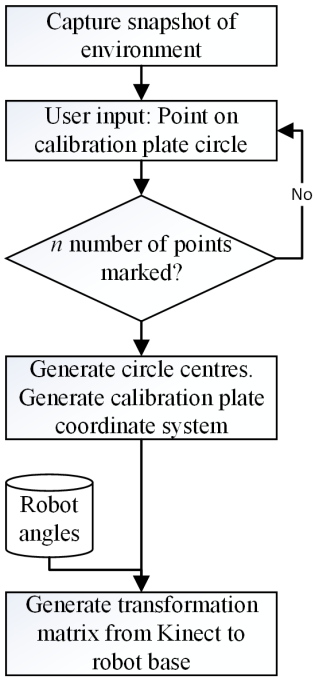


Figure E.13: *Kinect to robot base calibration algorithm*

product between the x- and y-axis. The two centre points are determined by user input to a graphical user interface. The user marks a number of points on each of the two circles, the points from the colour image is transformed to the [x,y,z] depth data and the algorithm estimates the circle centre. It would of course be possible to directly mark the pixel in the centre, but the estimation should yield a better result since the result would not suffer directly from the resolution of the sensor or the ability of the user to click exactly on the centre point. By using only one point, this value would also be very susceptible to noise. After the pose of the calibration plate is determined in the Kinect frame, the transformation from the Kinect frame to the robot base could be determined by the robot kinematics and current joint angles. Even though this calibration currently is semi-automatic, it can be modified to achieve automatic calibration. In the described system, the calibration plate is mounted in the centre of the end-effector, but the positioning of the calibration is not limited as long as the pose is known. For a system which operates in a fully dynamic system, where even the position of the sensor and the robot potentially can change, the calibration plate can be mounted on the tool itself and be of a design

that does not prevent the intended robot motions, to account for each of the six robot joints.

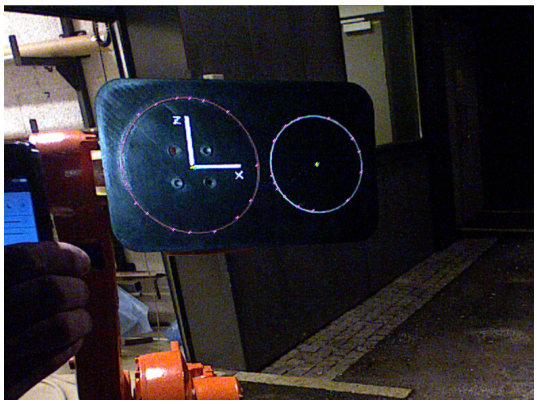


Figure E.14: *Calibration plate with additional markers from graphical user interface for generation of the transformation matrix between the Kinect sensor and the robot base.*

2.2 Sensor calibration

Figure E.1b shows the implementation of the collision avoidance algorithm. This algorithm which runs on a separate computer is communicating with the robot controller through TCP/IP Ethernet. The robot controller can be seen as the master, sending angles to the computer and querying for new forces. As seen from the algorithm, it waits for new angles before it proceeds to calculate the reactive forces. One of the most important steps in generating the reactive forces is to correctly filter out the robot from the Kinect data. This filtering depends on a model of the robot which is transformed by using the forward kinematics using the joint angles. Correct filtration is important because if the algorithm filters out the robot with wrong link poses, parts of the actual robot will remain in the Kinect data. The remaining data should only consist of the obstacles, because they will be fed into the algorithm which calculates the reactive forces. In essence the reactive forces will not only be generated from the obstacles but also from the robot remaining in the data. This can result in very large and highly undesirable reactive forces being sent to the robot controller. The robot can in such cases be perceived to be chasing itself. This

unfortunate scenario can be avoided by initially having proper models of the robot, in addition to have a properly calibrated system with a reliable sensor.

2.2.1 Robot implementation

The user-friendliness for the operator has been the main focus when developing the robot controller algorithm. The algorithm is written in RAPID, which is an ABB propriety programming language for robots. Figure E.1a shows how the RAPID collision avoidance algorithm executes on the robot controller. For the user who would like to enable collision avoidance on a particular robot system, the following steps must be performed:

- Calibrate the sensor, this is done as explained in the Sensor Calibration section.
- Connect the computer to the robot controller to a common network (it is not required that the user has any knowledge of the computer algorithms).
- Copy the robot collision avoidance algorithm, which is a system module, to the robot controller.
- Replace all occurrences of the `MoveL` with `KMoveL` in the RAPID program where the user wants to apply collision avoidance.

When developing the robot algorithm, the focus has been on developing algorithms similar to the existing native RAPID algorithms. Further, for an elegant solution, the PC could be mounted inside the IRC5 robot controller cabinet. Currently only the collision avoidance for the MoveL instruction has been implemented. For instructions such as circular movements (`MoveC`), it is not straight forward to implement collision avoidance. This is because the robot is programmed to follow an arc in one instruction, and might get problems with following an arc which is segmented. For example, if the three points defining the arc lie on a straight line after collision avoidance, the `MoveC` must be replaced by two `MoveL`s. In addition the required interpolation for collision avoidance on the arc would ideally be implemented on a lower level in the control system to achieve the desired performance.

3 Experiments

The experiments are divided into two parts, an indoor and an outdoor experiment. The bottle that acts as the obstacle (see Figure E.4: Left) can only be represented by the surface that the camera observes, which means that the side which is closest to the robot is not represented in the data. This is one of the challenges that has to be met in a one-camera setup and in the following two experiments this lack of information is compensated by increasing the reactive force of the robot. In essence, when the robot tool is on the opposite side of the obstacle, the surface closest to the robot is not observed and the reactive forces are therefore generated by the side facing the sensor.

3.1 Indoor Experiment

Figure E.2 shows the setup of the first experiment. This experiment is conducted indoor in a lab environment, with good lighting conditions. It can be noted that the infrared projector and the camera are susceptible to interference in the operating band, and that good lighting conditions are no direct advantage, but on the contrary could contribute to disturbance. The sensor is placed such that the view plane is more or less parallel with the z-y-plane of the robot (see Figure E.16d) located at about 1.7m from the robot facing in negative x-direction. The computer setup is a multicore Intel Xeon 3.6 GHz processor with 8GB memory and a GeForce TITAN CUDA enabled graphics card with 2688 cores and 6GB memory.

Figure E.4, left side, shows the robot as it avoids the blue bottle which acts as the obstacle. Multiple tests where run with the bottle in different positions, where the robot successfully created a new path around the obstacle and reached its goal position. The right side of Figure E.4 shows the robot as it manipulates an industrial valve. The purpose of the valve is to demonstrate the robot intentionally interacting with the environment, and not avoiding it, since this is a part of the task. The valve serves no functional purpose in this experiment, and is therefore not connected to any piping.

Figure E.5 shows the path of the robot as it avoids the obstacle with final position above the valve (not depicted in the figure). The red line is the path that the robot

followed while avoiding the model of the bottle. In the trials without the obstacle, the lowest point to the right would be connected to the lowest point on the left side by segments creating a straight line.

3.2 Outdoor Experiment

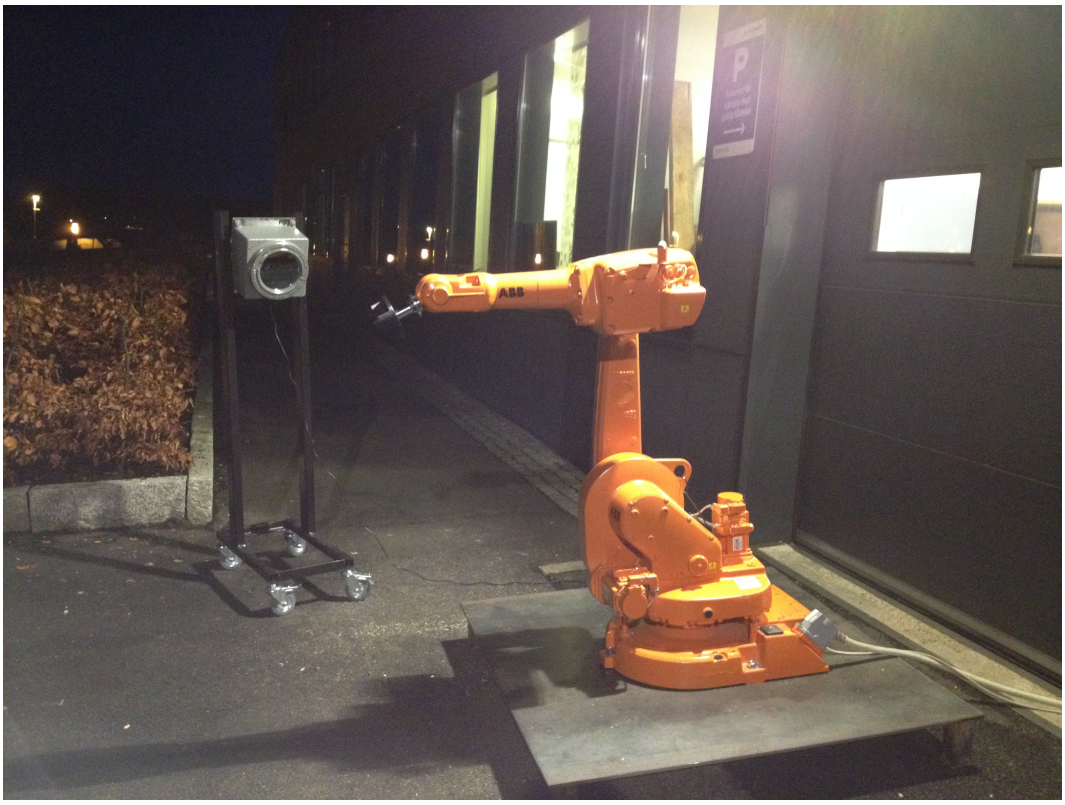


Figure E.15: *ABB IRB 1600 robot in outdoor environment.*

The second experiment is conducted outside to introduce effects not present in the indoor environment. The temperature is at sub-zero ^{o}C and it is dark outside. In this experiment the Kinect is not exposed to sunlight which might interfere with the projected infrared pattern projected and read by the sensor. As the work by (Rasmussen, 2012) demonstrates, the sensor can operate well in low sunlight, but increasing brightness decreases the amount of useful depth data. Figure E.15 shows the outdoor setup where the same robot is used as in the indoor experiment. The programmed robot path goes from point (1) (see figure E.16d), to point (2). The

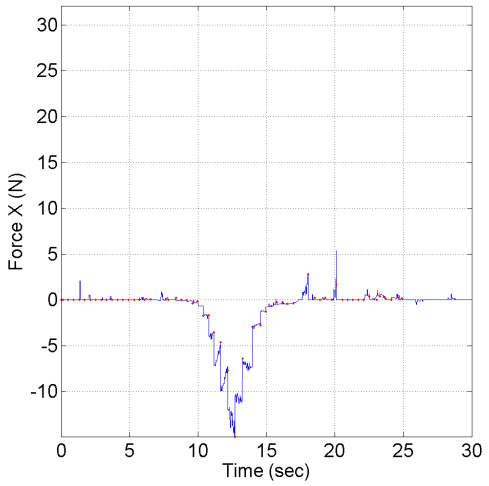
obstacle (3) is placed between point (1) and (2), and is therefore blocking a straight line path between these points. Since the obstacle blocks this path, it is left for the obstacle avoidance algorithm to generate a new trajectory such that collision is avoided. Figure E.16a— E.16c show the reactive forces generated by the obstacle. The x, y, z forces shown in figure E.16a-E.16c at time 13.5s correspond to the green force vector illustrated in figure E.16d. As seen in the figure, the green force vector acts in the opposite direction of the obstacle (the bottle shown in green-yellow-orange colours).

4 Conclusions

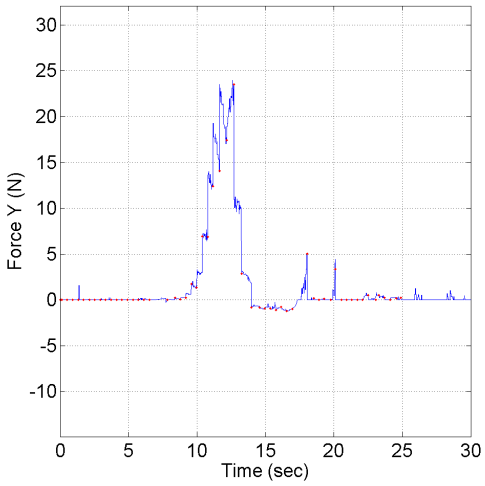
The work in this appendix has demonstrated that it should be possible to achieve real time robotic collision avoidance in industrial explosive zones in the not too distant future. The experiments demonstrate that it is possible to perform collision avoidance indoor and outdoor with 3D sensor equipment in compliance with ex-standard and using a standard off-the-shelf industrial robot. Both experiments show avoidance of an obstacle on a standard industrial robot controller, without modifications. It is however necessary with an additional computer with a GPU, but this additional hardware could be mounted inside the controller cabinet and hence the overall system would appear as one unit.

While the experiments demonstrate the potential of the system, there are still challenges left to resolve for the robot's reactive movement, such as getting stuck in certain positions depending on the forces. But even though the algorithms are very important on different layers, e.g. collision avoidance and goal position fulfilment, the data that some of the algorithms operate on are initially provided by sensors. A second sensor-related challenge is to cover the whole environment. Even if these two aspects currently are challenges, the sensor technology related to accuracy and field of view are certainly expected to improve. The advantages with the algorithms are that they operate on point clouds which enable sensor fusion in an easy manner. Any sensor capable of delivering point cloud data could be integrated.

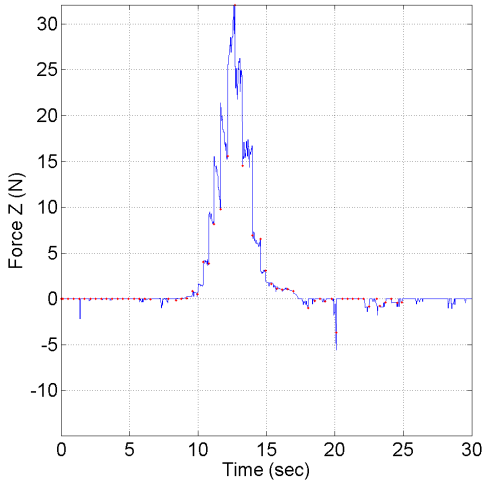
GPUs have currently an inherent advantage as new models with increasing number of cores appear year by year. This is in contrast to the more or less comparable CPU



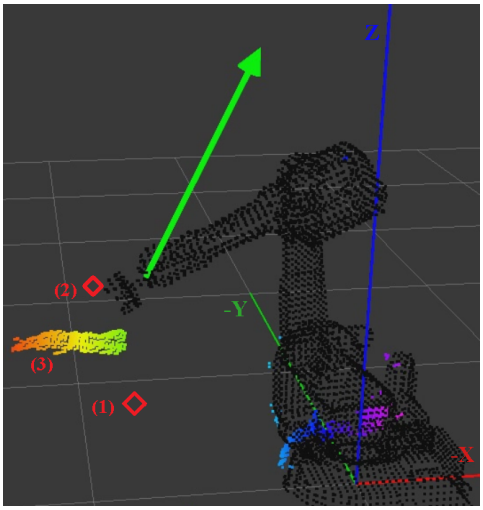
(a) Virtual forces acting in the direction of the x-axis.



(b) Virtual forces acting in the direction of the y-axis.



(c) Virtual forces acting in the direction of the z-axis.



(d) The robot vertex model and the coloured obstacle to the left. The green vector is the generated force. Snapshot taken at approximately 13.5 seconds.

Figure E.16: *Outdoor experiment results.*

frequency which has seen no significant increase since 2004 when the 3.8 GHz Intel Pentium 4 HT 570J was released. To justify this comparison, the GPUs advantage is massive parallel processing, and the CPU is still mainly serial processing. The serial performance of the CPU (mostly frequency dependent) is currently not increasing at the same rate as the parallel performance of the GPU.

The proposed solution only exposes one new function to the user, here named `KMoveL`, in order to enable collision avoidance. In addition the function has the exact same function parameter interface as the standard `MoveL` function. From the user perspective this new function could be implemented in any existing program with minimal effort, only introducing "K" in front of the desired `MoveL` function. However, the required interpolation for the new functionality should ideally be implemented by the robot manufacturer on a lower level in the control system to achieve improved performance.

REFERENCES

Rasmussen, C. (2012). Kinects for low-and no-sunlight outdoor trail-following. *RGB-D: Advanced Reasoning with Depth Cameras in conjunction with RSS*.