Endgames: Task 2 & Task 3

Jeroen Serdijn & Michiel Folkers

June 22, 2012

1 Introduction

1.1 Abstract

The configuration of a chessboard permits such variation that it is impossible for practical purposes to navigate each state as a unique case. Therefore methods of generalization must be applied to the game in order to navigate both the tactical and spatial problems that are presented.

This report is concerned with the application of robotics and computer software in order to move and take pieces on the board.

1.2 Challenges

In order for a robotic arm to move a piece from one point to another without human guidance the problem space must be described in general terms and heuristics must be applied in some form to advance an arbitrary move. Path planning and joint calculations are necessary for the robotic arm to move correctly without disturbing the other pieces, especially when taking a low path. In the case that a low path is no longer possible the arm must take a high path instead. Furthermore when an opponent piece is taken it must be moved off the board before the attacking piece is placed in its square.

The rules of chess are not important in this problem. The spatial parameters of a chess board however are critical to developing a solution. The algorithms must obtain the correct information describing the physical representation. Abstract representations of the game configuration must then be converted correctly into real world coordinates, angles, and measurements relative to the robotic arm, chess pieces, and the chess board.

2 Path Planning

The problem of Path Planning has many solutions which each have their own merits. The choice between implementing one method or another is

influenced both by the efficiency of the resultant algorithm but also by the constraints of the programmer.

Greedy, Best First Search and A* are obvious no nonsense choices for algorithm simplicity, however there is an impact to efficiency. When tested a naïve implementation of a greedy A* algorithm became too greedy and required more programming attention to debug than was available. The heuristic function consistently chose for the shortest path distance and quickly became entrapped between pieces.

For this reason the DistanceMatrix class was utilized to produce an optimal path and interface for navigating the path structure. However after considerable testing it was found that the DistanceMatrix class also had a similar short coming. A piece itself is not considered impassable, and when a piece is ringed with adjacent pieces the algorithm ignores the certain collisions from reaching a path space.

It was therefore decided to combine aspects of the DistanceMatrix with the previous algorithm to produce an optimal path, only in the case that this is in fact possible. The Distance Matrix has also been improved to report an encircled formation as an unreachable path node.
After much testing the Path Planning now works as intended and simply reverts to a safe height path when no completed low path can be constructed.

2.1 Low & highpath

If it is possible to navigate an piece on the board without touching other pieces, the low path method will be initiated. Using the A*-like algorithm, the program is able to calculate the best route to move a chess piece to another destination. The robot arm will navigate the chess piece between the other chess pieces without disturbing them.

If low path is not an option, high path will be initiated, the robot arm will use the coordinates obtained from StudentBoardTrans class and the joint angles from inverse kinematics, and move the chess piece at a safe height above the board.

2.2 Garbage collection

When the piece movement involves another chess piece at the final position, garbage collection is necessary. The method will check whether there is a chess piece at the end position, and will remove it before starting high or low path. The robot arm will use high path planning to obtain this chess piece, and safely remove the chess piece from the board by placing it in a

container on the right side of the chess board.

2.3 Coordinates

In order to convert these default moves, we need to know some default parameters of the board. To calculate the z-axis, the board thickness and the piece height must be known. In order to convert E3 to x and y coordinates, it is necessary to know the length and width of one square of the board, as well as the x and y values of square H8. These parameters give us enough information to convert an chess move to useful coordinates for the robot arm. The theta value of the board is used to represent the angle or the rotation of the board, the coordinates are then transformed with an internal rotation matrix.

3 Inverse Kinematics

In order to calculate the positions and the angles of the various joints in the robot arm, we first need to understand how to calculate the successive angles from each joint.

The first step is to determine the constants in the robot arm. The pitch is always faced downwards, therefore the angle is minus 90 degrees. The roll is always faced forward, thus the angle is zero degrees. The gripper is either open or closed, and can easily be obtained from the GripperPosition method. The angle of the shoulder, elbow and yaw are harder to obtain, the smart vector method is needed to calculate those angles. [1]

3.1 Smart Vector Method

By using the smart vector method we are able to calculate the angles of the shoulder and the elbow. Since the yaw is supposed to be zero degrees pertaining the zed, we need to calculate the angle between the yaw and the elbow. Using both the angle of the elbow and the shoulder, we are able to calculate the angle of the yaw.

4 Actuating Path Planning

The joint angles are calculated by inverse kinematics to correspond correctly with the coordinates produced with the Path Planning algorithm. It is now possible able to actuate the planned path to move one piece from one place to another. Each of these methods to move pieces on the board are just as important as the other.

5 Testing & Improvement

While testing it became clear that the arm switching mechanism was not working as intended. The default switching mechanism was used instead.
It became clear early on in the testing phase that many redundant moves were being planned. These moves had been carefully removed and minimized so that path planning only remembers the necessary points to travel in straight lines.
Many improvements to the program remain unimplemented due to time constraints and the excessive time spent testing the program and algorithems with trial and error. One improvement was for the garbage collection so that pieces are properly placed next to the board in the correct order. This can be accomplished by matching an array of pieces in the standard start order matched with the dead pieces, and then translating to the left or right of the board to place the lost pieces parallel to the board squares.

References

[1] Leo dorst. *An Introduction to Robotics for the Computer Sciences, AI.* UvA, 2001.