MarineBlue: A Low-cost Chess Robot.

Conference Paper · January 2003

Source: DBLP

CITATIONS
22

READS
1,651

2 authors, including:

Yolande Berbers
KU Leuven
45 PUBLICATIONS **557** CITATIONS

SEE PROFILE

MarineBlue: A Low-Cost Chess Robot

David URTING and Yolande BERBERS
{David.Urting, Yolande.Berbers}@cs.kuleuven.ac.be
KULeuven, Department of Computer Science
Celestijnenlaan 200A, B-3001 LEUVEN
Belgium

ABSTRACT

This paper describes the development of a chess-playing robot called MarineBlue. This robot consists of three components: a computer vision component to recognize chess board situations, a chess engine component to compute new moves and a robot control component to execute these moves by means of a robot arm. In the paper, we focus on the algorithms that have been used to implement the computer vision and robot control components. The MarineBlue robot is fully autonomous, in the sense that it can recognize the moves done by a user, calculate a move in response to the user's move and control a robot arm to perform this calculated move. The robot that was used to develop MarineBlue is a low-cost, educational robot, which results in a cost-effective and compact chess-playing robot.

KEY WORDS: Multimedia Robotics, Computer Vision, Chess Robot, Entertainment and Games

1. INTRODUCTION

This paper describes the development of a chess robot – MarineBlue[1] – that consists of a robot arm and a digital camera. The main goal was to develop a compact and low-cost robot for educational and entertainment purposes. This paper will focus on the robotic and vision parts of the chess robot and the integration between both. The chess algorithm principles will not be discussed in this paper.

In past centuries, many people have been fascinated by the idea of constructing an autonomous chess-playing machine. Researchers in the field of artificial intelligence have been working extensively on chess algorithms during the past decades: in 1950 Alan Turing wrote the first chess program, making use of simplified rules because of the limited available computing power. In 1957 the first fully functional chess program was developed, though it was initially too weak to win against a human opponent. It took until 1996 – when IBM's Deep Blue defeated chess master Garry Kasparov – before a chess algorithm was considered better than a human chess player.

In addition to the algorithmic attention the chess game received from the AI world, there were also people working on the development of mechanical chess playing machines. The first construction was the 'Turk' made in 1769 by Wolfgang von Kempelen, which gave the illusion of being autonomous, (though it was in fact operated by a human). With the advent of the computer, it became possible to build autonomous chess robots. However, most chess robots are either expensive and large, or else they are limited in their freedom of movement and functionality.

2. MARINEBLUE COMPONENTS

This section describes the components that were used for the construction of the chess robot. MarineBlue consists of 4 essential parts: the chessboard and chess pieces, the camera, the robot arm and the chess computer. The robot arm is a Robix RC-6 [1], which is a small and configurable robot that is used for entertainment and education purposes.

2.1. CHESSBOARD AND PIECES

To maximize the usability of MarineBlue, it is obviously best to use a common chessboard with standard wooden cylindric chess pieces.

As for the pieces, it was possible to adhere to this constraint. At first, the possibility of uniformizing all the chess pieces such that it would be easier to move them was considered. Finally, however, it seemed that a modification of the gripper would be the best possible solution.

The chessboard did have to be modified, however. A standard board has square sections with a side length of between 40 and 50 millimeters. This means that the total length of one side of the chessboard is at minimum 32 centimeters and at maximum 40 centimeters. The robot arm's range is insufficient for such a board. In order to solve this problem, the length of a square section was

[1] This paper is based on work done within the scope of Stijn Debruyckere's Master's thesis (2002).

reduced to 30 millimeters, in combination with a small extension of the robot arm.

A second board modification concerns the color of the square sections. Since the vision algorithm has to detect which sections are occupied and which are not, the colors of the square sections and the chess pieces play an important role. To guarantee a robust algorithm, disjunctive colors had to be chosen.

2.2. CAMERA

The camera that is used is a Sony DFW-VL500 [2], a high-quality camera. This is important since the quality of recorded images determines to a great extent the quality of vision applications. The camera was mounted 1 meter above the chessboard in order to minimize the perspective effects.

The camera complies with the IEEE 1394 [3] standard and sends its images uncompressed to the chess computer, where they are processed. Since a continuous flow of images is not necessary for this chess application, the camera is used in single frame mode. The interval between two frame shots is chosen small enough to keep the response time low. These frames are then analyzed in order to detect board situation changes.

2.3. ROBOT

The robot that is used is a Robix RC-6 [3], which is a small and configurable robot used for entertainment and education purposes. This robot is not a top performer in terms of precision and power, but the ability to be easily modified and its low price make this robot an excellent choice.

The Robix robot consists of a set of separate segments that can be attached to one another. These segments can move with respect to each other by means of servos. In addition, a gripper can be attached to the last segment. The servos are controlled by means of a controller, which is attached to the parallel port of the chess computer. Servo commands are sent in a Robix-dependent scripting language.

2.3.1. RADIUS

One of the problems experienced with the robot arm was its limited radius, which results in the fact that not all corners of the board could be reached. This problem has been solved, as mentioned earlier, by creating a smaller chessboard and extending some of the robot's segments. However, extending the segments also resulted in inaccuracies at the extreme limits of its reach. This issue has been dealt with by using a modified gripper (see section 2.3.2).

Three segments are used to move the arm in parallel to the board, and a fourth segment moves vertically on the third segment. The gripper has been attached to this fourth segment. A side view of this configuration is shown in Figure 1. A view from above is presented in Figure 2. Actual lengths are in millimeters.

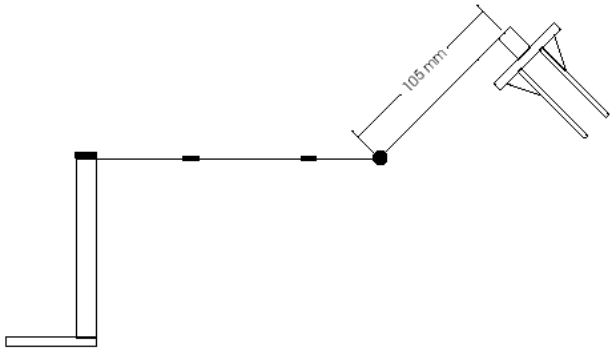

Figure 1: Side view of robot configuration

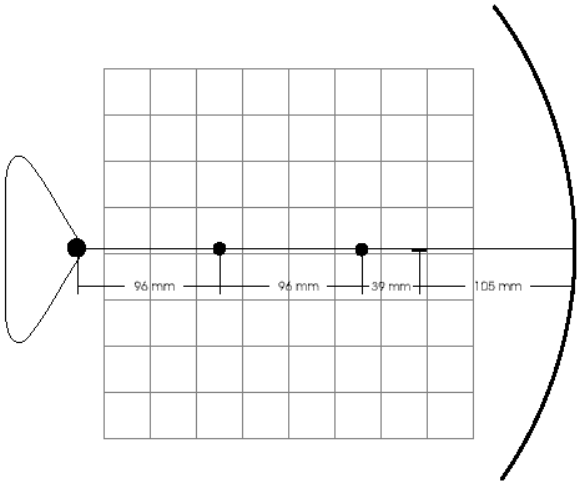

Figure 2: View of robot configuration from above

2.3.2. GRIPPER

To increase the accuracy of the robot arm (and to reduce possible swing problems) the gripper was made lighter by removing one degree of freedom: due to the cylindric form of the chess pieces, it was possible to remove the servo responsible for rotating the gripper.

The form of the gripper was also modified for this particular chess application. Firstly, the gripper was made longer since otherwise it would collide with other chess pieces when picking up a particular chess piece. Secondly, a hemispherical form was used in such a manner that it corrects small positioning errors.

There are other alternatives for improving the accuracy of the robot: the chessboard could be made much smaller than it is now, or we could switch to a more accurate – and expensive – robot, or we could use some feedback mechanism from the camera to the robot controller.

2.4. CHESS COMPUTER

The entire application that analyses the board situation, computes the moves and executes them, runs on a standard Windows 2000 computer with an IEEE 1394 adapter. The software was developed in C/C++ using MS Visual Studio.

3. ALGORITHMS

This section describes the algorithms, methods and techniques that were used for the development of the software. As was mentioned earlier, this application covers three important fields of computer science, in particular: computer vision, artificial intelligence and robotics. The important algorithms for the first and the third domains are investigated in more detail in the next subsections.

3.1. DETECTION OF GAME SITUATION

An important part of the application is the analysis of the chessboard by means of images that come from the camera. Firstly, it is necessary to detect the position and orientation of the chessboard, and secondly it must be possible to determine for each square section if there is a chess piece on it and what kind of piece it is.

This functionality has been subdivided in three layers:

- The *pixel classification layer* extracts features from images captured by the camera.
- The *board layer* determines the position of a board and the pieces on it by means of the features that were detected in the previous layer.
- The *chessboard layer* determines the current chess game situation and gets its input from the board layer.

3.1.1. PIXEL CLASSIFICATION LAYER

As input this layer receives an RGB (Red, Green, Blue) bitmap from the underlying camera driver. The necessary features extracted from this image can then be used by the next layer to recognize the board and the pieces.

Three different alternatives (see [4]) were investigated: edge detection, template matching and pixel classification. Ultimately pixel classification was chosen since it is a simple and robust method for recognizing board situations. Pixel classification attempts to determine the class to which each pixel belongs by means of the (color) attributes of these pixels. Before being able to classify pixels it is necessary to perform a calibration step that results in the computation of the color domain for each class. After this calibration step, one is able to classify all pixels in the image.

Four classes were defined: light square, dark square, light piece and dark piece. When using the RGB color space it seemed that the classes ‘light square‘ and ‘light piece’ did overlap and as a consequence it was not possible to perform an accurate pixel classification. An example of this can be seen on the left chessboard in Figure 3. To solve this problem, we switched to a HSB (Hue, Saturation, Brightness) color space, which solved the overlap problems. This is shown on the right chessboard in Figure 3.

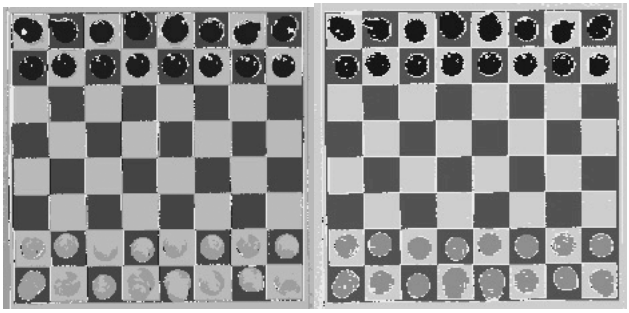

Figure 3: The figure on the left is the result of RGB pixel classification. Some pixels of the light pieces were classified as belonging to a light square. The figure on the right is the result of HSB classification. Most pixels have been classified correctly.

3.1.2. BOARD LAYER

The input of this layer is a matrix that states for each pixel to which class it belongs. From this information it will be attempted (1) to position the board and pieces in the image and (2) to determine which squares are occupied and which squares are not.

Concerning the first functionality, we use a reference frame having the same color as the dark squares. Firstly, the algorithm performs a search for the corners of this reference frame by searching at the extremities of the image for pixels belonging to the ‘dark square’ class. Next, the position of each square is computed by means of interpolation. This is relatively simple since all squares are equal in size and there are 8 squares for each dimension. The results of this computation are shown in Figure 4.

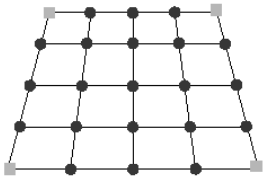

Figure 4: Position of the board and the squares

Next, the algorithm determines (1) which squares are occupied by pieces and (2) the color of each piece. This can be done by calculating the number of pixels belonging

to the class 'light piece' and the class 'dark piece'. If this number exceeds a particular threshold, then the algorithm decides that there is a piece on the square. The threshold value is dependent on the surface area that a piece occupies in proportion to the surface area of a square.

3.1.3. CHESS BOARD LAYER

As input, this layer gets a list of all squares that are occupied by a chess piece of a particular color. This layer also *remembers* the previous situation of the chessboard (which can also be regarded as input). By means of this information, the algorithm will then determine the exact position of each chess piece in the new board situation.

Since the camera is mounted above the chessboard and each chess piece has a circular form seen from this point of view, it is seemingly impossible to determine the type of each chess piece from only this information. This problem can be solved quite simply by retrieving the situation of the board before the last move was done. By means of this information (in which the exact position and type of each chess piece before the move is known) and the new information (in which is known which squares are occupied after the move), it is always possible to determine the chess piece that has been moved. The initial board situation is known when the game starts – since the chess rules state how the board must be set up – so there is no need for a configuration step before the game starts.

The algorithm can detect all valid chess moves, even the castling move (which involves a displacement of the king and the rook), given that the user adheres to the chess playing rules. Some cheating actions, such as swapping two chess pieces of the same color would go unnoticed by this algorithm, since it will not detect any changes on the chessboard in that case. The reason for this is that the *board layer* only recognizes the color of a piece, not its type.

3.2. CHESS ALGORITHM

We did not develop this part of the application. Instead an existing chess algorithm implementation was used, more specifically the GNU GPL Chessterfield implementation (see [5]). Alternative implementations can be integrated relatively simply since a generic interface is used to communicate with the algorithm.

3.3. ROBOT CONTROL

The most important problems that were encountered when developing the MarineBlue application were related to the robot control part. These problems were especially due to the limited capabilities of the hardware.

This section will focus on the algorithms that are used to control the robot arm. The functionality that was aimed at was the development of an algorithm that translates high-level commands (*move the piece from board coordinate (2,b) to board coordinate (2, c)*) to low-level commands for the robot's servos.

3.3.1. KINEMATICS LAYER

This subsection focuses on the inverse kinematics part, which calculates the different angles of the servos given a particular configuration of the robot and a requested position of the gripper. In order to set up this inverse kinematics algorithm, certain simplifications are carried out, which are removed later without many additional computations.

The simplification that is mentioned above concerns segments 3 and 4. Initially it was said that segment 4 is attached to the gripper and that it can turn vertically on segment 3. Segments 1, 2 and 3 move in a plane that is horizontal with the chessboard. We suppose now that the angle between segments 3 and 4 is always zero, such that it is now possible to consider a new extended segment '34'. The total length of this segment is the sum of the lengths of the constituting segments.

Figure 5 illustrates what needs to be computed: given a requested position (250, 70)[2], the angles θ_1, θ_2 and θ_3 need to be known. The position of the gripper is also determined by the angle γ between the last segment and the horizontal axis. Since a standard chess piece is cylindrical, this angle will have no significance in terms of the ability of the gripper to pick up the piece. However, this angle will remain in the calculations that are described in what follows.

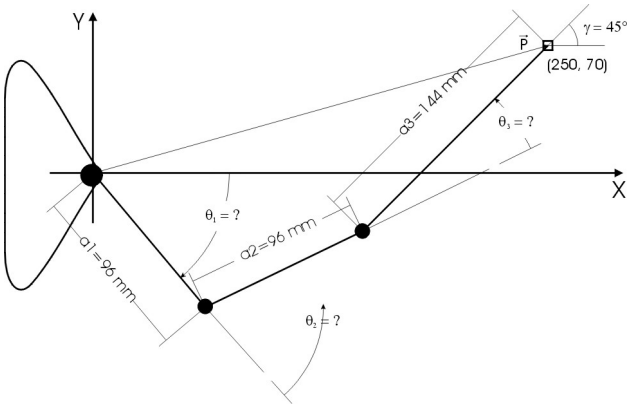

Figure 5: Inverse kinematics (view from above)

The solution of inverse kinematics problems is generally quite complicated and computing-power-intensive. For instance, it is not possible to solve this problem analytically for six or more segments. For most robot arms it is however possible to make major simplifications such that it can be solved analytically.

[2] These are 'robot' coordinates. Translation from 'chess board' coordinates to 'robot' coordinates is done at a higher level.

As a result of the assumption made above that segment 4 is always in line with segment 3, the calculations are simplified to calculations made in a plane parallel to the chessboard.

The formulas from which one can start[3] are:

$$3.1: \quad \gamma = \theta_1 + \theta_2 + \theta_3$$
$$P = a1 + a2 + a3$$

The second formula can be rewritten as formula 3.2:

$$x = a_1c_1 + a_2c_{12} + a_3c_{123}$$
$$y = a_1s_1 + a_2s_{12} + a_3s_{123}$$

with

$$c_1 = \cos(\theta_1) \qquad s_1 = \sin(\theta_1)$$
$$c_{12} = \cos(\theta_1 + \theta_2) \qquad s_{12} = \sin(\theta_1 + \theta_2)$$
$$c_{123} = \cos(\theta_1 + \theta_2 + \theta_3) \qquad s_{123} = \sin(\theta_1 + \theta_1 + \theta_1)$$

Given the requested gripper position (x,y,γ), the intent is now to solve the above equations for the three unknowns θ_1, θ_2 and θ_3.

Substitution of 3.1 in 3.2 gives formula 3.3:

$$x - a_3c_\gamma = a_1c_1 + a_2c_{12}$$
$$y - a3s_\gamma = a_1s_1 + a_2s_{12}$$

At this point, two unknowns (θ_1 and θ_2) remain. Squaring and summing both equations eliminates θ_1:

$$(x - a_3c_\gamma)^2 + (y - a_3s_\gamma)^2 = a_1^2 + a_2^2 + 2a_1a_2c_2$$

This results in two solutions for θ_2:

$$\theta_2 = +/- \arctan(s_2/c_2)$$

To determine θ_1 one can substitute θ_2 in 3.3. This also results in two solutions (depending on the actual θ_2 that is substituted):

$$\theta_1 = \arctan(s_1/c_1)$$

Finally, by using 3.1 it is also possible to determine θ_3:

$$\theta_3 = \gamma - \theta_1 - \theta_2$$

If the requested position (x,y,γ) can be reached by the robot arm, then there exist two possibilities for reaching that position. Given that the angle γ is of no importance, it can be stated that there are an infinite number of solutions. The method that is used here is to increase the angle γ with small increments (e.g. increments of 5 degrees) and to calculate the two solutions corresponding to this γ, if these two solutions exist.

When all solutions are computed, the *best* one is chosen. Typically one chooses the solution that incurs the least possible movement of the arm. The disadvantage of this approach is that the angles between the segments of the arm become dependent on the previous movement(s). This means that a particular square can be reached with the arm having different sets of servo positions and, as a consequence, the deviation of the gripper will also vary. To avoid this problem, MarineBlue will select the set of servo positions that is closest to the ideal position of the arm. The ideal position is the position in which each servo is positioned in the middle of its reach. This strategy ensures that the servos will rarely take up their extreme positions for reaching a particular square.

Until now it has been supposed that the fourth segment was an extension of the third segment, such that it always moves in a horizontal plane. However, when a chess piece is to be picked up, the fourth segment has to move vertically and thus the initial assumption is not valid anymore. Figure 6 illustrates that it is not only the fourth segment that has to move, but rather that the origin of this fourth segment also has to move in order to keep the center of the gripper above the piece. This is necessary; otherwise the gripper could collide with the chess piece (or adjacent chess pieces.)

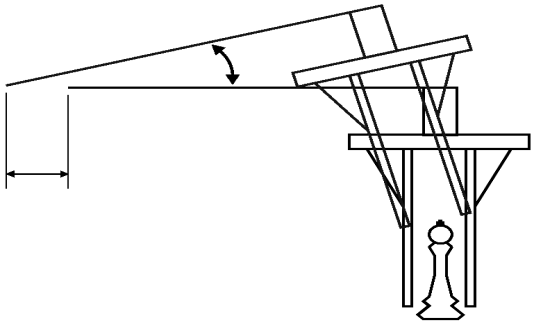

Figure 6: Horizontal displacement of the fourth segment when moving downwards and upwards

As a result, we can no longer assume that the length of the 34 segment equals the sum of the real lengths of segments 3 and 4. This length needs to be computed. From Figure 7 it becomes clear that this length can be derived from the following formula:

$$l = l_3 + l_4 .\cos(\alpha) + h.\sin(\alpha)$$

The length of segment 34 is thus always recalculated. This way the simplification – for computing the inverse kinematics – can be retained.

To pick up a piece, the gripper will move downwards in an opened state. The origin of the fourth segment moves to the left (of the figure) at the same time. The gripper

[3] This derivation is based on [6].

closes and clasps the chess piece. Next, the gripper moves upwards and the origin of the segment moves to the right of the figure. This is shown in Figure 6 as the horizontal displacement. This horizontal displacement when moving upwards and downwards ensures that the center of the gripper is kept right above the chess piece.

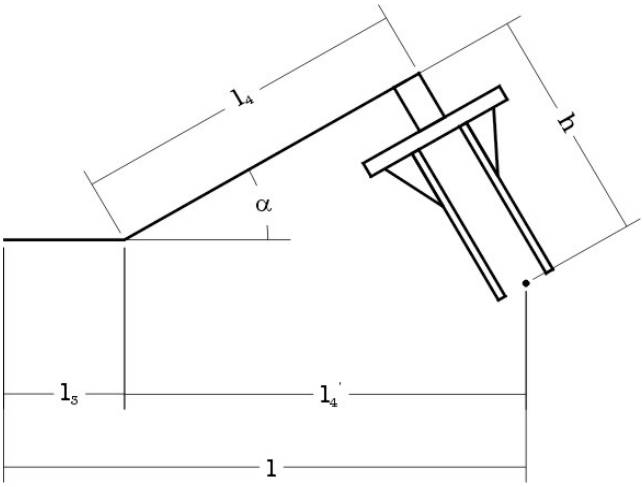

Figure 7: The computed length (l) of segment 34.

3.3.2. COMMAND LAYER

The *command layer* is situated right above the *inverse kinematics layer*. It is responsible for transforming high-level robot command sequences (such as 'open gripper', 'close gripper' and 'move to') into Robix script commands. This layer makes use of the inverse kinematics in order to transform chessboard coordinates into angles for controlling the servos.

3.3.3. ROBOT CALIBRATION

The calibration of the robot arm is done in three parts: (1) the length of the segments is measured, (2) for each servo a mapping is made between angles (degrees) and corresponding servo positions, and (3) the robot arm has to know the location of the chess board in robot coordinates. To perform this last calibration step, the robot arm is 'trained'[4]: the user indicates the four corners of the chessboard by moving the robot arm to these corners. The position of each square can then be easily determined by means of interpolation.

4. SOFTWARE ARCHITECTURE

The high-level architecture of the MarineBlue software consists of three algorithmic modules (vision, chess engine and robot control), an application layer and the GUI. Figure 8 presents a schematic overview of this architecture.

[4] An alternative approach would be the provision of feedback from the camera to the robot arm.

The *application layer* is responsible for controlling the entire chess game, which amounts to (1) the capture and analysis of an image, (2) the computation of a new move (if the human opponent has performed a move) and, finally, (3) the transformation of the computer's move into high-level instructions for the robot arm. This process repeats itself until the game is over.

The *GUI layer* is a Windows front-end for MarineBlue, which contains the necessary functionality for calibrating the vision and robot modules.

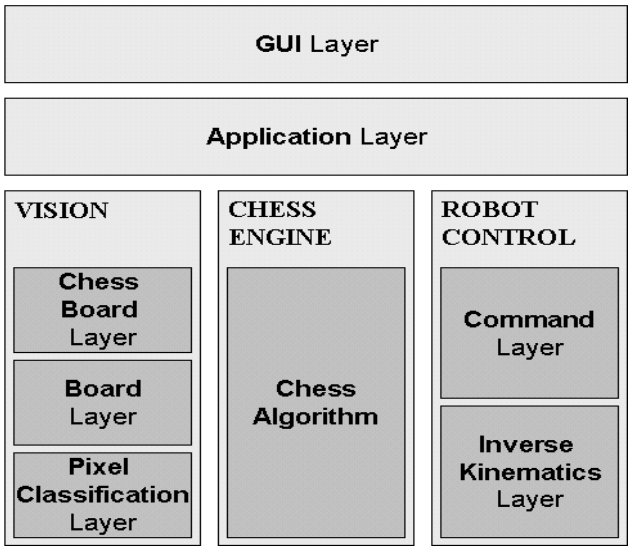

Figure 8: MarineBlue's software architecture

5. CONCLUSION

This paper gives an overview of the high-level design and implementation of an autonomous chess robot, which consists of three main functionalities: recognizing chess board situations, computing new moves and executing these moves by means of a robot arm.

The chess robot can be used for educational and recreational purposes, because it has been built from relatively simple components. Since the basic software architecture consists of three independent modules, it is relatively easy to change the algorithms or hardware-dependent modules.

REFERENCES

[1] Robix RCS-6, http://www.robix.com
[2] Sony DFW-VL500, http://www.sony.com
[3] FireWire IEEE 1394, http://www.apple.com/firewire/
[4] Sergios Theodoridis, Konstantinos Koutroumbas, "Pattern Recognition", Academic press San Diego, 1999
[5] Matthias Lüscher, GNU Chessterfield
[6] John J. Craig, "Introduction to Robotics, Mechanics and Control", Addison-Wesley, 1986