An Autonomous and Vendor Independent Robot-based Palletizing System


Sigurd Salvesen

August 10, 2009
Abstract

Robots has during the last decades taken an important role in the industry. These industrial robot systems are doing predefined jobs over and over again. One typical example is a palletizing system, where robots are used to pick up packages from the end of a production line, and thereafter placing it on a pallet for the hence of transport. Many problems could occur, like for instance what to do when new and unknown packages of different size and shapes occurs at irregular intervals at the production line? The system may not tackle this. Situations like this could be dangerous for humans.

A robot could be looked at with three aspects - mechanical, electrical and software. The latter one can be unnecessarily complicated since the robot manufactures often develop their own programming languages and development tools. This gives the robot manufactures control over their products, which is positive in many ways, e.g. stability, security, performance etc. From a consumer’s view this unfortunately tends to be quite expensive - only highly qualified personnel could install and upgrade the system and you may not use spare parts or new products from other robot manufactures.

With the goal to develop an autonomous robot-based system that should tackle these problem, the task is divided into three parts:

Robot frameworks - The system should be able to support robots from different manufactures with as little adjustment as possible. Therefore the control system is moved to a standard computer. What are the pros and cons? What about the lack of standards for robot communication and control in the robotic industry?

Vision system - To be able to tackle new and unknown packages of different size and shapes occurring at irregular intervals at the production line they have to be detected in some way. The implemented vision system uses two standard webcams, in contrast to more traditional machine vision solutions where special sensors and sophisticated cameras are used. Its task is to detect new packages, and measure their height, width and depth.

Packing algorithms - Traditional pallet loading systems place packages by some predefined schemes. Since the system should be autonomous, a algo-
rithm for automatically generating a good pallet scheme (in reasonable time) is implemented.

The control system and vision system is developed mainly using the Cycling ’74 Max and Jitter for the graphical interface, vision system and state machine, and Java for the packing algorithms and robot communication (SOAP). It has been emphasized to create objects that can easily be modified or replaced when needed.

The presented solution could run under Microsoft Windows and Mac OS X on different hardware. The robot communication has to be replaced or modified if another robot than Motoman NX100 and the NX100 robot controller is to be used.
Preface

This report describes my work undertaken for my Master Degree in Electronics and Computer Technology, Microelectronics from the University of Oslo. The work is handed out at the research group Robotics and Intelligent Systems (ROBIN) at the Department of Informatics.

First of all, I would like to thank my supervisor, Associate Professor Mats Høvin, for excellent guidance, creative inputs, and for always inspiring me. Thanks also to my co-supervisor, Associate Professor Omid Mirmotahari, for his constructive inputs and guidance whenever Mats was not around.

Last, but not least, I want to thank my family and my dear Ragnhild Marie for their support and for always believing in me.

Sigurd Salvesen
August 10, 2009
Oslo
# Contents

1 Introduction ................................................. 1
   1.1 Background for the Thesis .......................... 1
   1.2 Goals for the Thesis .................................. 1
   1.3 Introduction to Robotics ............................ 2
   1.4 Introduction to Vision Systems .................... 2
   1.5 Introduction to Packing Algorithms ............... 3
   1.6 Thesis Overview ...................................... 3
   1.7 Short Conclusion ..................................... 4

2 Background ..................................................... 5
   2.1 Robotics ................................................ 5
      2.1.1 Programming Robots .......................... 5
      2.1.2 Vendor Specific Software ..................... 6
      2.1.3 Microsoft Robotics Developer Studio .......... 6
      2.1.4 Real-Time ....................................... 7
      2.1.5 Product Example: FPT Standard Palletizer .... 7
   2.2 Vision Systems ....................................... 8
      2.2.1 Traditional Machine Vision ................... 10
   2.3 The Pallet Loading Problem .......................... 10
      2.3.1 Traditional Pallet Schemes ................... 12
      2.3.2 The Distributor's Pallet Packing Problem .... 12
      2.3.3 Manufacturer's Pallet Loading Problem ....... 12
      2.3.4 Fundamental Heuristics and Algorithms ..... 12

3 Used Tools ..................................................... 15
   3.1 Motoman IA20 ......................................... 15
      3.1.1 Working Area & Physical Dimensions ........ 16
      3.1.2 Technical Facts ................................ 16
      3.1.3 Package Gripper ................................ 16
   3.2 Motoman NX100 ....................................... 18
      3.2.1 Communication Features ....................... 18
3.2.2 Additional Features .................................................. 19
3.2.3 Touch Screen Pendant ............................................... 19
3.3 Cycling ’74 Max/MSP/Jitter ............................................. 19
   3.3.1 Introduction to Max ............................................. 20
   3.3.2 Introduction to Jitter ........................................... 21
   3.3.3 Robot Control .................................................... 24
   3.3.4 Pure Data - An Alternative to Max ......................... 24
3.4 Java ........................................................................... 24
   3.4.1 Why Choose Java? ............................................... 25
   3.4.2 Java Runtime in Max ............................................ 26
   3.4.3 Sun Java Real-Time System .................................. 26
3.5 SOAP ......................................................................... 27
   3.5.1 Skeleton of a SOAP Message ................................. 27
   3.5.2 Sample SOAP Messages ........................ ................. 28

4 Robot Control System .................................................... 29
   4.1 System Overview ..................................................... 29
   4.2 Moving Software from Specialized Hardware to a Standard Personal Computer .................................................. 29
      4.2.1 Quality ............................................................ 31
      4.2.2 Needs for Real-Time ......................................... 32
   4.3 Selecting Development Framework ............................. 32
      4.3.1 Microsoft Robotics Developer Studio .................. 32
      4.3.2 Cycling ’74 Max and Jitter ................................. 33
   4.4 Graphical User Interface ............................................ 34
   4.5 State Machine .......................................................... 34
      4.5.1 The States ....................................................... 35
      4.5.2 Concurrency and Synchronization ....................... 35
      4.5.3 Global Data .................................................... 38
   4.6 Robot Communication ................................................. 39
      4.6.1 Synchronization ............................................... 39
      4.6.2 Programming the Robot Controller ..................... 40
      4.6.3 Communication Handler - A Max External ............ 42
   4.7 Pick and Place Packages ............................................. 43
      4.7.1 A Test Setup ................................................... 47
   4.8 Summary ................................................................. 47

5 Vision System .............................................................. 49
   5.1 Traditional Solutions ................................................ 49
   5.2 A Two Camera Solution .............................................. 50
      5.2.1 Camera 1 .......................................................... 50
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Product example: FPT Standard Palletizer (©FPT [4])</td>
<td>8</td>
</tr>
<tr>
<td>2.2</td>
<td>Some topics related to vision systems</td>
<td>9</td>
</tr>
<tr>
<td>2.3</td>
<td>An example of an advanced modern vision system: Mars Exploration Rover with stereo sight (NASA/JPL/Cornell University)</td>
<td>11</td>
</tr>
<tr>
<td>2.4</td>
<td>Illustration of the Next Fit heuristic and a block algorithm</td>
<td>13</td>
</tr>
<tr>
<td>3.1</td>
<td>Motoman IA20 robot</td>
<td>16</td>
</tr>
<tr>
<td>3.2</td>
<td>Motoman IA20's dimensions [6]</td>
<td>17</td>
</tr>
<tr>
<td>3.3</td>
<td>Motoman IA20's working area [6]</td>
<td>17</td>
</tr>
<tr>
<td>3.4</td>
<td>Remote device for the Motoman NX100 robot controller</td>
<td>18</td>
</tr>
<tr>
<td>3.5</td>
<td>Motoman NX100 robot controller - Dimensions [9]</td>
<td>19</td>
</tr>
<tr>
<td>3.6</td>
<td>Max/Jitter example: Subpatch &quot;video_grab&quot;</td>
<td>20</td>
</tr>
<tr>
<td>3.7</td>
<td>Max/Jitter example: Patcher Mode</td>
<td>22</td>
</tr>
<tr>
<td>3.8</td>
<td>Max/Jitter example: Patcher Mode - Locked</td>
<td>22</td>
</tr>
<tr>
<td>3.9</td>
<td>Max/Jitter example: Patching presentation</td>
<td>23</td>
</tr>
<tr>
<td>3.10</td>
<td>Max's Java Runtime</td>
<td>26</td>
</tr>
<tr>
<td>4.1</td>
<td>System overview</td>
<td>30</td>
</tr>
<tr>
<td>4.2</td>
<td>Main window of the control system developed in Max (patch: robot-based_pallet_loading_system)</td>
<td>34</td>
</tr>
<tr>
<td>4.3</td>
<td>Flowchart over the system</td>
<td>35</td>
</tr>
<tr>
<td>4.4</td>
<td>The state controller patch (patch: state_controller)</td>
<td>36</td>
</tr>
<tr>
<td>4.5</td>
<td>State synchronization (patch: semaphore)</td>
<td>37</td>
</tr>
<tr>
<td>4.6</td>
<td>State synchronization (patch: semaphore2)</td>
<td>37</td>
</tr>
<tr>
<td>4.7</td>
<td>Source code NX100-program with semaphore illustration</td>
<td>40</td>
</tr>
<tr>
<td>4.8</td>
<td>Pick package (patch: robot_pick_package)</td>
<td>45</td>
</tr>
<tr>
<td>4.9</td>
<td>Place package (patch: robot_place_package)</td>
<td>46</td>
</tr>
<tr>
<td>4.10</td>
<td>A test setup with the 7 axis Motoman IA20 snake robot</td>
<td>47</td>
</tr>
<tr>
<td>5.1</td>
<td>Vision overview</td>
<td>51</td>
</tr>
<tr>
<td>5.2</td>
<td>Camera control window (patch: camera_control)</td>
<td>53</td>
</tr>
<tr>
<td>5.3</td>
<td>Grab video matrix (patch: grab_above)</td>
<td>54</td>
</tr>
</tbody>
</table>
5.4 Demonstration of the principle of optical flow. Generated by Doug Hatfield (Wikipedia [22]).

5.5 Detect package (patch: vision_detect_package)

5.6 Measure aspect ratio depth vs. width (patch: vision_measure_depth)

5.7 Results of morphological operations: a) Original image. b) Dilated image. c) Eroded image. d) Closed image. e) Opened image.

5.8 Video input type: normal (patch: large_view)

5.9 Video input type: black & white (patch: large_view)

5.10 Video input type: threshold (patch: large_view)

5.11 Video input type: open (patch: large_view)

5.12 Video input type: binedge (patch: large_view)

5.13 Large preview of camera 1 (patch: large_view2)

5.14 Measure height and width (patch: vision_measure_width_height)

5.15 Calculate height (calc_height)

5.16 Calculate width (patch: calc_width)

5.17 Measure depth (patch: calc_depth)

5.18 Calculate placement (patch: calc_placement)

5.19 OpenCV error message

5.20 A test setup of the vision system

5.21 White box used during tests of the vision system

5.22 Milk used during tests of the vision system

6.1 Packing scheme - Next Fit

6.2 Packing scheme - Next Fit with all packages rotated

6.3 Packing scheme - Next Fit with optimal row rotation

6.4 Packing scheme - 4 block algorithm

6.5 4 block structure

6.6 Packing algorithm in Max (patch: package_find_placement)

6.7 2D view of a pallet scheme

6.8 3D view of a pallet with one package

6.9 3D view of a pallet with many packages

A.1 Max patch: robotbased_pallet_loading_system

A.2 Max patch: state_controller

A.3 Max patch: semaphore

A.4 Max patch: semaphore2

A.5 Max patch: camera_control

A.6 Max patch: grab_above

A.7 Max patch: video_preview_above

A.8 Max patch: grab_front

A.9 Max patch: video_preview_front
List of Tables

5.1 Items used during tests of the vision system 66
5.2 Test results of the vision system 66
6.1 Test results of the packing algorithms 81
Chapter 1

Introduction

1.1 Background for the Thesis

Loading packages on a pallet for transport was traditionally a job carried out by humans beings. The job could often be heavy and monotonous. During the last decades industrial robots of various kind has replaced the human operators. There are several benefits to be gained, like reduction in number of employees, avoid monotone and non-ergonomics working position, higher speed and perhaps higher quality (in the means that the work is done the exactly same way over and over again).

There are however some disadvantages to this robotification. Fore the first the robot are often programmed to place just known types of packages by following some kind of placing-map of the pallet made in advance. Advanced runtime algorithms for generating such placing-maps may even find new and better solution than a human brain could come up with.

Traditional robotic systems perform only predefined tasks. When unforeseen conditions arise the robot (hopefully) stops. If not, it could be dangerous to its surroundings, which may include people. One of the most typical example perhaps of such a system - a robot-based system for loading packages on a pallet, is the basis of this thesis.

1.2 Goals for the Thesis

The project's goal is to evaluate a control system for automatically handling packages from a production line, and placing them on a pallet. The system or more precise an experimental version of it, will be developed as a part of this thesis for the evaluation purpose. One of the goals is to look at possibilities for
making the system autonomous. This means that the system should be able to find solutions to unfamiliar problems. Primary to this is to find out how packet handling and placement can be conducted even if there is new types of packages arriving at the production line. A system like this could be used in the manufacturing industry, which creates large quantities of the same product. Algorithms for this problem area is known under the term *The Manufacturer's Pallet Loading Problem* (MPLP).

The second goal is to make the control system vendor and platform independent. Many robot manufacturers use their own standards and create their own programming languages and development tools. This makes the development process and upgrades unnecessarily expensive since it often only could be handled by high-skilled technicians. It is desirable that the control system should be vendor independent when it comes to such things as computers, operating systems and cameras. Robots from different vendors with necessary adjustments should be easy to take in use.

### 1.3 Introduction to Robotics

The word *robot* was introduced by the Czech artist Josef Čapek, and was later popularized by his brother Karel Čapek in the play *Rossum’s Universal Robots* (R.U.R.) in 1920 [16]. The word *robota* derives from *robota* which means work in Czech [13].

The Robotics Institute of America (RIA) has the following definition of a robot: "A robot is a reprogrammable multifunctional manipulator designed to move material, parts, tools, or specialized devices through variable programmed motions for the performance of a variety of tasks." [12]. From a technical point of view Jazar [12] describes robots as complex, versatile devices that contain a mechanical structure, a sensory system, and an automatic control system.

Industrial robots has during the last decades replaced more and more workers in the industry. Employees doing monotone and repeated tasks could easily be replaced by robots. The robots could do these tasks even faster and better, which in addition to reduced salary costs by time can be quite profitable. As the technology develops, more complex work could be handed by robots.

### 1.4 Introduction to Vision Systems

A robot has to interact with its environment to be able to do its job. To be able to interact, the robot must be able to sense necessary information from
its environment. Robotics applications needs to "see" where to pick and place objects, measure distance etc.

Traditionally different types of sensors have been used in robotics application [13]. Today these have often been replaced with low-priced and powerful digital camera chips. Since computing resources normally is not a limit, all this together makes computer vision one of the most powerful sensing modalities that currently exists.

From a digital camera source an object can be detected by its shape, colour or combination of shape and colour [1].

1.5 Introduction to Packing Algorithms

Packing is often done by some predefined schemes which tells where to place packages on the pallet. These schemes traditionally has to be designed and thereafter loaded into the robot control system. Thereby the system has to be updated at least every time a new package is introduced.

There are two main categories of packing problems; The Distributor’s Pallet Packing Problem (DPPP) and The Manufacturer’s Pallet Loading Problem (MPLP). The first one is the problem of loading a pallet of inhomogeneous packages coming off a production line, while in the latter one the packages are equally sized.

The search space for these problems can be quite enormously...

The main reason for searching for new and better packing algorithms is to reduce shipping costs.

1.6 Thesis Overview

In the following chapter more detailed and related background material for robotics, computer vision and packing algorithms will be discussed.

A control system for the robot is presented in chapter 4, including state machine and robot communication. In addition you will also find an overview of the project as a whole.

The 5th chapter deals with the vision system. A solution based on two standard webcams associated with processing in Max will be presented. This is followed by a chapter about package algorithms, which includes implementations of simple packet algorithms, and issues related to run-time generation of new palletizing schemes.

In the last two chapters you’ll find suggestions for further work, and a summary and a conclusion.
1.7 Short Conclusion

A control system for the robot is implemented in Cycling ’74 Max/MSP/Jitter with some components written in JavaScript and Java. The vision system is tightly integrated into the control system, and uses 2 standard webcams to detect and measure the size of arriving packages. Finally, a runtime generation of palletizing schemes functionality is also added, with both 2D and 3D visualization.

The system has been more vendor independent, since most of the code is placed on a standard modern computer. The implementation is attempted develop as abstract as possible. All this will (hopefully) result in that the required product specific adaptations will be as small as possible.

The autonomic advantages of the system is that it can detect new types of packages and automatically generate palletizing schemes of good quality.

All source code and screenshots of the Max patches can be found in the appendix at the end of this thesis.
Chapter 2

Background

This thesis covers three different research fields; robotics control systems, machine vision and packing algorithms. Each of them will be presented in this chapter.

2.1 Robotics

Robotics has a lot in common with computer science and programming, but it is often found to have a high barrier to enter because the term interoperability doesn’t exist. Every robot vendor often makes their own robots with specialized robot controllers, programming languages, development software etc. There is a lack of reusable components, basically since no standard hardware or hardware abstractions exist. Concurrent and distributed programming is hard and is a necessary part in most robotics software projects. Testing in the real-world is excessively costly and advanced simulation can be complex.

In this project I will try to make a vendor independent system where it would be possible to replace hardware (robot, PC, cameras) and operating system, and still reuse as much as possible of the code.

2.1.1 Programming Robots

Controlling robots traditionally includes advanced mathematics (kinematics) for moving the robot to a defined place in a defined path.

Modern robots often have the possibility for moving the robot by giving X, Y and Z-coordinates. In a XYZ-coordinate system one define the X, Y and Z axis data for placing the robot head, while the Tx, Ty and Tz axis data are used to rotate the tool mounted on top of the robot’s head.
2.1.2 Vendor Specific Software

Most robot manufacturers keep their software hidden [21]. It is impossible to find out how most robots are programmed. And it is almost as if they had no software in many cases. This is because these companies do not intend their customers to do their own programming and they sell complete proprietary application packages rather than general purpose software.

There is very little doubt that robots will be powered by PC technology - even on the industrial side, we are seeing more and more robots that use PC-based technology. As this industry moves forward it will take advantage of technology that already exists on the PC.

In computer science the term interoperability describes the ability of software and hardware on different machines from different vendors to share data. At the time being this term seems to be non-existing within the robot industry.

2.1.3 Microsoft Robotics Developer Studio

One of the newest vendor independent development frameworks for robotics is the Microsoft Robotics Developer Studio (MRDS) [11]. It makes use of standard programming languages like C#, with some add-ons for required robot facilities. The main benefits could be summarized as follows:

- Reusable components: Introduces a paradigm with reusable components (web-services) that facilitates reuse.
- Standardization: Introduces a hardware abstraction paradigm.
- Concurrency and distributed computing: Introduces Concurrency and Coordination Runtime (CCR) and Decentralized Software Services (DSS) for simplifying such tasks.
- Simulation: An advance physical simulation package, making it possible to program robots without having access to one.
- Visual programming: Makes the advanced features more accessible to newcomers.

Concurrency and Coordination Runtime

The Concurrency and Coordination Runtime (CCR) makes it simple to write programs to handle asynchronous input from multiple robotics sensors and output to motors and actuators. It enables programmers to create applications that have a high level of synchronicity, so several processes can run in parallel.
In the modern computer technology when you have multicore computers, and the ability to really use this new type of computing power depends on programmers being able to write applications that can have their parts running with a high level of concurrency. CCR is designed with this aim.

**Decentralized Software Services**

Decentralized Software Services (DSS) is a lightweight .NET-based runtime environment that sits on top of the Concurrency and Coordination Runtime. It provides a state-oriented service model that combines the notion of representational state transfer with a system-level approach for building high-performance, scalable applications.

**Reuse of Services**

In MRDS you can build high-level functions using simple components, providing for reuse of code modules as well as better reliability and replaceability. For example, a lower-level sensor service could be integrated into a navigation service. This is done mainly by creating and programming web-services. It is easy to imagine that such web-services often completely or partly can be reused, modified or replaced later.

**2.1.4 Real-Time**

Many developers typically think that real-time means "really fast" as in moment-to-moment interaction with the system. Real-time means the ability to reliably and predictably respond to a real-world event. So real-time is more about timing than speed, but of course many real-time systems are quite fast! The ability to program a real-time solution means that developers need the appropriate support from the system in order to reason in a temporal fashion. Without such support, this can be very time consuming, if not impossible.

In an robotic installation real-time may be important for handling critical actuators or sensors. Especially when it comes to safety, and communication with other robots or systems.

**2.1.5 Product Example: FPT Standard Palletizer**

As an example of a modern palletizing system, I've picked the FPT Standard Palletizer [4] (see figure 2.1 on the following page), which is a state of the art palletizing system. It is a complete system including software, robot, claw etc.
The manual says: "Easy creation/retrieval of the palletizing scheme by means of the user-friendly FPT Flex Pack software" [4].

Could it be possible to let the packing algorithms generate the palletizing scheme by itself? Is it possible to create software that can run on different computers, and manage a robot from almost any manufacturer?

2.2 Vision Systems

Of our five senses is vision undoubtedly what we are most dependent on. Computers are not able to "see" in the same way as us. While people can rely on inference systems and assumptions, computers must "see" by examining the individual pixels of images, processing them and try to develop conclusions. A vision system may be used to recognize and locate objects, and perhaps their motion. Object can be detected by its shape, colour or an combination of shape and colour [1]. Vision systems covers a lot of areas in computer science, mathematics and physics (see figure 2.2 on the next page for an overview).
Figure 2.2: Some topics related to vision systems
Computer vision (CV) is mainly focused on machine-based image processing [23]. Machine vision (MV) is the application of computer vision to industry and manufacturing, and often requires input/output devices and computer networks to control manufacturing equipment such as robotic arms [23].

Machine vision and computer vision systems are capable of processing images consistently, but computer-based image processing systems are typically designed to perform single, repetitive tasks, and despite significant improvements in the field, no machine vision or computer vision system can yet match some capabilities of human vision in terms of image comprehension, tolerance to lighting variations and image degradation.

For this project the machine vision system will be used to detect new packages and measure their sizes. In this way the system does not have to know what types of packages that will have to be picked up from the assembly line.

2.2.1 Traditional Machine Vision

A typical machine vision system is often quite technically quite complicated and consists of many parts. It often consist of one or more camera. A embedded processor or a personal computer to process images and sensor data. Analog or digital input/output hardware or communication links to report results. Sensors (optical, magnetic, mechanical) is often used to trigger image acquisition and processing.

2.3 The Pallet Loading Problem

Packing problems can be thought as an combination of mathematics and puzzles, and it consists of filling up one or several pallets or containers with as many packages as possible. This can be done with a search algorithm. A search algorithm is an algorithm that takes a problem as input and returns a solution to the problem, usually after evaluating a number of possible solutions. The set of all possible solutions to a problem is called the search space, which in packing problems can be quite enormous.

A brute-force search algorithm searches through the entire search space, which make the process quite time-consuming. Research during the last decades have tried to develop informed search algorithms by using e.g. heuristic functions to apply knowledge about the structure of the search space, with the purpose of trying to reduce the amount of time spent searching.
Figure 2.3: An example of an advanced modern vision system: Mars Exploration Rover with stereo sight (NASA/JPL/Cornell University)
2.3.1 Traditional Pallet Schemes

Packing is often done by some predefined schemes which tells where to place packages at the pallet. These schemes traditionally has to be designed and thereafter loaded into the robot control system. This makes it necessary to update the system every time a new type of package is introduced.

2.3.2 The Distributor's Pallet Packing Problem

The Distributor's Pallet Packing Problem (DPPP) is the problem of loading a pallet of non-homogeneous items coming off a production line. DPPP is NP-complete which means that it is being complex and difficult to solve within a reasonable time span. Common heuristics for the DPPP typically decompose the problem into two sub-problems; one of pre-scheduling the items on the production line and one of packing the items on the pallet.

When it comes to finding the optimal placement of the package at the pallet, there are two main problems. For the first, you may have none or little knowledge on the next arriving packages. Secondly the search space for finding an optimal (or as good as possible) solution is enormous.

2.3.3 Manufacturer's Pallet Loading Problem

Manufacturer's Pallet Loading Problem (MPLP) consists on arranging (orthogonally and without overlapping) the maximum number of identical (rectangular) packages into a larger rectangle (e.g., a pallet). Since all the packages are identical you have a complete knowledge of the arriving packages, and you may concentrate on finding an optimal packing solution. Although the packing of a pallet is a three-dimensional problem, it can be regarded as a two-dimensional task since all the layers are equal.

2.3.4 Fundamental Heuristics and Algorithms

There are many algorithms and heuristics for the MPLP. The most fundamental are the Next Fit (NF) and block algorithms.

Next Fit

The Next Fit (NF) may be used both for the Distributor's Pallet Packing Problem and the Manufacturer's Pallet Loading Problem. It has no knowledge about the arriving items. The algorithm may be implemented using a level technique for
arranging the packages in rows. The packages are placed one after one in a row until it is full, and then it will start on a new row. If used on DPPP, the highest item on the row determines the height of the next row, making the packing fitness worse in cases with large variability in package sizes.

**Block Algorithms**

The recursive block algorithms is basically dividing a rectangle into regions (normally no more than five) through first-order non-guillotine cuts, as showed in figure 2.4. Packages can be placed inside these blocks. More advanced variations of the algorithm are working recursively for each region. The size given to each block may have great significance for the outcome of the packing fitness.
Chapter 3

Used Tools

In this chapter the essential development tools and robotic hardware used in the experiments will be presented. First, the Motoman IA20 robot is described. The IA20 will later be used to carrying out some practical demonstrations and tests. The IA20 robot is controlled by the Motoman NX100 robot controller which is more thorough described. An NX100 program will later on be developed for being able to control the robot from an computer connected to it.

The Cycling '74 Max/Jitter framework is also described. Later on an (machine) vision system for package detection and a control system is developed in Max/Jitter. At last the object oriented programming language Java is presented. Java will be used for the communication handling with the robot and searching for packaging solutions.

3.1 Motoman IA20

The Motoman IA20 robot (see picture 3.1 on the next page) was released late in 2006 [7]. Back then it featured a revolutionary 7-axis actuator-driven design that enabled amazing freedom of movement [6]. It has a wide area of applications like for instance assembly, packaging, handling machine tending and part transfer. The standard robot controller used together with IA20 is the Motoman NX100 (see section 3.2 on page 18).

The reason for using Motoman IA20 and NX100 in this project was that this robot and robot controller was leased by the research group Robotics and Intelligent Systems (ROBIN) at the University of Oslo during 2008/2009 for other research projects.
3.1.1 Working Area & Physical Dimensions

This "snake robot" has the ability to maneuver in very tight areas. And with its space-saving design, it can straighten out vertically to take up only one square foot of floorspace or straighten out horizontally at a height of about two feet above the floor. All this together allows the manipulator to be positioned between machines or out of the normal working area in a floor-, ceiling or wall-mounting configuration. IA20’s physical dimensions is described in figure 3.2 on the next page. The vertical reach is as much as 159.8 cm. An detailed overview over its working area is described in figure 3.3 on the facing page.

3.1.2 Technical Facts

The IA20 robot has a 20 kg payload which should be suitable for most packaging installations. And its accuracy of repeatability of ±0.1 mm should be quite sufficient.

3.1.3 Package Gripper

In the experimental setups there wasn't a gripper available at the lab. However the IA20 has great possibilities for carrying different types of grippers, like for instance a vacuum claw. As you can see in figure 3.2 there are available air hoses and user wiring connectors that could be used to control an gripper.
Figure 3.2: Motoman IA20’s dimensions [6]

Figure 3.3: Motoman IA20’s working area [6]
3.2 Motoman NX100

The IA20 robot is controlled by the Motoman NX100 [6] robot controller that features a robust PC architecture, Windows CE programming pendant and IN-FORM III programming language. The PC architecture [8] offers high-speed processing, unmatched memory (60,000 steps, 10,000 instructions). It easily handles multiple tasks, and has an ability to control up to FOUR robots and I/O devices from a single point of control. It also has a built-in collision detection.

3.2.1 Communication Features

The available interfaces are RS-232C and Ethernet [9]. The latter one makes it possible to access the controller via HTTP, FTP and OPC. It also supports an optional web server that allows remote monitoring and diagnosis through the Internet or a local network. It has also Fieldbus support of the 15 most common brands on the market.
3.2.2 Additional Features

The NX100 achieves high-performance path accuracy and vibration control, using improved Advanced Robot Motion (ARM) control algorithms. Best-in-class path planning features dramatically reduce teaching time. The NX100 uses the enhanced INFORM II programming language [5].

3.2.3 Touch Screen Pendant

With the user friendly touch screen pendant (see figure 3.4 on the preceding page) running Windows CE operating system, one will have full control of the robot, with complete access to all operations from programming to maintenance. The 6.5" full colour LCD touch-screen display and the possibility to create one's own user menus eliminates the need for a separate operator's panel.

3.3 Cycling ’74 Max/MSP/Jitter

Max/MSP/Jitter is an interactive graphical programming environment mainly used for music, audio and media. Compatible with both Mac and Windows,
Max/MSP/Jitter provides true cross-platform development. A free runtime version and the possibility of create standalone applications makes it possible to distribute the software easily.

Max/MSP/Jitter is three things:

- Max is a graphical programming environment that provides user interface, timing, communications and MIDI support.
- MSP for real-time audio synthesis and DSP.
- Jitter for video and matrix data processing.

For the user these three elements appears as one unit, since all the programming and running the application is done within the same window(s). The direct sound-related parts of the software, such as MSP and MIDI support will not be used in this project (of natural causes).

### 3.3.1 Introduction to Max

Programming with Max is done in a graphical environment with components (built-in or self-made) that can be added and connected together. This way of programming instead of conventional programming languages makes Max extremely visual and interactive. A smooth learning curve makes it possible
for non-programmers to immediately start programming. For small projects
the development time is short. But as with conventional programming, more
complicated applications will take a lot longer.

A Max document, called a "patcher", can be thought of as a complete audio
or video program. But the programs need the Max runtime to run. Max is
also capable of building "stand-alone" applications, which can be executed on
computers that don't have Max installed. The application then includes the Max
kernel and the external objects used. Max operates as an interpreter, calling
the native code of external objects according to how they are connected and
used in the patcher. The objects are executed right-to-left, then bottom-to-top.
Connected objects are executed by a depth-first traversal. The "bang"-message,
a trigger signal, is essential in Max, and can be used to create automated or
self-running actions in the programs.

The objects that the user fits together to make a patcher are segments of
code that perform specific functions such as numerical calculations, manipu-
lating text, processing audio signals or video matrices. There are objects for
inputting and outputting for example video data (see the "jit.dx.grab"-object in
figure 3.6 on the facing page), so a patcher can operate as a complete appli-
cation. Many of the objects supplied in the Max library are designed as user-
interface components: Buttons, sliders, number boxes and so on. Connect a
set of objects together by lines or "patch cords", and one will have a real-time
processing graph, which might implement something as simple as a stopwatch
or a more complicated system.

You can make your own objects, often called externals, using JavaScript,
Java or C. JavaScript is the quickest way of making your own objects, and it can
be programmed from Max.

Max provides two distinct ways of organizing and viewing a patcher. In
Patcher Mode (see figure 3.7 on the next page) objects are assembled and con-

3.3.2 Introduction to Jitter

Jitter extends the Max/MSP with real-time manipulation of video and other data
sets. In Max/MSP/Jitter all signals are numbers, and since Jitter abstracts all
data as multidimensional matrices, it can most likely use the same objects for
Figure 3.7: Max/Jitter example: Patcher Mode

Figure 3.8: Max/Jitter example: Patcher Mode - Locked
processing image, audio, 3d vertices's or any numerical information you can get into the computer.

Jitter is tightly integrated with Max’s graphical programming environment which lets you visually connect data processing objects together with patchcords to create custom applications. This visual framework provides the power to build your own unique video effects or whatever your heart desires. In figure 3.7 on the facing page, objects like "jit.rgb2luma" and "jit.op" shows how easily video can be processed.

Jitter has a great interoperability since it let you easily grab data from video stream, stored movies or connected cameras (see the "jit.dx.grab"-object in figure 3.6 on page 20).

cv.jit - A Collection of Vision Objects

The cv.jit objects are a collection of Max/MSP/Jitter tools for computer vision applications. Its goal is to simplify tasks such as image segmentation, shape and gesture recognition, motion tracking, etc. As you see in figure 3.7 on the facing page, edge detection can easily be handed out by the object "cv.jit.binedge".

The collection is based on OpenCV, which is a cross-platform computer vision library originally developed by Intel [19]. It focuses mainly on real-time image processing. OpenCV is a open source software.
3.3.3 Robot Control

To my knowledge there are little or none research on using Max for robot control in industrial environments. But some research experiments have used Max in social interaction and performance related robotic installations [14].

In chapter 4 on page 29 Max's suitability for robotics will be discussed.

3.3.4 Pure Data - An Alternative to Max

Many of the features and ideas from Max are copied into other sound programming software. The Open Source project Pure Data [2] (or pd” is maybe the most prevalent. Since it is Open Source its free to use. The disadvantages or reasons for not using it in this project as I see it are:

- Lack of (good) documentation.

- The graphical interface is quite bad compared to Max 5.

- C, C++ or FORTRAN (which all are compiled languages) can be used to write your own objects. Support for a high-level scripting language like JavaScript would be desirable since it can reduce development time and does not have to be compiled.

- The video part doesn’t seem fully comparable to Jitter (hard to tell because of lack of documentation).

- Fewer object available online.

- Max has an great benefit with its watchpoints and debugging possibilities.

On the other hand it is free to use and it is available for Linux. And since it is Open Source you can study and modify if you want to. Maybe some day Max/MSP/Jitter, Pure Data and other tools will be interoperable, making it possible to share objects.

3.4 Java

Java is an object oriented programming language originally developed at Sun Microsystems [18]. Java applications are typically compiled to bytecode that can run on any Java Virtual Machine (JVM) regardless of computer architecture. Officially platforms are available for Microsoft Windows, Linux, Mac OS X and Solaris.
The syntax of Java is largely derived from C++, but has fewer low-level facilities. Unlike C++, Java was built almost exclusively as an object oriented language. All code is written inside a class (see listing 3.1) and everything is an object, with the exception of the intrinsic data types.

Graphical user interfaces can relatively easily be made using e.g. the Swing library.

```java
/*
 * Outputs "Hello, World!" to the terminal, and then exit.
 */

public class HelloWorld {
    public static void main (String[] args) {
        System.out.println("Hello, world!");
    }
}
```

Listing 3.1: HelloWorld example in Java

For a quick introduction to Java programming in Max, you can take a look at the built-in tutorials or the manual "Writing Max External in Java" [3].

### 3.4.1 Why Choose Java?

Java, as a programming language, lends itself nicely to rapid prototyping because it is an object oriented language which often promotes code reuse. One can take advantage of the features from the standard class libraries, and all the codes available on the Internet to quickly provide new and desired functionality.

Java also handles memory management for one with its built-in garbage collector, so one does not need to worry about memory allocation issues present in C. Java externals are cross-platform. This means that one only need to develop and maintain one single binary file for distribution. The same file can be used for both the Windows and OS X versions of Max. This saves a lot of programming and testing time.

On the other side, since Java is running on a Virtual machine machine, there are no official low-level facilities like USB and RS-232 available that works on all operating systems.

Another advantage is that Max provides a simple integrated development environment that allows one to interactively program and compile Java externals without ever leaving the Max patch.
3.4.2 Java Runtime in Max

Accessing the functionality of Java externals in a Max patcher is slightly different than accessing the functionality of a Max external written in C. C externals are loaded directly into the Max application environment at runtime, but Java externals are loaded into the Max environment and thereafter communicate with the Max environment through a proxy external object called \textit{mxj} [3].

\textit{mxj} is a Max external written in C that interacts with an instance of the Java Virtual Machine (JVM) within the Max environment, and provides the glue that allows calls from Max to be dispatched to one's Java code through the JVM and vice versa [3]. Only one JVM is ever created over the lifetime of your Max application and all \textit{mxj} instances share this single JVM instance to execute their given functionality. This is illustrated in figure 3.10.

3.4.3 Sun Java Real-Time System

In robotic systems there will often be a need of hard real-time. The Sun Java Real-Time System (Java RTS) is Sun's commercial implementation of the Real-Time Specification for Java [15]. Implementations of the RTSJ makes standard Java technology more deterministic and enable it to meet rigorous timing requirements for mission-critical real-time applications. At the time being Java
RTS is only available for Solaris operating system and some Linux distributions.

3.5 SOAP

SOAP, originally defined as Simple Object Access Protocol, is a protocol specification for exchanging structured information in the implementation of Web Services in computer networks [24]. It relies on Extensible Markup Language (XML) as its message format. It also usually relies on other Application Layer protocols like Remote Procedure Call (RPC) and HTTP for message negotiation and transmission.

The advantages for using SOAP in this experiment is that it is both platform and language independent. And using SOAP over HTTP allows easy communication through proxies and firewalls.

On the other side the main disadvantage may be that SOAP is built on top of the verbose XML format. This may result in that fact that SOAP (in some cases) can be considerably slower than competing middleware technologies such as CORBA [24].

3.5.1 Skeleton of a SOAP Message

The construction of a SOAP message is outlined in listing 3.2.

```xml
<?xml version="1.0"?>
<soap:Envelope
 xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
 soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Header>
    ...
  </soap:Header>
  <soap:Body>
    ...
    <soap:Fault>
      ...
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

Listing 3.2: Skeleton SOAP message
3.5.2 Sample SOAP Messages

Listing 3.3 shows an example of a SOAP message of type request, which sends the request for the share price of the IBM stock. The response to this request message is received in the form of a response message. This is illustrated in the listing 3.3.

```xml
<?xml version="1.0"?>
<soap:Envelope
    xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
    soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body
      xmlns:m="http://www.example.org/stock">
    <m:GetStockPrice>
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

Listing 3.3: Sample SOAP message: Request

```xml
<?xml version="1.0"?>
<soap:Envelope
    xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
    soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body
      xmlns:m="http://www.example.org/stock">
    <m:GetStockPriceResponse>
      <m:Price>34.5</m:Price>
    </m:GetStockPriceResponse>
  </soap:Body>
</soap:Envelope>
```

Listing 3.4: Sample SOAP message: Response
Chapter 4

Robot Control System

This chapter starts with an overview of the project and its goals. This is followed by a discussion of what development framework to select, platform interoperability and selection of hardware in terms of e.g. operational stability.

The control system software that have been developed will be presented and discussed. The vision system for detecting and measuring packages and packing algorithms will be presented more thoroughly in the following chapters.

4.1 System Overview

The goal of this project is to develop an experimental release of an autonomous and vendor independent robot-based palletizing system. It is necessary with an advanced system to be able to control the robot. This system could be divided into many parts, and the main parts are the control system, vision system for packet detection and run-time algorithms for generating pallet schemes. Each of these parts can again be split into several smaller parts. An abstract overview of the system is illustrated in figure 4.1 on the next page.

4.2 Moving Software from Specialized Hardware to a Standard Personal Computer

Traditionally, the control systems of robot installations have been developed to run completely or partially on an appropriate robot controller, a programmable logic controller (PLC) [26] or other types of specialized hardware. Such hardware can be expensive, difficult to replace, and often more complicated and time-consuming to program. The idea of this project is to move as much as possible of the control system off the modern computer.
Figure 4.1: System overview
The aims of this are:

- Reduce cost (cheaper hardware, and faster / easier programming).
- A more vendor independent system.
- Easier to obtain replacement parts.
- Easier to deploy new hardware and robots.
- Greater degree of reusable code.

4.2.1 Quality

By using standard modern hardware it will probably be far easier to replace any defective parts. This may increase the systems uptime. But if the electronic parts of such a system have worse quality, it is not inconceivable that these parts often must be replaced more frequently.

This may be a contradiction to the manufacturers of robot controllers or PLCs, which often add prestige to only select the best component and testing the product properly in relevant / extreme environments.

There are many terms and standards describing hardware quality. In systems like this the term \textit{Mean time to failure (MTTF)} which measures the average time between failure with the modeling assumption that the failed system is not repaired. [20].

The following things should be taken into account by the use of modern computers in an industrial setting:

- Hard disk drive - Failures may occur.
- Fan - The system often need continuous cooling
- Hardware quality / testing.

There are also on the industry-based issues that modern computers are designed for high uptime and placement in more extreme environments. Such computers can run standard operating systems, and often have the same opportunities for IO.
4.2.2 Needs for Real-Time

Specialized hardware and software designed for industrial applications often have built a good support for e.g. hard real-time, in contrast to many operating systems on modern computers that in the best case support soft real-time. Microsoft Windows XP is an example of an operating system that is considered as soft real-time compatible [10].

Cycling ’74 Max/MSP/Jitter is often referred to as a real-time system. Maybe the description could be run-time passed better, but Max is maybe soft real-time, since it never can have "stronger" real-time facilities than the underlying operating system. For processes that require high priority Max has something called overdrive, which allows for other (and often heavier) processes downgraded.

In this project, there are no processes that stand out with a clear need for hard real-time. But in projects where it is found to be necessary, it may be appropriate to place such tasks (e.g. critical sensor handling) in the software placed on the robot controller.

4.3 Selecting Development Framework

Manufacturers of modern industrial robots, which usually develops their own development frameworks for users to program their robots. In addition, they often also develop their own programming languages. All in all, this means one must have specific knowledge about the actual robot, robot controller and software. On the other hand, these things can make the robot’s potential be exploited to the full.

4.3.1 Microsoft Robotics Developer Studio

Microsoft Robotics Developer Studio (MRDS) is probably among the very latest released development tools for robot systems. MRDS has many advantages in a project like this since it is intended to function as a framework for all kinds of robots from all manufacturers. The widely used programming language C# can be used, in addition to several other programming languages. MRDS also contains libraries with relevant features like concurrency and synchronization (see section 2.1.3 on page 6), and also distribution of tasks on different devices and processors (see section 2.1.3 on page 7).

The development is done mainly by creating and programming web-services for the features one want to have. It is easy to imagine that such web-services often completely or partly can be reused later. In addition, it will be easy to
replace a Websters for example if one wish to use another robot or other type of vision system.

There are no weighty reasons why MRDS is not used in this project (apart from the experimentation of the pilot), since it has everything required of functionality. In the experimental pilot stage of the project I experienced problems with the stability of webcams in MRDS. The same was experienced by other people's research group. In addition, there was a solid expertise of the group and associated research project on the use of Cycling’74 Max, including robot control.

Another reason not to choose MRDS is that at the time being is only official support for the Windows platform.

4.3.2 Cycling ’74 Max and Jitter

Cycling’ 74 Max/Jitter (Max) is not designed to be a robot development tools. But Max has many interesting features such as:

- Graphical user interface.
- Reusable objects, with a wide range of objects and libraries available on the Internet.
- Seamless use of the IO (cameras, sensors, etc).
- Excellent opportunities for advanced image processing in a simple manner.
- Visualization facilities using OpenGL.
- High degree of interoperability - running on both Windows and Max OS X.
- Good opportunity for expanded functionality using JavaScript, Java and C.

The choice fell on Max mainly because of the growing competence of the research group and Robotics Intelligent Systems at the Department of Informatics, University of Oslo. It is also interesting to see whether Max can be used for such purposes or not.
4.4 Graphical User Interface

When programming in Max, all or some of the visual objects used when programming constitute the user interface. This project has not focused very much on the user interface, so the user interface the user sees is the same as the "programming code" with all the objects and the cables drawn between them. In this way it is easier to understand and analyze the system. The user interface should be improved using in the presentation mode if any final version should be developed.

The main window of the control system developed in this project can be seen in figure 4.2. The figure gives an overview of the whole system. Each subpatch is clickable if the user wants to adjust settings etc.

4.5 State Machine

A palletizing system has many tasks or processes that will be implemented for each package to be handled. To get an overview of all the processes and need
of concurrency it beam necessary to draw a flowchart (see figure 4.3 on the next page).

4.5.1 The States

The six conditions as outlined in the flow chart is implemented in Max. The result of this you can see in figure 4.4. The four red or conditional decision boxes that contains a YES / NO question are in two of the cases implemented using a delay object, but in the last two cases some synchronization patches had to be made.

![Figure 4.4: The state controller patch (patch: state_controller)](image)

4.5.2 Concurrency and Synchronization

Process synchronization refers to the idea that multiple processes are to join up or handshake at a certain point, so as to reach an agreement or commit to a certain sequence of action [27].

As mentioned, two sub-patchers were made to easily deal with synchronization (see figure 4.5 on page 37 and 4.5 on page 37). Their task is to ensure that the next state is not activated before both the two previous have finished.
Figure 4.3: Flowchart over the system
Figure 4.5: State synchronization (patch: semaphore)

Figure 4.6: State synchronization (patch: semaphore2)
Java Implementation

Since a synchronization was needed, a subpatch for such a propose had to be made. Its task should be to hold the status of the two ongoing states, and enable the next state when both are finished. The easiest solution I found was to make a Max external in Java:

```java
private boolean leftInput = false;
private boolean rightInput = false;

public void bang() {
    int inlet_num = getInlet();

    if (inlet_num == 0) { // leftBang
        leftInput = true;
    } else if (inlet_num == 1) { // rightBang
        rightInput = true;
    } else if (inlet_num == 2) { // allFalse
        leftInput = rightInput = false;
    }

    if (leftInput && rightInput) {
        leftInput = rightInput = false;
        outlet(0, "bang");
    }
}
```

Listing 4.1: Parts of the Semaphore external (for the entire class see listing A.17 on page 122)

The `bang()` method is called from Max every time a new value is submitted one of its inlets. The actual inlet where it received the bang message can tell whether a state has finished or the system should be reset. When both states have finished, a bang message is sent to Max through its outlet. This will activate the next state.

4.5.3 Global Data

All use of the send (s) and receive (s) objects to send data is not controlled by the state or states that are active. In many ways such data can be viewed as global data. One must therefore be very particular that only the intended states are able to read the sent values. Probably can an advanced solution be made (patcher or external) with the aim to buffer the data until a new state is activated, but it can have impact on the simplicity of the system.
This problem can be called data synchronization which refers to the idea of keeping multiple copies of a data set in coherence with one another, or to maintain data integrity [25].

4.6 Robot Communication

As mentioned earlier, there has been a desire to try to minimize the vendor and product-specific code. The main reason is that this code rarely can be reused, in addition, it is often more time-consuming to program. The main idea is to put as little code as possible at the robot controller, since this code is quite product specific.

4.6.1 Synchronization

In order to synchronize the robot control process on the computer with the robot controller, so one need a synchronization mechanism. With use of this synchronization mechanism the computer could tell the robot to move to the new position, in addition to knowing whether the robot has completed the movement or not.

For this purpose a (distributed) shared memory communication solution has been selected. Since the data memory for NX100 can easily be accessed via SOAP, and internally on the NX100 it can be read by using simple commands. A solution with a common semaphore have been chosen in this project.

This semaphore is active low, which means that if nothing happens with the robot, the shared variable holds the value 1. When new position variables are sent from the computer to the robot. Then the robot should move to the new position since the semaphore variable is given the value 0. The robot controller is continuously reading the semaphore whenever the robot is not moving to a new position. Once the value has been set to 0 it starts the movement of the robot to the new position. When the robot has finished the movement, the robot controller sets the semaphore variable equal to 1 again. The computer which is waiting for the robot to finish moving and therefore continuously reads the semaphore, will now know that the robot has finished.

How the semaphore synchronization is implemented on NX100 is shown in figure [4.7] on the next page.

Disadvantages

The probably most significant disadvantage with this implementation is that the computer continuously sends SOAP messages to request the value of the
Semaphore while it waits for the robot to finish. This has probably taken some amount of CPU usage at the robot controller, and thus, it is conceivable that the robot movement calculations which done by the robot is executed slower. The ideal solution would be if the SOAP server at the robot controller could be set up to first answer the SOAP message when the robot has stopped moving. It is uncertain whether this is possible on the NX100, and it is probably a complicated job if even possible.

4.6.2 Programming the Robot Controller

The goal is to minimize the required code for robot control. Since the Motoman robot controller NX100 and the IA20 robot can be controlled in an XYZ coordinate system, we must be able to send over X, Y and Z values. Moreover, it must also be possible to rotate the claw so that packages can be rotated if necessary. The last feature which is clearly necessary to control, is the status of the claw, in terms of whether it should be open or closed.

It is conceivable that it could be desirable with the ability to send and set data like e.g. the robot’s movement speed. Such extensions are no problem to add in the developed solution, since there are available variables at the robot controller that can be set via SOAP.
Pseudocode

```
1 set claw position horizontal
2 set semaphore high

3 loop forever:
   4 wait while semaphore is high

   5 copy new position and rotation from shared memory
   6 move to new position and rotation

   7 copy gripper status from shared memory
   8 open/ close gripper

   9 set semaphore high
```

Listing 4.2: Pseudocode for NX100 program

Listing 4.2 shows the pseudo code for the NX100 program. The first thing that happens is that the position variables that control the robot's head, is set to a horizontal position. Then the shared semaphore is given the value 1, which means that the robot is ready to receive data from the computer.

Then follows the eternal loop (see line 4 in listing 4.2) which is repeated until the program is stopped manually at the robot control. The loop is not executed as long as the semaphore is high, but starts processing the contents of the loop immediately when the semaphore has been set to active low.

What follows next in the loop is that the position and rotation values that were sent from the computer is copied from the shared memory area over to the robot controller's own memory area for position variables. The robot then moves to the new position. The gripper status is then copied from the shared memory area over to the robot controller's own memory area. The robot then either opens or closes the gripper.

At the end of the loop the semaphore value is set to 1. In this way, the computers know that the robot has completed the move.

Inform II Implementation

The final code developed for the Motoman NX100 robot controller is programmed in Inform II [5]. The source code can be found in figure 4.7 on the facing page and listing A.5 on page 97.
4.6.3 Communication Handler - A Max External

In order to communicate with the robot from Max, an external called *RobotCom* was made. Its task is to collect all relevant data from Max, put it into a SOAP message and then send it to the robot. Once the robot is finished with the movement, a *bang* message is sent to Max.

**Pseudocode**

```plaintext
initialize SOAP client

loop forever:
    while (last data not received from Max):
        receive data

        set semaphore active low

        pack data and semaphore into a SOAP-message
        send SOAP-message to robot

        pack get semaphore into a SOAP-message
        send SOAP-message
        receive and read SOAP-message with semaphore status

        while (while semaphore is active low):
            receive and read SOAP-message with semaphore status

        send finished to Max
```

Listing 4.3: Pseudocode for communication handler program

The first thing that happens in the pseudocode for the communication handler program (see listing 4.3) is that a SOAP client is initialized. This SOAP client class will contain information about the robot’s IP-address and port number, and a sketch of a SOAP message where the data can later can be put into.

Then follows an eternal loop. Inside this loop we can find an another loop. This loop receives various data from Max. When the last data has received, the loop stops. A local semaphore variable is created, and set to active low. Then all the data and the semaphore is put into the SOAP message. Then this message is sent to the robot.

Afterwards a SOAP request is sent to the robot in order to get the status of the shared semaphore to be able to determine whether the robot has completed
or not. This is repeated until a positive response is received. Finally, a message is sent to Max telling that the robot has completed.

**Java Implementation**

The implemented code differs slightly from the pseudocode in terms. The solution is not loop-based, but consists of a method of each type of data that can be received from Max. The actual method is called each time data is received. These methods have a unique Max-syntax [3].

The Java code for the RobotCom external can be found in listing A.1 on page 89. An standalone Java class for testing the communication without Max is to be found in listing A.2 on page 91. When the SOAP client was written I had no knowledge about the good built-in libraries for SOAP setup in Java, so much of the code could probably have been made easier and prettier.

Java classes developed:

- RobotCom - A Max external for handling the communication with the robot.
- RobotCommunication - Testing the functionality used from Max.
- SendDataToRobot - Generates a SOAP message with the position data. The data is sent to the robot.
- WaitRobotFinished - Waits for the robot to finish (the robot controller sets its semaphore variable low (= 0)).

### 4.7 Pick and Place Packages

Most articulated robots perform tasks by storing a series of positions in its memory, and then moving to them at various times in its program sequence. In order to be able to pick packages from the production line, and placing them on the pallet, we must define some positions so the robot can move to where it should. The following 3 positions (P1-P3) has to be defined in each setup:

- 0 cm above the production line, centered at the end (defined as P1).
- Safely position between the production line and pallet (defined as P2).
- 0 cm above the pallet, the lower right corner (defined as P3).
To be able to perform the necessary movements, one have to derive even more positions based upon position P1 to P3. These new positions needs data like package size (height, depth and width), package location on the production line, and placement on the pallet (XYZ).

- 10 cm above package at the production line (defined as PC1, derived from P1).
- 0 cm above package at the production line (defined as PC2, derived from P1).
- Safely position between production line and pallet (defined as PC3, equal to P2).
- 20 cm above package placement position at the pallet (defined as PC4, derived from P2).
- 0 cm above package placement position at the pallet (defined as PC5, derived from P2).

**Pseudocode**

From these positions one can define a pseudocode which can be found in listing 4.4. This code snippet provides a good overview of the instructions needed to pick up a package and thereafter placing it on a pallet at a specified place. All this must be done in a safe manner so that we avoid collision with other objects or persons nearby. The code is repeated every time a new package arrives at the production line.

```plaintext
1 Open gripper.
2 Move to PC1.
3 Move to PC2.
4 Close gripper.
5 Move to PC1.
6 Move to PC3.
7 Move to PC4.
8 Move to PC5.
9 Open gripper.
10 Move to PC4.
11 Move to PC3.
```

Listing 4.4: Pseudocode for robot movement

44
Max Implementation

The Max implementation is illustrated in figure 4.8 on the following page and 4.9. As possible to see in these images, the user can easily set the positions P1, P2 and P2. Each of the calculated positions PC1-PC5 has one or more of the class Max external PalletHandler, which collects, and then sends the received data to the robot.

Figure 4.9: Place package (patch: robot_place_package)
Figure 4.8: Pick package (patch: robot_pick_package)
4.7.1 A Test Setup

A test setup was set up in order to test the robot communication and practical use of the robot. Two tables were taken in use under testing, where one should imagine a production line, while the other would be a pallet where the packages were placed. A picture of the setup is shown in figure 4.10.

4.8 Summary

The control system is designed with the aim that it should be as vendor independent as possible when it comes to integrating a new robot, computer, etc. As much as possible of the required program code has been moved over on a standard modern computer, that can be replaced with any standard computer if necessary. At the robot controller, the required code for controlling the robot has been minimized to a minimum. The system is developed in Max, where
graphical objects easily can be reused, modified or replaced. The software is developed as abstract as possible. Any robot specific requirements for other input data, can be easily modified in the implemented Max external RobotCom.

The autonomic advantages of the system will be more visible when the tightly integrated vision system and automatic generation of palletizing schemes are presented in the two following chapters.

No precise standards for robotics has been proposed in this chapter. The solution to this challenge, I believe lies in reusable and replaceable services or objects, together with a universal and platform independent control system.

One thing that has not been emphasized in this chapter is the benefits from robot simulation. Simulation of a robot and its surroundings may ensure a time-saving development process, since you do not need access to the robot to test out new solutions and changes. Such simulations are possible to create in Max since it has built-in support for OpenGL. The disadvantage compared to more traditional and product specific development tools, is that such tool often have ready-made 3D models of a selection of robots and associated equipment. In Max you have to develop the simulation setup yourself.
Chapter 5

Vision System

This chapter begins with looking at the possibility of using two standard webcams instead of advanced cameras and/or sensors for detecting and measuring packages. This is followed by a study of Jitter’s simplicity and suitability when it comes to image processing and previewing. This is summarized in a discussion of advantages and disadvantages of the implemented solution.

The vision system has the following two missions:

1. Detect new packages at the production line.
2. Measure their width, height and depth.

In this chapter an attempt for developing an vision system for the earlier mentioned goals is presented. The system is developed using Max, Jitter and cv.jit - a collection of Max/MSP/Jitter tools for computer vision applications (see section 3.3.2 on page 23). The vision software is an integrated part of the control system presented earlier (see figure 4.2 on page 34). On the left hand side of the figure you can see the package detection part, and second and third from left one can find the measuring parts.

5.1 Traditional Solutions

As described in section 2.2 on page 8, vision systems for industrial systems often tends to be advanced and consist of hardware that may be both expensive and difficult to replace.

There are several solutions to this problem and the solution one choose will vary from one installation to another. The best solution perhaps, which is the fastest and safest in the production is to get the package size from e.g. a PLC. This PLC then needs to retrieve values about the package from a previous place
in the process. But this solution require industrial / specialized hardware, and that communication must operate at all times.

In an automated production, you can usually know the package’s dimensions from product range, barcode, sensors, or a predefined production order from the production machine. In some cases it may also know which production or assembly line the package comes from, as by a limited range of packages can tell us what type of package that has arrived.

One can find very advanced camera for sale, with built-in processor(s) that can calculate and give one all the values one wants, and then transfer them to the robot controller. But these cameras often cost a lot of money, and the integration between the camera and robot can offer some unfamiliar challenges each time which also cost money. Very few customers are interested in paying for an expensive camera solution if it is possible to retrieve information from another location without using a camera.

5.2 A Two Camera Solution

In this experiment two standard webcams, which one could find in a everyday electronic store will be used. The first camera is placed above the production line, while the second camera is placed in front of the of the production line. The first camera will detect new packages and measure the aspect ratio between the package’s depth and width, while the latter one will measure its height and width.

The presented solution is illustrated in figure 5.1 on the facing page. Placement of each of the cameras will be explained and discussed in the two following sections.

5.2.1 Camera 1

The first idea was to make this camera measuring the depth of the package, but the distance between the camera and the top of the package vary due to different package heights. Because of this it is not possible to measure absolute values. However, it is possible to measure the aspect ratio between the depth and width of the package. This aspect ratio will later on be used to calculate the depth in millimeters from the width. Since there was also a need to detect new packages, this camera received two applications.

As you can see in figure 5.1 on the next page, camera 1 is placed above the production line. It is located some to the right to avoid getting in the way of the robot picking up packages.
5.2.2 Camera 2

As one can see in figure 5.1, camera 2 is placed to the left of the production line. At the end of the production line, there is a "stopper" which stops the package (marked with a red square in the figure). This means that there will be a fixed distance between a package and the camera.

It will be possible to calculate the absolute values of the width and height, given the aspect ratio between a camera pixel and millimeters at the package front (see figure 5.2 on page 53 for input field). Once the actual width is measured, it is possible to derive the absolute depth using the aspect ratio between the depth and width.

For calculating the height, a second parameter is needed. Camera 2 will not be able to capture the whole package (because of the "stopper"), but only the upper part. The system needs to know the height difference between the production line and the top pixels of the camera in millimeters (see figure 5.2 on page 53 for input field).

The black plastic strips that one can see in the figure are there to ensure that only one package is visible from the camera's location at the same time, as well as to achieve a higher contrast between the package and its background.
5.2.3 Pseudocode

The following pseudocode gives an overview of the process:

```plaintext
loop forever:
    detect new package using camera 1
    wait for package to be centered underneath camera 1
    measure aspect ratio between depth and width using camera 1
    wait for package to stand still at the end of the production line
    measure width, height and placement using camera 2
    calculate depth from width using aspect ratio, width vs. depth
```

Listing 5.1: Pseudocode vision algorithm

This pseudocode is based upon the criteria presented and explained in the two previous sections.

5.3 Camera Control

For setup and configuration a camera control subpatch has been made (see figure 5.2 on the next page). The most essential task for this subpatch is to capture video data from two selected cameras. The capturing is done by the two subpatches (see figure 5.3 on page 54 for the subpatch "p grab"). A new video frame (called Matrix in Max) is sent every 30 ms. The "settings" button allows one to adjust any parameter available for the active camera (e.g., brightness and white balance).

The video stream is converted into luma, which represents the brightness in an image. Luma represents the achromatic image without any colour information. This is done because the detection will not take into account colours of the packages since one has no knowledge about their colours.

As you can see to the left; the thresholding level can be adjusted for each of the cameras. Upper right, one can adjust the camera height and the aspect ratio between a pixel and real millimeters on the package. The last one is typically handed out by measuring a package width a known width, and adjust the parameter to get the correct output.
Figure 5.2: Camera control window (patch: camera_control)
5.4 Detecting Packages

Detecting new packages is done by using motion detection. One way of doing this is by using optical flow (see figure 5.4 on the next page for an example of optical flow). The "cv.jit" collection includes an implementation of the Lucas-Kanade method, a two-frame differential method for optical flow [22]. With this implementation it is possible to obtain movement horizontally or vertically in a given direction. Obtaining a certain movement direction is done by the "jit.unpack" object (see figure 5.5 on the facing page). Thereafter "cv.jit.sum" is used to measure the level of movement. If it is higher than the trigger level, a new package is detected.

5.4.1 Known Problems

The detection algorithm has no knowledge of what it detects beyond that the movement is in the specified direction. This means that everything that may move within the camera's image area will be able to trigger a false package detection, as long as the movement occurs in the right direction. Selecting an appropriate threshold is therefore crucial to avoid false detections.
Figure 5.4: Demonstration of the principle of optical flow. Generated by Doug Hatfield (Wikipedia [22]).

Figure 5.5: Detect package (patch: vision_detect_package)
5.5 Measuring Packages

Measuring the size of a package can be a difficult task. A package should be significantly brighter or darker than its surroundings for the computer to be able to recognize it. The package itself often contains darker or brighter areas, making the contrast within a package higher than contrast between the package and its background. The system should work all the time, not just a set of test packages. Therefore it is difficult to obtain an adequate training set of images.

The subpatch for measuring the aspect ratio between depth and width of the package is measured by the first subpatch (see figure 5.6 on the next page), while the last one measures height, width and position, and calculates the depth (see figure 5.14 on page 63). There is a time delay in between these two in order to ensuring that the package stands still at the end of the production line.

5.5.1 Relative Depth and Width

In the subpatch "vision_calculate_depth" one can find a "switch" object. This object activates and stops the calculations to avoid unnecessary use of the CPU.

The first step in the image processing sequence is that the image is thresholded. The thresholding of the camera input is handed out by the object "jit.op" (one can see the result in figure 5.10 on page 60). It converts the video into ON and OFF-pixels based on the threshold value set in the subpatch "camera_control". The jit.op object offers a numerous of opportunities in terms of mathematical operations on (video) matrices.

The morphological "cv.jit.open" operation consists of performing eroded and dilated operations successively (see figure 5.7 on page 58 for an overview over morphological operations). The erode stage removes ON pixel noise, while the dilate stage returns the shape to a similar area. This operation will make small gaps larger. Hopefully this will give a nicer and more correct shape.

In the search of the item in the picture, all connected components, or blobs, will be numbered by its size. A package has often text or dark areas, giving many connected components. By first performing edge detection, the package gets more visible. A simple edge detection where ON-pixels stays ON if they have at least one OFF neighbor, is used (see figure 5.12 on page 61. It returns only edge pixels in the binary image. In the next step this is used to find and identify the connected components, and marks them with a unique value. Thereafter the largest one is selected. Then the largest connected component (only existing) is found again by "cv.jit.label @charmode 1". Now it is possible
Figure 5.6: Measure aspect ratio depth vs. width (patch: vision_measure_depth)
Figure 5.7: Results of morphological operations: a) Original image. b) Dilated image. c) Eroded image. d) Closed image. e) Opened image.
for "cv.jit.blobs.bounds" to find the relative depth and width from where the
bounding box is placed (see figure 5.13 on page 62 for the bounding box).

The result is visualized, and the bounding box is measured in a JavaScript
(source code can be found at A.18 on page 147). The result from this script is
sent into a number box and sent further to the next state.

5.5.2 Absolute Height, Depth, Width and Position

The procedure is the same as for the depth measuring, but with a few excep-
tions. The JavaScript is a little different since it calculates the package's height
by measuring the distance from the top of the camera and the top of the pack-
age. It also calculates the placement / position of the package (source code can
be found at A.19 on page 147).

Thereafter height (see figure 5.15 on page 63), width (figure 5.16), depth
(figure 5.17) and placement is converted into absolute values in millimeters
(figure 5.18).

The output of a measurement is sent to the control system, and will be
handled by the packaging algorithm. It is also visualized with number boxes,
making it easy to debug or adjust necessary parameters.

5.5.3 Failed Attempts

During the development stage, some experiments with cv.jit objects, which
make use of the Hough transform [28] for line detection, was tried out. The
algorithm was too unstable, so the experiment was laid dead. It is difficult to
Figure 5.9: Video input type: black & white (patch: large_view)

Figure 5.10: Video input type: threshold (patch: large_view)
Figure 5.11: Video input type: open (patch: large_view)

Figure 5.12: Video input type: binedge (patch: large_view)
Figure 5.13: Large preview of camera 1 (patch: large_view2)
Figure 5.14: Measure height and width (patch: vision_measure_width_height)

calc_height: Calculates the absolute height of the package

63
Figure 5.16: Calculate width (patch: calc_width)

Figure 5.17: Measure depth (patch: calc_depth)

Figure 5.18: Calculate placement (patch: calc_placement)
say anything directly about the causes. One possible explanation could be low resolution and image quality from the webcams.

5.5.4 Known Errors and Problems

The methods for measuring package sizes have been designed with the primary goal to demonstrate the opportunities that exist within modern image processing techniques and applications, like e.g. Jitter. It is doubtless that solution could have been developed with more advanced techniques, e.g. corner or edge detection or template matching. It would be natural to believe that smaller error margins could have been achieved with such techniques.

Camera Position Limitations

In the image area to the camera, it is only the front of the package that should be visible. Neither the sides nor the top surface of the package should be visible. This could result in the package is measured too wide or too tall. The cameras must therefore be centered in front of / above the package. The camera positions will determine how tall and deep packages that can be measured. This problem could (probably) be avoided if a more advanced algorithm had been implemented.

Problems with OpenCV

Now and then some error messages from OpenCV occurs. OpenCV is the underlying vision library which cv.jit is built on top of. The problem appears to be caused by a memory handling error (see figure 5.19). The system needs to be restarted afterwards.
5.6 Test Results

A test setup was set up in order to be able to test the accuracy of the vision system. The test setup can be seen in figure 5.20 on the next page.

The following items (see figure 5.21 on page 68 and 5.22 on page 68 for pictures) was used during the tests:

<table>
<thead>
<tr>
<th>Item#</th>
<th>Item</th>
<th>Height</th>
<th>Width</th>
<th>Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>White box, standing</td>
<td>144 mm</td>
<td>96 mm</td>
<td>62 mm</td>
</tr>
<tr>
<td>2</td>
<td>White box, lying</td>
<td>96 mm</td>
<td>144 mm</td>
<td>62 mm</td>
</tr>
<tr>
<td>3</td>
<td>Milk</td>
<td>180 mm</td>
<td>71 mm</td>
<td>71 mm</td>
</tr>
</tbody>
</table>

Table 5.1: Items used during tests of the vision system

<table>
<thead>
<tr>
<th>Item</th>
<th>Measured Depth/Width</th>
<th>Error Depth/Width</th>
<th>Measured Height</th>
<th>Error Height</th>
<th>Measured Width</th>
<th>Error Width</th>
<th>Measured Depth</th>
<th>Error Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0,66</td>
<td>-1,55 %</td>
<td>143,30 mm</td>
<td>-0,50 mm</td>
<td>95,94 mm</td>
<td>-0,06 mm</td>
<td>62,94 mm</td>
<td>0,94 mm</td>
</tr>
<tr>
<td>1</td>
<td>0,66</td>
<td>-2,15 %</td>
<td>143,50 mm</td>
<td>-0,50 mm</td>
<td>95,60 mm</td>
<td>-0,40 mm</td>
<td>63,10 mm</td>
<td>1,10 mm</td>
</tr>
<tr>
<td>1</td>
<td>0,66</td>
<td>-2,74 %</td>
<td>144,10 mm</td>
<td>0,10 mm</td>
<td>96,28 mm</td>
<td>0,28 mm</td>
<td>63,93 mm</td>
<td>1,93 mm</td>
</tr>
<tr>
<td>2</td>
<td>0,45</td>
<td>4,95 %</td>
<td>98,38 mm</td>
<td>2,38 mm</td>
<td>140,00 mm</td>
<td>-4,00 mm</td>
<td>63,42 mm</td>
<td>1,42 mm</td>
</tr>
<tr>
<td>2</td>
<td>0,45</td>
<td>4,74 %</td>
<td>98,38 mm</td>
<td>2,38 mm</td>
<td>140,00 mm</td>
<td>-4,00 mm</td>
<td>63,28 mm</td>
<td>1,28 mm</td>
</tr>
<tr>
<td>2</td>
<td>0,46</td>
<td>5,37 %</td>
<td>98,38 mm</td>
<td>2,38 mm</td>
<td>141,00 mm</td>
<td>-3,00 mm</td>
<td>64,16 mm</td>
<td>2,16 mm</td>
</tr>
<tr>
<td>3</td>
<td>0,96</td>
<td>-3,95 %</td>
<td>175,30 mm</td>
<td>-4,70 mm</td>
<td>77,29 mm</td>
<td>6,29 mm</td>
<td>74,35 mm</td>
<td>3,35 mm</td>
</tr>
<tr>
<td>3</td>
<td>0,96</td>
<td>-3,84 %</td>
<td>175,00 mm</td>
<td>-5,00 mm</td>
<td>77,20 mm</td>
<td>6,20 mm</td>
<td>74,34 mm</td>
<td>3,34 mm</td>
</tr>
<tr>
<td>3</td>
<td>0,97</td>
<td>-2,99 %</td>
<td>175,30 mm</td>
<td>-4,70 mm</td>
<td>77,63 mm</td>
<td>6,63 mm</td>
<td>75,38 mm</td>
<td>4,38 mm</td>
</tr>
<tr>
<td>3</td>
<td>0,97</td>
<td>-3,41 %</td>
<td>175,30 mm</td>
<td>-4,70 mm</td>
<td>77,29 mm</td>
<td>6,29 mm</td>
<td>74,74 mm</td>
<td>3,74 mm</td>
</tr>
</tbody>
</table>

Table 5.2: Test results of the vision system

The vision parameters was calibrated with the first item.

5.6.1 Results

The test result shows that the measuring is quite inaccurate. The large margin of error may, as said earlier, be due to poor algorithm and low low-resolution camera (see the poor image quality in e.g. figure 5.13 on page 62).

It is also possible that in some of the calculations made in cv.jit objects or the underlying OpenCV libraries, uses matrices that are smaller than the camera resolution. In some of the visualization objects in cv.jit, only a 160x120 resolution is used.
Figure 5.20: A test setup of the vision system
Figure 5.21: White box used during tests of the vision system

Figure 5.22: Milk used during tests of the vision system
5.7 Summary and Discussion

In this chapter have been presented a (possibly new) solution to detect and measure packages on a production line, with use of two standard webcams. This solution makes the system more autonomous since it now can detect and measure a wide range of package types.

Max and Jitter turns out to have infinite possibilities in video processing in an easy manner. There are none ready-made vision solutions in Jitter. This is something you can find in advanced development tools for machine vision. A direct comparison with such software could be interesting.

The results of the test setup shows that the implemented vision system does not provide desirable degree of accuracy. There is no clear indicia what margin of error is due. It is assumed that cameras of better quality combined with better algorithms, could get the margin of error down to an acceptable level.

Screenshots of all the different patches of the vision system can be found in the appendix at the end of this thesis.
Chapter 6

Packing Algorithms

As described in section 2.3 on page 10, packing algorithms is a wide area of research. In this chapter some algorithms for the Manufacturer’s Pallet Loading Problem (MPLP) is presented, discussed and implemented. The implementation also contains run-time generating of packing schemes. This is a contradiction to traditional systems where such schemes in most cases are made in advance. Since the MPLP can be thought of as a one layer problem (if all layers can be equal), the algorithm only has to generate a packing scheme for one layer.

A Max external named PalletHandler has been made, and will be presented in this chapter. PalletHandler handles the run-time packing algorithm. The pallet schemes are visualized in two dimension, while the pallets are visualized in three dimensions in run-time. This latter visualization shows where each package is to be placed.

It is assumed that all the packages on the pallet are identical, and the algorithm therefore only has to run once for every pallet. This equals to the MPLP. The biggest challenge with the run-time generation of packing schemes is the short amount of time that is disposal to the search algorithm. This means that the algorithm must be quite fast. The first package arriving at the production line must be picked up before the next package arrives. In the same period the algorithm must also have produced a good or optimal scheme, so that the package can be placed on the pallet, to avoid packages accumulating on the production line.

6.1 Fitness of Packing Algorithms

To compare different packing algorithms and heuristics, we need a basis for comparison. It can be done in two ways. The first one is to count and compare the number of packages the algorithms manage to fit onto the pallet. The
second alternative is to calculate the percentage of covered area of the pallet. It can be done with the following fitness formula (if all the packages are equal):

\[
\text{Fitness} = \frac{\text{PackageAreal} \times \text{NumberOfPackages}}{\text{PalletAreal}}
\]

### 6.2 Overview of Java Classes

The following Java classes have developed working with packing algorithms:

- **PalletHandler** - A Max external for generating and handling pallet schemes.
- **PalletOrganizer** - Testing the functionality used from Max.
- **PalletSearch** - Search for different pallet schemes.
- **PalletConfig** - An abstract class for holding information of a pallet and its packages.
- **PalletConfigRowRotate** (extends PalletConfig) - Class for holding information of a pallet and its packages. A given number of rows are rotated.
- **PalletConfigEqualRotate** (extends PalletConfig) - Class for holding information of a pallet and its packages. All packages are either rotated or not rotated.
- **PalletConfigBlockHeuristics** (extends PalletConfig) - Class for holding information of a pallet and its packages. 4 blocks of various size each handled as an individual pallet.
- **Pallet** - Holding information of a pallet and its packages. Used for run-time palletizing.
- **Package** - Holding information of a single package. Used for run-time palletizing.
- **DrawPallet** - Visualize a pallet scheme.
- **PalletLoadingTest** - Main class. Generates different tests.

The most important of these classes will be reviewed in this chapter.
6.2.1 PalletConfig - Searching after Packing Schemes

The different search algorithms need a data structure to work on. Since Java is an object-oriented language, it felt natural to use a class structure. PalletConfig is an abstract class with the common methods needed by the algorithms. Here follows a simplified version:

```java
public abstract class PalletConfig {
    // private variables

    PalletConfig(int palletWidth, int palletDepth, int palletHeight, 
                int packageWidth, int packageDepth, int packageHeight) {
        ... // set variables
    }

    public int getPalletWidth() { return palletWidth; }

    public int getPalletDepth() { return palletDepth; }

    public int getPackageWidth() { return packageWidth; }

    public int getPackageDepth() { return packageDepth; }

    abstract public void rotate(int i);

    abstract public void rotateAll();

    public float getFitness() {
        return (float) (getPackageDepth() 
                * getPackageWidth() * getPackagesPlaced()) 
                / (getPalletDepth() * getPalletWidth());
    }

    abstract public int getPackagesPlaced();

    abstract public PalletConfig clone();

    abstract public Pallet convertToPalletFormat();
}
```

Listing 6.1: PalletConfig - An abstract pallet class for searching algorithms

The convertToPalletFormat() method converts the current object into an object of the Pallet class. The Pallet class is a run-time palletizing class. It contains information of where all the packages should be placed. This makes it possible
Figure 6.1: Packing scheme - Next Fit

Figure 6.2: Packing scheme - Next Fit with all packages rotated

to switch packing algorithm easily, if only the new algorithm outputs the result in this format.

6.3 Next Fit - A First Attempt

The Next fit algorithm is a simple packing algorithm (see section 2.3.4 on page 12 about Next Fit). An expanded variant of this algorithm was implemented as a first attempt to generate packing schemes. The expansion consists of the possibility of rotating all packages (see figures 6.1 and 6.2). A fitness calculation determines whether the packages should be rotated or not.
6.3.1 Pseudocode

The following pseudocode explains how the new packing schemes are generated:

```java
boolean rotate = false
if fitness of rotated scheme > fitness of standard scheme:
    rotate = true
loop forever:
    if rotate:
        rotate package
    if enough space on current row:
        place package on current row
    else:
        place package on a new row
```

Listing 6.2: Pseudocode packing algorithm - Next Fit

6.3.2 Java Implementations

The algorithm is implemented in the PalletConfigEqualRotate class (see listing A.11 on page 106), which is an extension of the PalletConfig class.

The following code is necessary to determine whether the packets should be rotated or not:

```java
public PalletConfig getBest() {
    PalletConfig reversePallet = this.clone();
    if (isRotated())
        reversePallet.rotate(0);
    else
        reversePallet.rotateAll();
    if (getFitness() > reversePallet.getFitness())
        return this;
    else
        return reversePallet;
}
```

Listing 6.3: Next Fit - Find best solution
The necessary code snippet for calculating the fitness is straight forward:

```java
public int getPacakagesPlaced() {
    int sum;

    if (rotated)
        sum = (palletWidth / packageDepth) * (palletDepth / packageWidth);
    else
        sum = (palletWidth / packageWidth) * (palletDepth / packageDepth);

    if (sum < 0)
        return 0;
    else
        return sum;
}
```

Listing 6.4: Next Fit - getPacakagesPlaced()

### 6.3.3 Discussion

In some cases this algorithm generates a acceptable packing schemes, but not all. The algorithm has a few simple calculations, which makes the algorithm very fast.

### 6.4 Next Fit With Row Rotation

With the aim to improve the packing results from the previous algorithm, a second modification of my own was added; a variable number of rows can be rotated. The number of rotated rows will in most cases provide different fitness results.

#### 6.4.1 Java Implementation

The algorithm is implemented in the PalletConfigRowRotate class (see listing A.10 on page 104), which also extends the PalletConfig class.
The following code is necessary to determine whether the packets should be rotated or not:

```java
public PalletConfig getBest() {
    PalletConfig bestPallet = this.clone();
    PalletConfig currPallet = this.clone();

    for (int i = 0; i <= getMaxRows(); i++) {
        currPallet.rotate(i);
        if (currPallet.getFitness() > bestPallet.getFitness())
            bestPallet = currPallet.clone();
    }

    return bestPallet;
}
```

Listing 6.5: Find the best row rotation
Calculating the number of packages that will fit onto the pallet, is slightly more advanced with optional number of rotated rows:

```java
public int getPacakgesPlaced() {
    int currPosY = 0;
    int packagesPlaced = 0;
    int i = 0;

    // Rotated rows
    for (; i < rowsRotated; i++) {
        if ((currPosY + packageWidth) > palletDepth)
            break;

        packagesPlaced += palletWidth / packageDepth;
        currPosY += packageWidth;
    }

    // None-rotated rows
    for (; i < getMaxRows(); i++) {
        if ((currPosY + packageDepth) > palletDepth)
            break;

        packagesPlaced += palletWidth / packageWidth;
        currPosY += packageDepth;
    }

    return packagesPlaced;
}
```

Listing 6.6: Next Fit - getPacakgesPlaced()

### 6.4.2 Discussion

This algorithm generates better packing schemes than the previous algorithm. The necessary calculations are still few and simple, which makes the algorithm fast.

### 6.5 A Block Algorithm

The two preceding algorithms can, in many cases, not find an optimal packing scheme. So I started looking at the possibility of the dividing the pallet into sec-
tions. The choice fell on the four sectors (see figure 6.4). After some research it turned out that this is a fairly common way to solve packing problems. The number of sectors and their shape varies from one algorithm to another.

The idea is that the packages should have an equal rotation inside each of the four sectors. The algorithm task then becomes to find the optimal distribution of the sizes of the sectors. In order to reduce the otherwise huge search space, it is necessary with some interceptions. Therefore, sectors can only have sizes that are a multiple of the package’s width or depth.

![Figure 6.4: Packing scheme - 4 block algorithm](image)

### 6.5.1 Pseudocode

Figure 6.5 on the following page illustrates the four sections of the pallet. The size of the four blocks can be set by the following algorithm:

```plaintext
1 choose a within the pallet’s width
2 choose c within a’s width
3 choose d within the pallet’s depth
4 choose h within d’s depth
b, e, f and g are given the remaining space
```

Listing 6.7: Pseudocode packing algorithm - A 4 block algorithm

In most cases, it is possible to search through all possible solutions to find the optimal solution (often called brute force). The size of the search space depends on the package size in relation to the pallet size. Larger difference provides a larger search space.

Another strategy is to try out random sizes, and select the best result is achieved. With a high number of attempts, will be possible to achieve accept-
able results. The algorithm takes the same amount of time regardless of the size of the search room, but the chance to find an optimal solution is reduced.

6.5.2 Discussion

This algorithm generates better packing schemes than the previous algorithm. The necessary calculations are still few and simple, which makes the algorithm fast.

This algorithm usually produces good or optimal solutions, but it will not always manage to find the optimal solutions in the most difficult cases.

6.6 Test Results

The test data used by Pureza et al. [17] is used to compare the implemented algorithms and Pureza et al.’s algorithm.
<table>
<thead>
<tr>
<th>Pallet Size</th>
<th>Package Size</th>
<th># Packages Placed</th>
<th>Next Fit</th>
<th>Next Row Fit</th>
<th>Block Algorithm</th>
<th>Pureza et al. algorithm [17]</th>
<th>Max. Execution Time, Block Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>First order problems</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(22, 16)</td>
<td>(5, 3)</td>
<td>21</td>
<td>22</td>
<td>22</td>
<td>23*</td>
<td>16 ms</td>
<td></td>
</tr>
<tr>
<td>(57, 44)</td>
<td>(12, 5)</td>
<td>33</td>
<td>38</td>
<td>40</td>
<td>41*</td>
<td>16 ms</td>
<td></td>
</tr>
<tr>
<td>(86, 82)</td>
<td>(15, 11)</td>
<td>35</td>
<td>38</td>
<td>41</td>
<td>42*</td>
<td>15 ms</td>
<td></td>
</tr>
<tr>
<td>(42, 39)</td>
<td>(9, 4)</td>
<td>40</td>
<td>42</td>
<td>44</td>
<td>45*</td>
<td>16 ms</td>
<td></td>
</tr>
<tr>
<td>(124, 81)</td>
<td>(21, 10)</td>
<td>40</td>
<td>42</td>
<td>46</td>
<td>47*</td>
<td>16 ms</td>
<td></td>
</tr>
<tr>
<td>(40, 25)</td>
<td>(7, 3)</td>
<td>40</td>
<td>44</td>
<td>46</td>
<td>47*</td>
<td>16 ms</td>
<td></td>
</tr>
<tr>
<td>(52, 33)</td>
<td>(9, 4)</td>
<td>40</td>
<td>44</td>
<td>46</td>
<td>47*</td>
<td>16 ms</td>
<td></td>
</tr>
<tr>
<td>(56, 52)</td>
<td>(12, 5)</td>
<td>44</td>
<td>45</td>
<td>47</td>
<td>48*</td>
<td>15 ms</td>
<td></td>
</tr>
<tr>
<td>(87, 47)</td>
<td>(7, 6)</td>
<td>84</td>
<td>94</td>
<td>95</td>
<td>97*</td>
<td>16 ms</td>
<td></td>
</tr>
<tr>
<td>(67, 44)</td>
<td>(6, 5)</td>
<td>91</td>
<td>96</td>
<td>97</td>
<td>97*</td>
<td>16 ms</td>
<td></td>
</tr>
<tr>
<td>Superior order problems</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(43, 26)</td>
<td>(7, 3)</td>
<td>48</td>
<td>52</td>
<td>52</td>
<td>53*</td>
<td>16 ms</td>
<td></td>
</tr>
<tr>
<td>(63, 44)</td>
<td>(8, 5)</td>
<td>60</td>
<td>64</td>
<td>68</td>
<td>69*</td>
<td>16 ms</td>
<td></td>
</tr>
<tr>
<td>(61, 35)</td>
<td>(10, 3)</td>
<td>66</td>
<td>70</td>
<td>70</td>
<td>71*</td>
<td>16 ms</td>
<td></td>
</tr>
<tr>
<td>(61, 38)</td>
<td>(6, 5)</td>
<td>72</td>
<td>76</td>
<td>76</td>
<td>77*</td>
<td>16 ms</td>
<td></td>
</tr>
<tr>
<td>(61, 38)</td>
<td>(10, 3)</td>
<td>72</td>
<td>76</td>
<td>76</td>
<td>77*</td>
<td>16 ms</td>
<td></td>
</tr>
<tr>
<td>(93, 46)</td>
<td>(13, 4)</td>
<td>77</td>
<td>81</td>
<td>81</td>
<td>82*</td>
<td>16 ms</td>
<td></td>
</tr>
<tr>
<td>(141, 71)</td>
<td>(13, 8)</td>
<td>85</td>
<td>91</td>
<td>95</td>
<td>96*</td>
<td>16 ms</td>
<td></td>
</tr>
<tr>
<td>(106, 59)</td>
<td>(13, 5)</td>
<td>88</td>
<td>95</td>
<td>95</td>
<td>95</td>
<td>16 ms</td>
<td></td>
</tr>
<tr>
<td>(74, 46)</td>
<td>(7, 5)</td>
<td>90</td>
<td>92</td>
<td>96</td>
<td>96</td>
<td>16 ms</td>
<td></td>
</tr>
<tr>
<td>(86, 52)</td>
<td>(9, 5)</td>
<td>90</td>
<td>96</td>
<td>98</td>
<td>98</td>
<td>16 ms</td>
<td></td>
</tr>
<tr>
<td>(108, 65)</td>
<td>(10, 7)</td>
<td>90</td>
<td>95</td>
<td>99</td>
<td>99</td>
<td>16 ms</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.1: Test results of the packing algorithms

The results marked with * is the optimal number of packages.

One can clearly see the improvements from eg. the Next Fit algorithm to the block algorithm. The block algorithm produces quite good results (keeping in mind that this is a difficult dataset). It also proves to be quick.

### 6.7 PalletHandler - A Max External

All the algorithms was developed in Java, and thereby an interface to them from Max was needed. A Max external called PalletHandler was made (the Java code can be found in the listing A.6 on page 98). Its task is to generate new packing schemes, and tell where each packet should be placed. For each packet that the vision system detects, placement coordinates for the package at pallet is sent to Max. These coordinates are used to calculate the robots movements.

### 6.8 Visualization

Visualization of both the packing scheme and the pallet is needed to be able to easily analyze the performance of the algorithms.
Figure 6.6: Packing algorithm in Max (patch: package_find_placement)
6.8.1 Pallet Scheme

The packing schemes is visualized using the Swing library in Java. The DrawPallet class (see listing A.15 on page 116) draws the pallet and its packages.

6.8.2 3D View of the Pallet

Max has great support for the use of OpenGL. OpenGL was used to visualize the pallet and its packages in three dimensions (see figure 6.8 on the next page and 6.8 on the following page). The implemented solution supports only 100 packages, but it should be possible to extend if needed.

6.9 Summary

Three relatively simple algorithms were implemented in Java. The most advanced of them, a block algorithm, is currently used when generating new packing schemes. The algorithm is robust and relatively quick.

An external Max has been developed to take care of the whole process from generating packing schemes to specify where each and every package is to be placed on the pallet.

A two and three dimensional visualization of the packing schemes and the pallet, is also implemented.
Figure 6.8: 3D view of a pallet with one package

Figure 6.9: 3D view of a pallet with many packages
Chapter 7

Conclusions

A control system for the robot is implemented in Max, with some externals written in JavaScript and Java. The vision system is tightly integrated into the control system, and uses 2 standard webcams to detect and measure package sizes. Finally, runtime generation of palletizing schemes functionality is also added, with both 2D and 3D visualization.

7.1 Robot Control System

The control system is designed with the aim that it should be as vendor independent as possible when it comes to integrating a new robot, computer, etc. As much as possible of the required program code has been moved over on a standard computer, that can be replaced with any standard computer if necessary. At the robot controller, the required code for controlling the robot has been minimized to a minimum.

7.1.1 Max for Robot Control

Max is a very flexible development framework, but the target group does not contain robotic systems. Therefore one has to experiment on one’s own to work out solutions. The needed functionality can mostly be found in Max’s wide range of objects. When needs are not be met in Max, the possibility to write externals JavaScript, Java or C is good to have.

7.1.2 Standardization

No precise standards for robotics have been proposed in this chapter. The solution to this challenge, I believe lies in reusable and replaceable services
or objects, together with a universal and platform independent control system placed on a standard computer.

7.2 Vision System

A (possibly new) solution based on two webcams is presented. These webcams are used to detect and measure packages. The implementation in Max and Jitter shows that there are plenty of opportunities when it comes to video processing. The disadvantage is however that there are none ready-made vision solutions in Jitter.

The results of the test setup shows that the implemented vision system does not provide desirable degree of accuracy. The results of the test setup shows that the implemented vision system does not provide desirable degree of accuracy. It is assumed that cameras of better quality combined with better algorithms, could get the margin of error down to an acceptable level.

7.3 Packing Algorithms

Three relatively simple algorithms were implemented using Java. The most advanced of them, a block algorithm, is currently used when generating new packing schemes. A two dimensional visualization of the packing schemes and a three dimensional visualization of the pallet, is also implemented.
Chapter 8

Further Work

Things take time. This project, covers large research areas such as robotics, code re-usage, interoperability, vision systems and packaging algorithms. It is clearly many things that could be studied further both in terms of a wider scope and more thorough.

8.1 Robot Control System

It would be desirable to create a collection of Max objects with the functionality needed in a robot control system. A simulation framework for robot cells in OpenGL might also be useful.

A comparison between the development methodology, limitations, code re-usage e.g. between Max, Microsoft Robotics Developer Studio and other development tools could be useful.

8.2 Vision System

More advanced packet recognition algorithms should be implemented in order to achieve better results in the form of lower error margin. It could also be interesting to compare Max and Jitter with machine vision frameworks from e.g. NI LabVIEW.

8.3 Packing Algorithms

Optimal packing requires advanced algorithms. Run-time packing also needs fast algorithms. It could therefore be useful to do a deeper study of algorithms
suitable for run-time palletizing.

Other factors such as the packages weight or the stability of pallet, may be important in some cases, and should have been looked into.
Appendix A

Source Code

This chapter contains all the source code and Max patchers developed during this thesis.

A.1 Robot Communication Handler

Listing A.1: RobotCom.java

```java
import com.cycling74.max.*;

/* *
* @description A communication handler for the Motoman NX100 Robot Controller
* (SOAP).
* @class RobotCom - A Max object for handling the communication with the robot.
* @author Sigurd Salvesen, sigurds@ifi.uio.no.
* @info Some small parts of the code is made by Mats Hovin.
*/

public class RobotCom extends MaxObject {
    SendDataToRobot sendData = null;
    WaitRobotFinished checkFinished = null;
    private boolean robotEnabled = false; // Default: robot disconnected
    private int xAxis = 0;
    private int yAxis = 0;
    private int zAxis = 0;
    private int rotation = 0;
    private int claw = 0;
    private static final String[] INLET_ASSIST = new String[] { "inlet 1 help" };
    private static final String[] OUTLET_ASSIST = new String[] { "outlet 1 help" };

    public RobotCom(boolean bang, boolean reset, int x, int y, int z, int rotation, int claw, int robot_enabled, String robot_adress) {
        declareInlets(new int[] { DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.A
setInletAssist(new String[] { "bang to output", "bang to reset", "x", "y", "z", "rotation", "claw", "robot_enabled", "robot_adress (URL)" });
setOutletAssist(new String[] { "finished" });

try {
    checkFinished = new WaitRobotFinished(); // Get data (semaphore)
} catch (Exception e) {
    post("RobotCom: unhandled error - nxClientGet0 = new NxClientGet0();");
    post("e: " + e);
}

public void bang() {
    int inlet_num = getInlet();
    post("RobotCom: banged at: " + inlet_num);
    if (inlet_num == 0) { // bang
        post("RobotCom: bang - send data to robot");
        if (robotEnabled) {
            post("RobotCom: Robot connected.");
        } else {
            post("RobotCom: Robot disconnected.");
        }
    } else if (inlet_num == 1) { // reset
        post("RobotCom: reset");
    }
}

if (inlet_num == 0) { // bang
    post("RobotCom: Sending data to robot (bang received from Max).";
    post("RobotCom: Data - x: " + xAxis + " | y: " + yAxis + " | z: "
           + zAxis + " | rotation: " + rotation + " | claw: " + claw
           + " | robotEnabled: " + robotEnabled);
    try {
        sendData.put(xAxis, yAxis, zAxis, rotation, claw);
    } catch (Exception e) {
        post("RobotCom: unhandled error - nxClientPutArray.put()");
        post("e: " + e);
    }
    post("RobotCom: Waiting for robot to finish.");
}

if (robotEnabled) {
    try {
        checkFinished.waiting();
    } catch (Exception e) {
        post("RobotCom: unhandled error - nxClientGet0.waiting();");
        post("e: " + e);
    }
    post("RobotCom: Robot finished");
    outlet(0, "bang"); // Send finished to Max
}
else if (inlet_num == 1) { // reset
    post("RobotCom: reset");
}
Listing A.1: RobotCom.java

```java
public void inlet(int i) {
    int inlet_num = getInlet();

    if (inlet_num == 2) { // x
        xAxis = i * 1000;
    } else if (inlet_num == 3) { // y
        yAxis = i * 1000;
    } else if (inlet_num == 4) { // z
        zAxis = i * 1000;
    } else if (inlet_num == 5) { // rotation
        rotation = i * 100;
    } else if (inlet_num == 6) { // claw
        claw = i;
    } else if (inlet_num == 7) { // robot enabled (robot connected?)
        if (i == 0)
            robotEnabled = false;
        else
            robotEnabled = true;
    }
}

public void inlet(String s, Atom[] list) {
    int inlet_num = getInlet();

    if (inlet_num == 8) { // robot adress
        if (s != null) { // list.isString()
            sendData = new SendDataToRobot(s);
        }
    }
```

Listing A.2: RobotCommunication.java

```java
import com.cycling74.max.*;

/**
 * @description A communication handler for the Motoman NX100 Robot Controller
 * (SOAP).
 * @class RobotCommunication – Testing the functionality used from Max.
 * @authors Mats Hovin (matsh@ifi.uio.no), Sigurd Salvesen, sigurds@ifi.uio.no.
 * @info Originally code by Mats Hovin. Rewritten & modified by Sigurd Salvesen.
 */

public class RobotCommunication {
    static SendDataToRobot sendData = null;
    static WaitRobotFinished checkFinished = null;
    static boolean robotEnabled;

    public static void main(String[] args) throws Exception {
        if (args.length >= 1) {
            System.out.println("Robot disconnected.");
```
robotEnabled = false;
}
else {
    System.out.println("Robot connected.");
    robotEnabled = true;
    // generating SOAP-messages
    sendData = new SendDataToRobot(); // Send data
    checkFinished = new WaitRobotFinished(); // Get data (semaphore)
}

run();
}

Listing A.2: RobotCommunication.java

Listing A.3: SendDataToRobot.java

import com.cycling74.max.*;
import java.io.*;
import java.net.*;

/**
 * @description A communication handler for the Motoman NX100 Robot Controller (SOAP).
 * @class SendDataToRobot -- Generates a SOAP message with the position data. The data is sent to the robot.
 * @authors Mats Hovin (matsh@ifi.uio.no), Sigurd Salvesen, sigurds@ifi.uio.no.
 * @info Originally code by Mats Hovin. Rewritten & modified by Sigurd Salvesen.
 */
public class SendDataToRobot {
  private String nxUrlString = "http://192.168.255.1:8080";
  private String[] part = new String[12]; // SOAP-message elements

  SendDataToRobot() {
    init();
  }

  SendDataToRobot(String nxUrlString) {
    this.nxUrlString = nxUrlString;
    init();
  }

  private void init() {
    // Send data message:
    part[1] = makeSoapElement(10);
    + "<q1:NxPutVarData xmlns:q1="urn:CoreService">
      + <sData href="#id1" /></q1:NxPutVarData>
    + "<soapenc:Array id="#id1" xmlns:q2="urn:CoreService">
      + soapenc:arrayType="VAR-DATA[10]">
        + "<Item href="#id2" />
        + "<Item href="#id3" />
        + "<Item href="#id4" />
        + "<Item href="#id5" />
        + "<Item href="#id6" />
        + "<Item href="#id7" />
        + "<Item href="#id8" />
        + "<Item href="#id9" />
        + "<Item href="#id10" />
    </soapenc:Array>";
  }

  // Generate SOAP-message elements
  private String makeSoapElement(int i) {
    int id = i + 1;
    int q = i + 2;
    String start = "";
    if (i != 1) start = "</ulValue></q" + Integer.toString(q - 1) + "::VAR-DATA";  
    // Semaphore-element
    if (i == 10) i = 0;
    return start + "<q" + Integer.toString(q) + "::VAR-DATA id="#id" + Integer.toString(id) + " xsi:type="q" + Integer.toString(q) + "::VAR-DATA" xmlns:q" + Integer.toString(q) + "::VAR-DATA\" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xs="http://www.w3.org/2001/XMLSchema-instance">" + "<usType xsi:type="xsd:unsignedShort">3</usType>" + "<usIndex xsi:type="xsd:unsignedShort">3";
public void put(int xAxis, int yAxis, int zAxis, int rotation, int claw) throws Exception {
    // index 0: Handshake
    // index 1: xAxis
    // index 2: yAxis
    // index 3: zAxis
    // index 4: Rotation
    // index 5: Claw
    // .... : unused

    String unusedVariable = "0";
    String semaphore = "0"; // Semaphore at the NX100

    // Putting the SOAP-message together

    byte[] byArr = soap.getBytes();
    String SOAPAction = "";
    URL url = new URL(nxUrlString);

    // Initialize connection
    URLConnection connection = url.openConnection();
    HttpURLConnection httpConn = (HttpURLConnection) connection;
    httpConn.setRequestProperty("Content-Length", String.valueOf(byArr.length));
    httpConn.setRequestProperty("Content-Type", "text/xml; charset=utf-8");
    httpConn.setRequestProperty("SOAPAction", SOAPAction);
    httpConn.setRequestMethod("POST");
    httpConn.setDoOutput(true);
    httpConn.setDoInput(true);

    // Send data
    OutputStream out = httpConn.getOutputStream();
    out.write(byArr);
    out.close();

    // Read data
    InputStreamReader isr = new InputStreamReader(httpConn.getInputStream());
    BufferedReader in = new BufferedReader(isr);
    in.close();
}
import java.net.*;

/**
 * @description A communication handler for the Motoman NX100 Robot Controller (SOAP).
 * @class WaitRobotFinished - Waits for the robot to finish (the robot controller sets its semaphore variable low (=0)).
 * @authors Mats Hovin (matsh@ifi.uio.no), Sigurd Salvesen, sigurds@ifi.uio.no.
 * @info Originally code by Mats Hovin. Rewritten & modified by Sigurd Salvesen.
 */

public class WaitRobotFinished {
    private String nxUrlString = "http://192.168.255.1:8080"; // Robot address
    private URL url;
    private byte[] byteMessage;

    WaitRobotFinished() throws Exception {
        String uT;

        byteMessage = uT.getBytes();
        url = new URL(nxUrlString);
    }

    // Waiting for the robot to finish
    public void waiting() throws Exception {
        while (!semaphoreHigh());
    }

    // Check if the robot has finished
    private boolean semaphoreHigh() throws Exception {
        boolean semaphoreHigh = false;

        // Initialize connection
        String SOAPAction = "";
        URLConnection connection = url.openConnection();
        HttpURLConnection httpConn = (HttpURLConnection) connection;

        httpConn.setRequestProperty("Content-Length", String.valueOf(byteMessage.length));
httpConn.setRequestProperty("Content-Type", "text/xml; charset=utf-8");
httpConn.setRequestProperty("SOAPAction", SOAPAction);
httpConn.setRequestMethod("POST");
httpConn.setDoOutput(true);
httpConn.setDoInput(true);

// Send data
OutputStream out = httpConn.getOutputStream();
out.write(byteMessage);
out.close();

// Read data
InputStreamReader isr = new InputStreamReader(httpConn.getInputStream());
BufferedReader in = new BufferedReader(isr);

String inputLine;

while ((inputLine = in.readLine()) != null) {
    if (inputLine.indexOf("<item>1")) > 0)
        semaphoreHigh = true;
}
in.close();
return semaphoreHigh;

Listing A.4: WaitRobotFinished.java
A.2 NX100 Program

Listing A.5: nx100.inform

1 NOP
2 SETE P000 (4) 18000
3 SETE P000 (5) 0
4 SETE P000 (7) 1447
5 SET D000 1
6 *WAIT
7 JUMP *WAIT IF D000=1
8 SETE P000 (1) D001
9 SETE P000 (2) D002
10 SETE P000 (3) D003
11 SETE P000 (6) D004
12 /* Add code: open/close gripper */
13 MOVL P000 V=80.0
14 SET D000 1
15 JUMP *WAIT
16 END
A.3 Packing Algorithm

Listing A.6: PalletHandler.java

```
import com.cycling74.max.*;
import java.awt.*;
import javax.swing.*;

/**
 * @description A system for generating/searching for good/optimal pallet configuration for an autonomous robot-based pallet loading system.
 * @class PalletHandler - A Max object for generating and handling pallet schemas.
 * @author Sigurd Salvesen, sigurdsa@ifi.uio.no.
 */

public class PalletHandler extends MaxObject {
    private int packageHeight = 0;
    private int packageWidth = 0;
    private int packageDepth = 0;

    private int palletHeight = 0;
    private int palletWidth = 0;
    private int palletDepth = 0;

    private boolean reset = true; // no package placed

    private int x = 0;
    private int y = 0;
    private int z = 0;
    private boolean rotate = false;
    private int packageNumber = 0;

    private float errorMargin = 0;

    private Pallet pallet = null;
    private DrawPallet palletWindow = null;

    private static final String[] INLET_ASSIST = new String[] { "inlet 1 help" };
    private static final String[] OUTLET_ASSIST = new String[] { "outlet 1 help" };

    public PalletHandler(boolean bang, boolean reset, float packageHeight, 
        float packageWidth, float packageDepth, int palletHeight, 
        int palletWidth, int palletDepth, float errorMargin) {
        declareInlets(new int[] { DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, 
            DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, 
            DataTypes.ALL, DataTypes.ALL, DataTypes.ALL });
        declareOutlets(new int[] { DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, 
            DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, DataTypes.ALL, 
            DataTypes.ALL, DataTypes.ALL, DataTypes.ALL });

        setInletAssist(new String[] { "bang to output", "bang to reset", 
            "package height", "package width", "package depth", 
            "pallet height", "pallet width", "pallet depth", "error margin" });
        setOutletAssist(new String[] { "finished", "x", "y", "z", "rotate", 
            "package_height", "package_width", "package_depth", 
            "package_number" });
    }

    DrawPallet palletWindow = null;
    public static final String[] INLET_ASSIST = new String[] { "inlet 1 help" };
    public static final String[] OUTLET_ASSIST = new String[] { "outlet 1 help" };

```

98
```java
public void bang() {
    int inletNum = getInlet();
    post("PalletHandler: banged at: " + inletNum);

    if (inletNum == 0) { // bang
        post("PalletHandler: bang - output placement (x, y, z)\n" + inletNum);
        if (pallet == null || reset) {
            // Generate a new pallet scheme
            PalletSearch scheme = new PalletSearch(palletWidth,
                                                   palletDepth,
                                                   palletHeight,
                                                   packageWidth,
                                                   packageDepth,
                                                   packageHeight);

            pallet = scheme.makeScheme();
            pallet.visualize();
        }

        Package currPackage = pallet.getCurrPackage(true);
        packageHeight = currPackage.getHeight();
        packageWidth = currPackage.getWidth();
        packageDepth = currPackage.getDepth();

        x = currPackage.getX();
        y = currPackage.getY();
        z = currPackage.getZ();

        rotate = currPackage.getRotated();
        packageNumber = currPackage.getPackageNumber();

        sendPos();
    } else if (inletNum == 1) { // reset
        post("PalletHandler: reset");
        reset();
    }
}

public void inlet(int i) {
    int inletNum = getInlet();

    if (inletNum == 5) { // pallet height
        palletHeight = i;
    } else if (inletNum == 6) { // pallet width
        palletWidth = i;
    } else if (inletNum == 7) { // pallet depth
        palletDepth = i;
    }
}

public void inlet(float f) {
    int inletNum = getInlet();

    if (inletNum == 2) {
        float diff = Math.abs(packageHeight - f);
    }
```

if (!reset && diff > errorMargin)
    illegalPackageSize("height", diff);
else
    packageHeight = (int) f + 1;
} else if (inletNum == 3) {
    float diff = Math.abs(packageWidth - f);
    if (!reset && diff > errorMargin)
        illegalPackageSize("width", diff);
    else
        packageWidth = (int) f + 1;
} else if (inletNum == 4) {
    float diff = Math.abs(packageDepth - f);
    if (!reset && diff > errorMargin)
        illegalPackageSize("depth", diff);
    else
        packageDepth = (int) f + 1;
} else if (inletNum == 8) {
    post("PalletHandler: new errorMargin:" + f);
    errorMargin = f;
}

// Reset pallet (empty pallet)
public void reset() {
    reset = true;
}

// Send X, Y and Z-position and rotation at the pallet to Max
public void sendPos() {
    if (reset) {
        reset = false;
    } else {
        outlet(0, true);
        outlet(1, x);
        outlet(2, y);
        outlet(3, z);
        outlet(4, rotate);
        outlet(5, packageHeight);
        outlet(6, packageWidth);
        outlet(7, packageDepth);
        outlet(8, packageNumber);
    }

// New package size diff. > error_margin
public void illegalPackageSize(String side, float diff) {
    post("PalletHandler: empty illegalPackageSize (side:" + side + " | diff:" + diff + ")");
}
import com.cycling74.max.*;

/** *
 * @description A system for generating/searching for good/optimal pallet configuration for an autonomous robot-based pallet loading system.
 * @class PalletOrganizer – Testing the functionality used from Max.
 * @author Sigurd Salvesen , sigurdsa@ifi.uio.no.
 */

public class PalletOrganizer {
    private static int palletWidth = 300;
    private static int palletDepth = 282;
    private static int palletHeight = 200;
    private static int packageWidth = 23;
    private static int packageDepth = 29;
    private static int packageHeight = 15;

    private static Pallet pallet = null;

    public static void main(String[] args) {
        run();
    }

    // Test functions used from Max
    public static void run() {
        PalletSearch scheme = new PalletSearch(palletWidth, palletDepth, palletHeight, packageWidth, packageDepth, packageHeight);

        pallet = scheme.makeScheme();
        pallet.visualize();
    }
}

import com.cycling74.max.*;

/** *
 * @description A system for generating/searching for good/optimal pallet configuration for an autonomous robot-based pallet loading system.
 * @class PalletSearch – Search for different pallet schemas
 * @author Sigurd Salvesen , sigurdsa@ifi.uio.no.
 */

public class PalletSearch {
    private int palletWidth;
    private int palletDepth;
    private int palletHeight;
    private int packageWidth;
    private int packageDepth;
    private int packageHeight;

    private PalletConfigBlockHeuristics bestPalletBlocks;
PalletSearch(int palletWidth, int palletDepth, int palletHeight,  
   int packageWidth, int packageDepth, int packageHeight) {
    this.palletWidth = palletWidth;  
    this.palletDepth = palletDepth;  
    this.palletHeight = palletHeight;  
    this.packageWidth = packageWidth;  
    this.packageDepth = packageDepth;  
    this.packageHeight = packageHeight;  
}

public PalletConfig search(boolean print, boolean display) {
    System.out.flush();
    long start = System.currentTimeMillis();
    // Block BRUTE FORCE
    bestPalletBlocks = new PalletConfigBlockHeuristics(palletWidth,  
        palletDepth,  
        palletHeight,  
        packageWidth,  
        packageDepth,  
        packageHeight);  
    PalletConfigBlockHeuristics currPallet =  
        (PalletConfigBlockHeuristics) bestPalletBlocks.clone();  
    int currPackageDepth;  
    int currPackageWidth;
    int count = 0;
    for (int run = 0; run < 2; run++) {
        if (run == 0) {
            currPackageDepth = packageDepth;
            currPackageWidth = packageWidth;
        } else {
            currPackageDepth = packageWidth;
            currPackageWidth = packageDepth;
        }
        for (int a = 0; a <= palletWidth / currPackageWidth; a++) {
            currPallet.setA(a * currPackageWidth);
            for (int c = 0; c <= currPallet.getA() / currPackageDepth; c++) {
                currPallet.setC(c * currPackageDepth);
                for (int d = 0; d <= palletDepth / currPackageWidth; d++) {
                    currPallet.setD(d * currPackageWidth);
                    for (int h = 0; h <= currPallet.getD() / currPackageDepth; h++) {
                        currPallet.setH(h * currPackageDepth);
                        if (currPallet.getFitness() > bestPalletBlocks.getFitness())
                            bestPalletBlocks = currPallet.clone();
                        count++;
                    }
                }
            }
        }
    }
}
82 }
83 }
84 }
85

86 long end = System.currentTimeMillis();
87
88 if (print)
89 System.out.println(bestPalletBlocks.getPacakgesPlaced() + "\t");
90
91 if (display)
92 new DrawPallet(bestPalletBlocks, "Block (brute force): "+ bestPalletBlocks.getFitness(),
93 640, 340);
94
95 if (print)
96 System.out.println((end - start) + "(ms)");
97
98 return bestPalletBlocks;
99 }
100 }
101
102 public Pallet makeScheme() {
103 PalletConfig palletConfig = search(true, true);
104 Pallet pallet = palletConfig.convertToPalletFormat();
105 pallet.setCurrPackage(pallet.getFirstPackage());
106
107 return pallet;
108 }
109 }
110 }

Listing A.8: PalletSearch.java

import com.cycling74.*;
/*
 * @description A system for generating/searching for good/optimal pallet
 * configuration for an autonomous robot-based pallet loading
 * system.
 * @class PalletConfig - An abstract class for holding information of a pallet
 * and it's packages.
 * @author Sigurd Salvesen, sigurdsa@ifi.uio.no.
 */
public abstract class PalletConfig {
 protected int palletWidth;
 protected int palletDepth;
 protected int palletHeight;
 protected int packageWidth;
 protected int packageDepth;
 protected int packageHeight;

 PalletConfig(int palletWidth, int palletDepth, int palletHeight,
     int packageWidth, int packageDepth, int packageHeight) {
     this.palletWidth = palletWidth;
     this.palletDepth = palletDepth;
     this.palletHeight = palletHeight;
     this.packageWidth = packageWidth;
     this.packageDepth = packageDepth;
     this.packageHeight = packageHeight;
}
```java
public int getPalletWidth() {
    return palletWidth;
}

public int getPalletDepth() {
    return palletDepth;
}

public int getPackageWidth() {
    return packageWidth;
}

public int getPackageDepth() {
    return packageDepth;
}

abstract public void rotate(int i);

abstract public void rotateAll();

abstract public void rotateAll();

abstract public float getFitness() {
    return (float) (getPackageDepth() * getPackageWidth() * getPacakagesPlaced())
        / (getPalletDepth() * getPalletWidth());
}

abstract public int getPacakagesPlaced();

abstract public PalletConfig clone();

abstract public Pallet convertToPalletFormat();
```

Listing A.9: PalletConfig.java

```java
import com.cycling74.max.*;

/* *
 * @description A system for generating/searching for good/optimal pallet
 * configuration for an autonomous robot–based pallet loading
 * system.
 * @class PalletConfigRowRotate – Class for holding information of a pallet and
 * it’s packages – A given number of rows are rotated.
 * @author Sigurd Salvesen, sigurds@ifi.uio.no.
 */

public class PalletConfigRowRotate extends PalletConfig {
    private int rowsRotated;

    PalletConfigRowRotate(int palletWidth, int palletDepth, int packageWidth,
                          int packageDepth, int packageHeight) {
        super(palletWidth, palletDepth, 0, packageWidth, packageDepth,
             packageHeight);
        this.rowsRotated = 0;
    }

    PalletConfigRowRotate(int palletWidth, int palletDepth, int packageWidth,
...
```

Listing A.10: PalletConfigRowRotate.java
int packageDepth, int packageHeight, int rowsRotated) {
    super(palletWidth, palletDepth, 0, packageWidth, packageDepth,
          packageHeight);
    this.rowsRotated = rowsRotated;
}

public void rotate(int i) {
    rowsRotated = i;
}

public void rotateAll() {
    rowsRotated = 9999;
}

public int getRotatedRows() {
    return rowsRotated;
}

// Return the maximum numbers of rows that fits into the pallet
public int getMaxRows() {
    if (packageDepth < packageWidth)
        return palletDepth / packageDepth;
    else
        return palletDepth / packageWidth;
}

// Count the number of packages that fits into the pallet
public int getPacakagesPlaced() {
    int currPosY = 0;
    int packagesPlaced = 0;
    int i = 0;

    // Rotated rows
    for (; i < rowsRotated; i++) {
        if ((currPosY + packageWidth) > palletDepth)
            break;
        packagesPlaced += palletWidth / packageDepth;
        currPosY += packageWidth;
    }

    // None-rotated rows
    for (; i <= getMaxRows(); i++) {
        if ((currPosY + packageDepth) > palletDepth)
            break;
        packagesPlaced += palletWidth / packageWidth;
        currPosY += packageDepth;
    }

    return packagesPlaced;
}

// Find the optimal number of rows rotated
public PalletConfig getBest() {
    PalletConfig bestPallet = this.clone();
    PalletConfig currPallet = this.clone();
    for (int i = 0; i <= getMaxRows(); i++) {

currPallet.rotate(i);

if (currPallet.getFitness() > bestPallet.getFitness())
    bestPallet = (PalletConfigRowRotate) currPallet.clone();
}

return bestPallet;
}

// Make a copy
public PalletConfig clone() {
    return new PalletConfigRowRotate(palletWidth, palletDepth,
        packageWidth, packageDepth,
        packageHeight, rowsRotated);
}

// Convert to Pallet class format
public Pallet convertToPalletFormat() {
    // Not implemented
    return null;
}

Listing A.10: PalletConfigRowRotate.java

import com.cycling74.max.*;

/**
 * @description A system for generating/searching for good/optimal pallet
 * configuration for an autonomous robot–based pallet loading
 * system.
 * @class PalletConfigEqualRotate – A class for holding information of a pallet
 * and it's packages — All packages are either rotated or not rotated
 * @author Sigurd Salvesen, sigurds@ifi.uio.no.
 */

public class PalletConfigEqualRotate extends PalletConfig {
    private boolean rotated;

    PalletConfigEqualRotate(int palletWidth, int palletDepth, int palletHeight, 
        int packageWidth, int packageDepth, int packageHeight) {
        super(palletWidth, palletDepth, palletHeight, 
            packageWidth, packageDepth, packageHeight);
        rotated = false;
    }

    PalletConfigEqualRotate(int palletWidth, int palletDepth, int palletHeight, 
        int packageWidth, int packageDepth, int packageHeight, 
        boolean rotated) {
        super(palletWidth, palletDepth, palletHeight, 
            packageWidth, packageDepth, packageHeight);
        this.rotated = rotated;
    }

    public void rotate(int i) {
        if (i == 0)
            rotated = false;
else
    rotated = true;
}

public void rotateAll() {
    rotated = true;
}

public boolean isRotated() {
    return rotated;
}

// Count the number of packages that fits into the pallet
public int getPackagesPlaced() {
    int sum;
    if (rotated)
        sum = (palletWidth / packageDepth) * (palletDepth / packageWidth);
    else
        sum = (palletWidth / packageWidth) * (palletDepth / packageDepth);
    if (sum < 0)
        return 0;
    else
        return sum;
}

// Find the optimal number of rows rotated
public PalletConfig getBest() {
    PalletConfig reversePallet = this.clone();
    if (isRotated())
        reversePallet.rotate(0);
    else
        reversePallet.rotateAll();
    if (getFitness() > reversePallet.getFitness())
        return this;
    else
        return reversePallet;
}

// Make a copy
public PalletConfig clone() {
    return new PalletConfigEqualRotate(palletWidth, palletDepth, palletHeight, packageWidth, packageDepth, packageHeight, rotated);
}

// Convert to Pallet class format
public Pallet convertToPalletFormat() {
    // Not implemented
    return null;
}

Listing A.11: PalletConfigEqualRotate.java

Listing A.12: PalletConfigBlockHeuristics.java

import com.cycling74.max.*;

107
public class PalletConfigBlockHeuristics extends PalletConfig {
    private int a;
    private int e;
    private int f;
    private int h;
    private int c;
    private int g;
    private int b;
    private int d;
    private PalletConfig block1;
    private PalletConfig block2;
    private PalletConfig block3;
    private PalletConfig block4;
    PalletConfigBlockHeuristics(int palletWidth, int palletDepth, int palletHeight, int packageWidth, int packageDepth) {
        super(palletWidth, palletDepth, palletHeight, packageWidth, packageDepth);
        block1 = null;
        block2 = null;
        block3 = null;
        block4 = null;
        setA(0);
        setC(0);
        setD(0);
        setH(0);
    }
    public void rotate(int i) {
        // Not in use
    }
    public void rotateAll() {
        // Not in use
    }
    public int getA() {
        return a;
    }
    public void setA(int a) {
        this.a = a;
        e = palletWidth - a;
    }
    public int getE() {

public int getF() {
    return f;
}

public int getH() {
    return h;
}

public void setH(int h) {
    this.h = h;
    f = palletDepth - h;
}

public int getC() {
    return c;
}

public void setC(int c) {
    this.c = c;
    g = palletWidth - c;
}

public int getG() {
    return g;
}

public int getB() {
    return b;
}

public int getD() {
    return d;
}

public void setD(int d) {
    this.d = d;
    b = palletDepth - d;
}

private void printVariables() {
    System.out.println("A: " + a);
    System.out.println("B: " + b);
    System.out.println("C: " + c);
    System.out.println("D: " + d);
    System.out.println("E: " + e);
    System.out.println("F: " + f);
    System.out.println("G: " + g);
    System.out.println("H: " + h);
}

public int getPalletWidth() {
    return palletWidth;
}

public int getPalletDepth() {
    return palletDepth;
}

public int getPackageWidth() {

}
return packageWidth;
}

public int getPackageDepth() {
    return packageDepth;
}

public int getPackageDepth(int i) {
    return 0;
}

public int getPackageWidth(int i) {
    return 0;
}

public PalletConfig getBlock1() {
    return block1;
}

public PalletConfig getBlock2() {
    return block2;
}

public PalletConfig getBlock3() {
    return block3;
}

public PalletConfig getBlock4() {
    return block4;
}

// Search for best rotation with given block sizes
public int getPacakgesPlaced() {
    PalletConfig block1_rotated = block1.clone();
    PalletConfig block2_rotated = block2.clone();
    PalletConfig block3_rotated = block3.clone();
    PalletConfig block4_rotated = block4.clone();

    block1_rotated.rotateAll();
    block2_rotated.rotateAll();
    block3_rotated.rotateAll();
    block4_rotated.rotateAll();

    int sum1 = block1.getPacakgesPlaced() +
               block2_rotated.getPacakgesPlaced() +
               block3_rotated.getPacakgesPlaced() +
               block4.getPacakgesPlaced();

    int sum2 = block1_rotated.getPacakgesPlaced() +
               block2_rotated.getPacakgesPlaced() +
               block3_rotated.getPacakgesPlaced() +
               block4_rotated.getPacakgesPlaced();

    return Math.min(sum1, sum2);
}
if (sum1 >= sum2) {
    block2.rotateAll();
    block3.rotateAll();
} else {
    block1.rotateAll();
    block4.rotateAll();
}

return block1.getPacakagesPlaced() + block2.getPacakagesPlaced() +
    block3.getPacakagesPlaced() + block4.getPacakagesPlaced();

// Make a copy
public PalletConfigBlockHeuristics clone() {
    PalletConfigBlockHeuristics newPallet = new PalletConfigBlockHeuristics(
        palletWidth, palletDepth, palletHeight, packageWidth, packageDepth, packageHeight);

    newPallet.setA(getA());
    newPallet.setC(getC());
    newPallet.setD(getD());
    newPallet.setH(getH());

    newPallet.getFitness(); // Fix block 1–4

    return newPallet;
}

// Convert to Pallet class format
public Pallet convertToPalletFormat() {
    Pallet pallet = new Pallet(palletWidth, palletDepth, palletHeight);

    // Add layers
    for (int z = 0; (z + packageHeight) <= palletHeight; z += packageHeight) {
        // Add packages from the 4 blocks to current layer
        placePackages(0, getD(), z, getBlock1(), pallet);
        placePackages(0, 0, z, getBlock2(), pallet);
        placePackages(getA(), getH(), z, getBlock3(), pallet);
        placePackages(getC(), 0, z, getBlock4(), pallet);
    }

    return pallet;
}

// Add packages from one block
private void placePackages(int startX, int startY, int z, PalletConfig palletBlock, Pallet newPallet) {
    int currX = 0;
    int currY = 0;

    int packageWidth = palletBlock.getPacakagesPlaced();
    int packageDepth = palletBlock.getPacakagesPlaced();
if (((PalletConfigEqualRotate) palletBlock).isRotated()) {
    packageWidth = palletBlock.getPackageDepth();
    packageDepth = palletBlock.getPackageWidth();
    rotated = true;
}

// Add packages
while (currY + packageDepth <= palletBlock.getPalletDepth()) {
    currX = 0;

    while (currX + packageWidth <= palletBlock.getPalletWidth()) {
        // Generate placement (X- and Y-coordinates)
        int packageX = startX + currX;
        int packageY = startY + currY;

        newPallet
            .addPackage(new Package(
                packageWidth,
                packageDepth,
                packageHeight,
                rotated,
                packageX,
                packageY,
                z,
                newPallet.getPackagesPlaced() + 1));

        newPallet.increasePackagesPlaced();
        currX += packageWidth;
    }

    currY += packageDepth;
}

Listing A.12: PalletConfigBlockHeuristics.java

Listing A.13: Pallet.java

import com.cycling74.max.*;

/**
 * @description A system for generating/searching for good/optimal pallet
 * configuration for an autonomous robot-based pallet loading system.
 * @class Pallet -- Holding information of a pallet and it's packages.
 * @author Sigurd Salvesen, sigurdsa@ifi.uio.no.
 */

public final class Pallet {

    private int width;
    private int depth;
    private int height;
    private int packagesPlaced = 0;
    private Package firstPackage = null;
    private Package currPackage = null;

    public Pallet(int width, int depth, int height) {
```java
    this.width = width;
    this.depth = depth;
    this.height = height;
}

public int getWidth() {
    return width;
}

public int getDepth() {
    return depth;
}

public int getHeight() {
    return depth;
}

public void setFirstPackage(Package p) {
    firstPackage = p;
    setCurrPackage(p);
}

public void setCurrPackage(Package p) {
    currPackage = p;
}

public Package getFirstPackage() {
    return firstPackage;
}

public Package getCurrPackage() {
    return currPackage;
}

public Package getCurrPackage(boolean move) {
    Package p = getCurrPackage();
    if (move)
        setCurrPackage(p.getNextPackage());
    return p;
}

public Package getNextPackage() {
    currPackage = currPackage.getNextPackage();
    return currPackage();
}

public void addPackage(Package p) {
    if (getFirstPackage() == null) {
        setFirstPackage(p);
    } else {
        getCaption().setNextPackage(p);
        p.setPrevPackage(getCurrPackage());
        setCurrPackage(p);
    }
}

public int getPackagePlaced() {
    return packagesPlaced;
}
```

113
public void setPackagesPlaced(int number) {
    packagesPlaced = number;
}

public int increasePackagesPlaced() {
    packagesPlaced++;
    return packagesPlaced;
}

public void visualize() {
    new DrawPallet(this, "Visualization of pallet", 50, 50);
}

import com.cycling74.max.*;

/* *
* @description A system for generating/searching for good/optimal pallet
* configuration for an autonomous robot–based pallet loading
* *
* @class Package – Holding information of a single package
* *
* @author Sigurd Salvesen, sigurds@ifi.uio.no.
*/

public final class Package {
    private int width;
    private int depth;
    private int height;
    private boolean rotated;
    private int x;
    private int y;
    private int z;
    private int packageNumber;
    private Package prevPackage = null;
    private Package nextPackage = null;

    public Package(int width, int depth, int height, boolean rotated, int x, int y, int z, int packageNumber) {
        this.width = width;
        this.depth = depth;
        this.height = height;
        this.rotated = rotated;
        this.x = x;
        this.y = y;
        this.z = z;
        this.packageNumber = packageNumber;
    }

    public void setPrevPackage(Package prevPackage) {
        this.prevPackage = prevPackage;
    }

    public void setPrevPackagePointers() {

if (prevPackage != null)
    prevPackage.setNextPackage(this);

public int getDepth() {
    return depth;
}

public int getWidth() {
    return width;
}

public int getHeight() {
    return height;
}

public boolean getRotated() {
    return rotated;
}

public void setXYZ(int x, int y, int z) {
    this.x = x;
    this.y = y;
    this.z = z;
}

public int getX() {
    return x;
}

public int getY() {
    return y;
}

public int getZ() {
    return z;
}

public int getPackageNumber() {
    return packageNumber;
}

public void setNextPackage(Package p) {
    nextPackage = p;
    nextPackage.setPrevPackage(this);
}

public Package getNextPackage() {
    return nextPackage;
}

public void print(boolean[][] utskrift, int pos) {
    for (int y = 0; y < depth; y++) {
        for (int x = pos; x < pos + width; x++) {
            utskrift[y][x] = true;
        }
    }
    if (nextPackage != null)
        nextPackage.print(utskrift, pos + width);
}
import com.cycling74.max.*;
import java.awt.*;
import javax.swing.*;

/**
 * @description A system for generating/searching for good/optimal pallet configuration for an autonomous robot-based pallet loading system.
 * @class DrawPallet - Visualize pallet configuration.
 * @author Sigurd Salvesen, sigurds@ifi.uio.no.
 */

public class DrawPallet extends Canvas {
    private PalletConfig pallet = null;
    private Pallet palletFinal = null;

    public DrawPallet(PalletConfig pallet, String title, int x, int y) {
        this.pallet = pallet;
        JFrame frame = new JFrame();
        frame.setBounds(x, y, pallet.getPalletWidth() + 10, pallet.getPalletDepth() + 30);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().add(this);
        frame.setTitle(title);
        frame.setVisible(true);
    }

    public DrawPallet(Pallet pallet, String title, int x, int y) {
        this.palletFinal = pallet;
        JFrame frame = new JFrame();
        frame.setBounds(x, y, pallet.getWidth() + 10, pallet.getDepth() + 30);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().add(this);
        frame.setTitle(title);
        frame.setVisible(true);
    }

    public void paint(Graphics graphics) {
        if (palletFinal != null) { // Pallet format
            drawPackages(0, 0, palletFinal, graphics);
        } else if (!(pallet instanceof PalletConfigBlockHeuristics)) {
            drawPackages(0, 0, pallet, graphics);
        } else if (pallet instanceof PalletConfigBlockHeuristics) {
            PalletConfigBlockHeuristics pallet2 = (PalletConfigBlockHeuristics) pallet.clone();
            drawPackages(0, 0, pallet2, graphics);
        }
    }
}

Listing A.14: Package.java

Listing A.15: DrawPallet.java
graphics.setColor(Color.DARK_GRAY);
graphics.fillRect(pallet2.getC(), pallet2.getH(), pallet2.getA() - pallet2.getC(), pallet2.getD() - pallet2.getH());

// Draw packages
drawPackages(0, pallet2.getD(), pallet2.getBlock1(), graphics);
drawPackages(0, 0, pallet2.getBlock2(), graphics);
drawPackages(pallet2.getA(), pallet2.getH(), pallet2.getBlock3(), graphics);
drawPackages(pallet2.getC(), 0, pallet2.getBlock4(), graphics);
graphics.setColor(Color.yellow);
graphics.drawRect(0, pallet2.getD(), pallet2.getA(), pallet2.getB());
graphics.setColor(Color.cyan);
graphics.drawRect(0, 0, pallet2.getC(), pallet2.getD());
graphics.setColor(Color.magenta);
graphics.drawRect(pallet2.getA(), pallet2.getH(), pallet2.getE(), pallet2.getF());
graphics.setColor(Color.green);
graphics.drawRect(pallet2.getC(), 0, pallet2.getG(), pallet2.getH());
}
}

// Draw packages (Pallet class format)
private void drawPackages(int startX, int startY, Pallet palletFinal, Graphics graphics) {
    Package currPackage = palletFinal.getFirstPackage();
    Graphics graphics = null;

    // Draw only one layer
    while (currPackage != null && currPackage.getZ() == 0) {
        boolean rotated = currPackage.getRotated();
        drawPackage(rotated, currPackage.getX(), currPackage.getY(),
                    currPackage.getWidth(), currPackage.getDepth(),
                    graphics);
        currPackage = currPackage.getNextPackage();
    }
}

// Draw packages
private void drawPackages(int startX, int startY, PalletConfig pallet, Graphics graphics) {
    int currPosX = 0;
    int currPosY = 0;
    int packageWidth;
    int packageDepth;
    packageWidth = pallet.getPackageWidth();
    packageDepth = pallet.getPackageDepth();

    if (pallet instanceof PalletConfigEqualRotate) {
        boolean rotated = false;
        if (((PalletConfigEqualRotate) pallet).isRotated()) {
            packageWidth = pallet.getPackageDepth();
        }
    }

packageDepth = pallet.getPackageWidth();
rotated = true;
}
while (currPosY + packageDepth <= pallet.getPalletDepth()) {
    currPosX = 0;
    while (currPosX + packageWidth <= pallet.getPalletWidth()) {
        drawPackage(rotated, startX + currPosX, startY + currPosY, 
        packageWidth, packageDepth, graphics);
        currPosX += packageWidth;
    }
    currPosY += packageDepth;
}
// Draw a single package
private void drawPackage(boolean rotated, int x, int y, int packageDepth, 
int packageWidth, Graphics graphics) {
    if (rotated) 
        graphics.setColor(Color.pink);
    else 
        graphics.setColor(Color.LIGHT_GRAY);
    graphics.fillRect(x, y, packageDepth, packageWidth);
    graphics.setColor(Color.black);
    graphics.drawRect(x, y, packageDepth, packageWidth);
}

Listing A.15: DrawPallet.java

Listing A.16: PalletLoadingTest.java

import com.cycling74.max.*;

/**
 * @description A system for generating/searching for good/optimal pallet 
 * configuration for an autonomous robot-based pallet loading system.
 * @class PalletLoadingTest – Main class – Generate different tests
 * @author Sigurd Salvesen, sigurds@ifi.uio.no.
 */

public class PalletLoadingTest {
    public static void main(String[] args) {
        PalletSearch[] tests = new PalletSearch[25];
        boolean print = true;
        boolean display = false;
        System.out.println("*** START ***");
        // this.palletWidth = palletWidth;
        // this.palletDepth = palletDepth;
        // this.palletHeight = palletHeight;
        // this.packageWidth = packageWidth;
        // this.packageDepth = packageDepth;
        // this.packageHeight = packageHeight;
    }
}
// First order
System.out.println("* First order *");

tests[0] = new PalletSearch(22, 16, 0, 5, 3, 0);
tests[1] = new PalletSearch(57, 44, 0, 12, 5, 0);
tests[2] = new PalletSearch(86, 82, 0, 15, 11, 0);
tests[3] = new PalletSearch(42, 39, 0, 9, 4, 0);
tests[4] = new PalletSearch(124, 81, 0, 11, 9, 0);
tests[5] = new PalletSearch(40, 25, 0, 7, 3, 0);
tests[6] = new PalletSearch(52, 33, 0, 9, 4, 0);
tests[7] = new PalletSearch(56, 52, 0, 12, 5, 0);
tests[8] = new PalletSearch(42, 39, 0, 9, 4, 0);
tests[9] = new PalletSearch(87, 47, 0, 7, 6, 0);

tests[0].search(print, display);
tests[1].search(print, display);
tests[2].search(print, display);
tests[3].search(print, display);
tests[4].search(print, display);
tests[5].search(print, display);
tests[6].search(print, display);
tests[7].search(print, display);
tests[8].search(print, display);
tests[9].search(print, display);

// Superior order
System.out.println("* Superior order *");

tests[10] = new PalletSearch(43, 26, 0, 7, 3, 0);
tests[11] = new PalletSearch(63, 44, 0, 10, 5, 0);
tests[12] = new PalletSearch(61, 35, 0, 10, 3, 0);
tests[13] = new PalletSearch(61, 38, 0, 6, 5, 0);
tests[14] = new PalletSearch(141, 71, 0, 13, 8, 0);
tests[15] = new PalletSearch(106, 59, 0, 13, 5, 0);
tests[16] = new PalletSearch(74, 46, 0, 7, 5, 0);
tests[17] = new PalletSearch(86, 52, 0, 9, 5, 0);
tests[18] = new PalletSearch(108, 65, 0, 10, 7, 0);

tests[10].search(print, display);
tests[11].search(print, display);
tests[12].search(print, display);
tests[13].search(print, display);
tests[14].search(print, display);
tests[15].search(print, display);
tests[16].search(print, display);
tests[17].search(print, display);
tests[18].search(print, display);
tests[19].search(print, display);
tests[20].search(print, display);

// EQUAL ROTATION
System.out.println("\n\n\n** EQUAL ROTATION **");

// First order
System.out.println("* First order *");

print(new PalletConfigEqualRotate(22, 16, 0, 5, 3, 0))
.getBest()
.getPacakgesPlaced());

print(new PalletConfigEqualRotate(57, 44, 0, 12, 5, 0))
.getBest()
.getPacakgesPlaced());

119
87 print(new PalletConfigEqualRotate(86, 82, 0, 15, 11, 0)
88 .getBest()
89 .getPacakagesPlaced());
90
91 print(new PalletConfigEqualRotate(42, 39, 0, 9, 4, 0)
92 .getBest()
93 .getPacakagesPlaced());
94
95 print(new PalletConfigEqualRotate(124, 81, 0, 21, 10, 0)
96 .getBest()
97 .getPacakagesPlaced());
98
99 print(new PalletConfigEqualRotate(40, 25, 0, 7, 3, 0)
100 .getBest()
101 .getPacakagesPlaced());
102
103 print(new PalletConfigEqualRotate(52, 33, 0, 9, 4, 0)
104 .getBest()
105 .getPacakagesPlaced());
106
107 print(new PalletConfigEqualRotate(56, 52, 0, 12, 5, 0)
108 .getBest()
109 .getPacakagesPlaced());
110
111 print(new PalletConfigEqualRotate(87, 47, 0, 7, 6, 0)
112 .getBest()
113 .getPacakagesPlaced());
114
115 print(new PalletConfigEqualRotate(67, 44, 0, 6, 5, 0)
116 .getBest()
117 .getPacakagesPlaced());
118
119 System.out.println(" * Superior order *");
120 print(new PalletConfigEqualRotate(43, 26, 0, 7, 3, 0)
121 .getBest()
122 .getPacakagesPlaced());
123
124 print(new PalletConfigEqualRotate(63, 44, 0, 8, 5, 0)
125 .getBest()
126 .getPacakagesPlaced());
127
128 print(new PalletConfigEqualRotate(61, 35, 0, 10, 3, 0)
129 .getBest()
130 .getPacakagesPlaced());
131
132 print(new PalletConfigEqualRotate(61, 38, 0, 6, 5, 0)
133 .getBest()
134 .getPacakagesPlaced());
135
136 print(new PalletConfigEqualRotate(61, 38, 0, 10, 3, 0)
137 .getBest()
138 .getPacakagesPlaced());
139
140 print(new PalletConfigEqualRotate(61, 38, 0, 10, 3, 0)
141 .getBest()
142 .getPacakagesPlaced());
143
144 print(new PalletConfigEqualRotate(93, 46, 0, 13, 4, 0)
145 .getBest()
146 .getPacakagesPlaced());
147
148 print(new PalletConfigEqualRotate(141, 71, 0, 13, 8, 0)
149 .getBest()
150 .getPacakagesPlaced());
151
152 print(new PalletConfigEqualRotate(106, 59, 0, 13, 5, 0)
153 .getBest()
154 .getPacakagesPlaced());
155
156 print(new PalletConfigEqualRotate(74, 46, 0, 7, 5, 0)
157 .getBest()
158 .getPacakagesPlaced());
159
160 print(new PalletConfigEqualRotate(86, 52, 0, 9, 5, 0)
161 .getBest()
162 .getPacakagesPlaced());
163
164 print(new PalletConfigEqualRotate(108, 65, 0, 10, 7, 0)
165 .getBest()
166 .getPacakagesPlaced());
167
168 // ROW ROTATION
149 System.out.println("\n\n** ROW ROTATION **");
150
// First order
151 System.out.println("* First order *");
152 print(new PalletConfigRowRotate(22, 16, 5, 3, 0).getBest()
153 .getPacakagesPlaced());
154 print(new PalletConfigRowRotate(57, 44, 12, 5, 0).getBest()
155 .getPacakagesPlaced());
156 print(new PalletConfigRowRotate(86, 82, 15, 11, 0).getBest()
157 .getPacakagesPlaced());
158 print(new PalletConfigRowRotate(42, 39, 9, 4, 0).getBest()
159 .getPacakagesPlaced());
160 print(new PalletConfigRowRotate(124, 81, 21, 10, 0)
161 .getBest()
162 .getPacakagesPlaced()));
163 print(new PalletConfigRowRotate(40, 25, 7, 3, 0).getBest()
164 .getPacakagesPlaced());
165 print(new PalletConfigRowRotate(52, 33, 9, 4, 0).getBest()
166 .getPacakagesPlaced());
167 print(new PalletConfigRowRotate(56, 52, 12, 5, 0).getBest()
168 .getPacakagesPlaced());
169 print(new PalletConfigRowRotate(87, 47, 7, 6, 0).getBest()
170 .getPacakagesPlaced());
171 print(new PalletConfigRowRotate(67, 44, 6, 5, 0).getBest()
172 .getPacakagesPlaced()));
173
// Superior order
174 System.out.println("* Superior order *");
175 print(new PalletConfigRowRotate(43, 26, 7, 3, 0).getBest()
176 .getPacakagesPlaced());
177 print(new PalletConfigRowRotate(63, 44, 8, 5, 0).getBest()
178 .getPacakagesPlaced());
179 print(new PalletConfigRowRotate(61, 35, 10, 3, 0).getBest()
180 .getPacakagesPlaced());
181 print(new PalletConfigRowRotate(61, 38, 6, 5, 0).getBest()
182 .getPacakagesPlaced());
183 print(new PalletConfigRowRotate(61, 38, 10, 3, 0).getBest()
184 .getPacakagesPlaced()));
185 print(new PalletConfigRowRotate(93, 46, 13, 4, 0).getBest()
186 .getPacakagesPlaced());
187 print(new PalletConfigRowRotate(141, 71, 13, 8, 0).getBest()
188 .getPacakagesPlaced());
189 print(new PalletConfigRowRotate(106, 59, 13, 5, 0).getBest()
190 .getPacakagesPlaced());
191 print(new PalletConfigRowRotate(74, 46, 7, 5, 0).getBest()
192 .getPacakagesPlaced());
193 print(new PalletConfigRowRotate(86, 52, 9, 5, 0).getBest()
194 .getPacakagesPlaced()));
195 print(new PalletConfigRowRotate(108, 65, 10, 7, 0).getBest()
196 .getPacakagesPlaced()));
197
System.out.println("*** END ***");
198 }
199
200 private static void print(int message) {
201 System.out.println(message);
202 }
203
}
A.4 Max Synchronization

Listing A.17: Semaphore.java

```java
import com.cycling74.max.*;

/** *
 * @description A Max object for synchronization
 * @class Semaphore − Checks that both left and right input has been banged
 * before output is banged.
 * @author Sigurd Salvesen, sigurdsa@ifi.uio.no.
 */

public class Semaphore extends MaxObject {
    private boolean leftInput = false;
    private boolean rightInput = false;

    private static final String[] INLET_ASSIST = new String[] { "inlet 1 help" };
    private static final String[] OUTLET_ASSIST = new String[] { "outlet 1 help" };

    public Semaphore(boolean bangLeft, boolean bangRight, boolean allFalse) {
        declareInlets(new int[] { DataTypes.ALL, DataTypes.ALL, DataTypes.ALL });
        declareOutlets(new int[] { DataTypes.ALL });

        setInletAssist(new String[] { "leftInput", "rightInput", "set all FALSE" });
        setOutletAssist(new String[] { "active" });
    }

    public void bang() {
        int inlet_num = getInlet();
        post("Semaphore: banged at: "+ inlet_num);

        if (inlet_num == 0) { // leftBang
            leftInput = true;
        } else if (inlet_num == 1) { // rightBang
            rightInput = true;
        } else if (inlet_num == 2) { // allFalse
            leftInput = rightInput = false;
        }

        if (leftInput && rightInput) {
            post("Semaphore: leftInput and rightInout TRUE");
            leftInput = rightInput = false;
            outlet(0, "bang");
        }
    }
}
```

Listing A.17: Semaphore.java
A.5 Max Patchers

An autonomous and vendor independent robot-based pallet loading system

robot-based_pallet_loading_system: Main window of the robot-based pallet loading application giving an overview of all the states and camera settings.

Figure A.1: Max patch: robotbased_pallet_loading_system
state_controller: Activates the different states in correct order and handles concurrency.

Figure A.2: Max patch: state_controller

semaphore: Handles concurrency to ensure that both the vision and robot has finished and are ready to pick the next package from the production line.

Figure A.3: Max patch: semaphore
Figure A.4: Max patch: semaphore2
Figure A.5: Max patch: camera_control
Figure A.6: Max patch: grab_above

Figure A.7: Max patch: video_preview_above
Figure A.8: Max patch: grab_front

Figure A.9: Max patch: video_preview_front
Figure A.10: Max patch: vision_detect_package

Figure A.11: Max patch: visualizer
Figure A.12: Max patch: vision_measure_depth
Figure A.13: Max patch: video_preview

Figure A.14: Max patch: large_view
Figure A.15: Max patch: vision_measure_width_height
Figure A.16: Max patch: video_preview2

Figure A.17: Max patch: large_view2
calc_height: Calculates the absolute height of the package.

Figure A.18: Max patch: calc_height

calc_width: Calculates the absolute width of the package.

Figure A.19: Max patch: calc_width

calc_depth: Calculates the absolute depth of the package using the absolute width and aspect ratio between width and depth.

Figure A.20: Max patch: calc_depth

134
Figure A.21: Max patch: calc_placement
Figure A.22: Max patch: package_find_placement
Figure A.23: Max patch: simulation_add_packages

Figure A.24: Max patch: add_package_group

Figure A.25: Max patch: add_package
Figure A.26: Max patch: random_numbers

Figure A.27: Max patch: activate_input
Figure A.28: Max patch: add_package_settings

Figure A.29: Max patch: empty
Figure A.30: Max patch: simulation_pallet
Figure A.31: Max patch: robot_pick_package
Figure A.32: Max patch: x1

Figure A.33: Max patch: y1

Figure A.34: Max patch: z1
Figure A.35: Max patch: rotation1

Figure A.36: Max patch: z2
robot_place_package. Place the current package at a given placement at the pallet.

Figure A.37: Max patch: robot_place_package
Figure A.38: Max patch: x3

Figure A.39: Max patch: y3

Figure A.40: Max patch: z3
Figure A.41: Max patch: rotation3

Figure A.42: Max patch: z4
A.6 JavaScripts

Listing A.18: findbox_2.js

```javascript
// global variables
var tmpmatrix;
var height = 0;
var depth = 0;

// inlets and outlets
inlets = 1;
outlets = 4;

function jit_matrix(inname)
{
    var matrix_width = inname.arrayLength;
    var matrix_height = inname[0].arrayLength;

    // adapt to input
    tmpmatrix = new JitterMatrix(4, "char", matrix_width, matrix_height);
    tmpmatrix.setInfo(inname);
    tmpmatrix.fromMatrix(inname);

    /*
     * Plane 1: left coordinate
     * Plane 2: top coordinate
     * Plane 3: right coordinate
     * Plane 4: Bottom coordinate
     */
    var array = tmpmatrix.getcell(0);
    width = (array[3] - array[1]);
    depth = (array[2] - array[0]);
    depth_vs_width = depth / width;

    outlet(0, "bang");
    outlet(1, width);
    outlet(2, depth);
    outlet(3, depth_vs_width);
}
```

Listing A.19: findbox_1.js

```javascript
// global variables
var tmpmatrix;
var height = 0;
var width = 0;
var position = 0;

// inlets and outlets
inlets = 1;
outlets = 4;

function jit_matrix(inname)
{
    var matrix_width = inname.arrayLength;
    var matrix_height = inname[0].arrayLength;
```
// adapt to input
tmpmatrix = new JitterMatrix(4, "char", matrix_width, matrix_height);
tmpmatrix.setinfo(inname);
tmpmatrix.frommatrix(inname);

/*
 * Plane 1: left coordinate
 * Plane 2: top coordinate
 * Plane 3: right coordinate
 * Plane 4: Bottom coordinate
 */

var array = tmpmatrix.getcell(0);
height = 480 − array[1];
width = (array[2] − array[0]);
position = 320 − (array[0] + (width/2));

outlet(0, "bang");
outlet(1, height);
outlet(2, width);
outlet(3, position);

Listing A.19: findbox_1.js


