BRCORP
TECHNOLOGY | INNOVATION | INDUSTRY

5th World Conference on
Applied Sciences, Engineering & Technology
02-04 June 2016, HCMUT, Vietnam

Trajectory optimization for chess-playing robot arm

THANH NHAT LUONG[1], HOANG LIEN SON CHAU[1], TRI CONG PHUNG[1], DUY ANH NGUYEN[1]
[1]Department of Mechatronics, the Faculty of Mechanical Engineering,
HCM City University of Technology, Vietnam
Email: duyanhnguyen@hcmut.edu.com

Abstract: Planning optimal trajectory is one of frequent and essential problem in controlling robot arm, especially in the condition of existing many obstacles in workspace. A chess match with human is an interesting and completely suitable issue to experiment the automatic operation of robot arm. In this paper, Dijkstra algorithm, was applied to optimize the trajectory of a SCARA which can carry out chess moves following shortest path and evade other chess pieces. Chess moves were also played and evaluated in real match with human.

Keywords: *Dijkstra Algorithm, Trajectory Optimization, Automatic Chess-Playing Robot*

Introduction:

Chess is a two-player strategy game played on chessboard. Beating grandmasters in not only classical chess match but also in rapid and blitz is the target of researchers. Therefore, it is necessary to optimize some criteria such as: trajectory, operation time for robot arm. Optimal trajectory method using Dijkstra algorithm is focused within this paper.
Firstly, a 3DOF manipulator system was established. Then, the status of chess match was determined continuously basing on images captured by camera. After that, optimal trajectory was calculated and selected. Finally, results of simulation and experiment processes are compared.

Detecting chessboard status:

Before conducting a response move, system must identify the move of opponent. A camera which was hung above chessboard is responsible for that task. Algorithm for detecting chess moves was presented in [1]. After that, chessboard status (including position and the height of chess pieces) was determined by a chessboard matrix (*Figure 1*).

	1	2	3	4	5	6	7	8
A	2	1	0	0	0	0	-1	-2
B	3	1	0	0	0	0	-1	-3
C	4	1	0	0	0	0	-1	-4
D	5	1	0	0	0	0	-1	-5
E	6	1	0	0	0	0	-1	-6
F	4	1	0	0	0	0	-1	-4
G	3	1	0	0	0	0	-1	-3
H	2	1	0	0	0	0	-1	-2

Figure 1: Chessboard matrix at beginning

Each chess piece is assigned by a number such as: Pawn: number 1, Rock: number 2, Knight: number 3, Bishop: number 4, Queen: number 5, King: number 6 and empty cells: number 0. Opponent pieces have negative values. This matrix will update after every move.

The height of each chess piece was known, thus, it is easy to establish chessboard status (*Figure 2*).

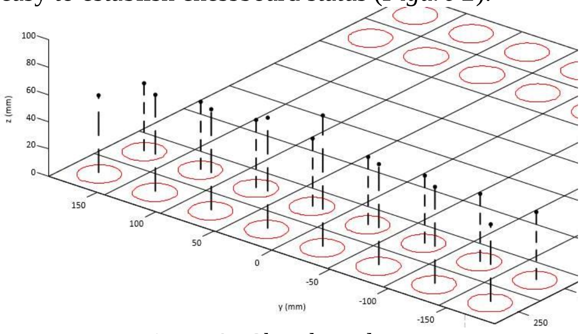


Figure 2: Chessboard status

Dijkstra algorithm in trajectory optimization:

Before applying the Dijkstra algorithm, it is necessary to define the matrix with weights for the network graph G (V, E), where V is the number of vertices in the graph and E is number of edges [2].
After network graph G was established, Dijkstra algorithm can be applied to choose shortest path [3]. Dijkstra algorithm is the popular way of calculating the shortest path, especially in calculating the shortest path from one point to all the rest points. It is the main characteristic of Dijkstra algorithm to expand from the beginning node which is the centre to the outer layers until reaching the terminal node. Dijkstra algorithm can find the best answer for solving the shortest path problem [4]. However, if there are too many nodes, it will not operate efficiently. Then, the number of edges can be considered as the length of the migration path. The elements of chess in every layer have weight shown in *Figure 3.*

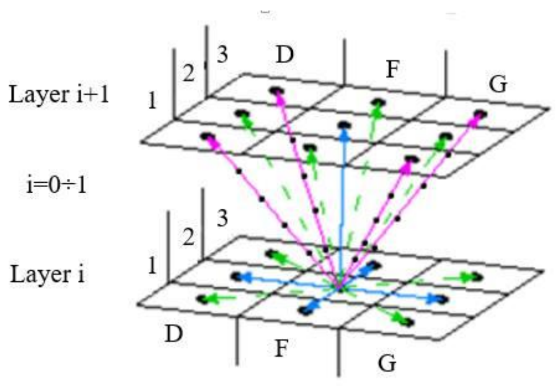
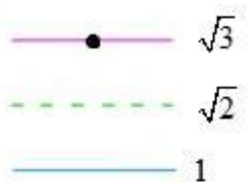


Figure 3: Weight matrix in 3D space

With:



To improve the optimizing performance of algorithms and calculate the optimal trajectory in 3D space, 3-layer board were created and all 192 nodes were linked. The distance from one node to the rest nodes on chessboard have weighted matrix as in *Figure 4*.

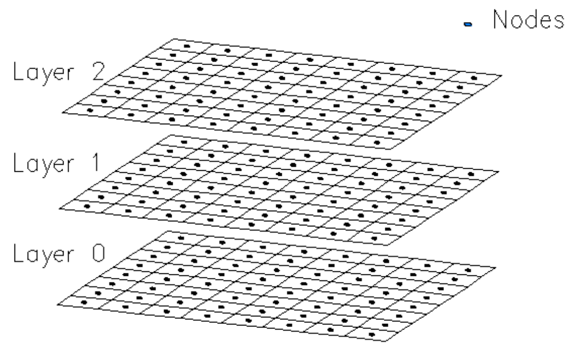


Figure 4: The nodes in 3D space

At each step, it is not necessary to check all the intermediate nodes, the order of process is as follow:

- Choose a node **u** which has smallest value d [u]
- Choose **u** is intermediate node to determine the next step

The length of the path p = e1, e2... [5]

e is calculated by equation : $w(p) = \sum_{i=1}^{k} w(e_i)$

Finding the shortest path from s to t is defined as follow:

$\min_{p \in P} w(\text{p})$ with: P is the set of all paths from s to t.

The equation below shows the idea of Dijkstra algorithm:

$$L_k(a,v) = min\left\{L_{k-1}(a,v),\ min_{(u,v)\in E}\left\{L_{k-1}(a,u) + w(u,v)\right\}\right\}$$

With:

- k: iteration
- $L_k(u,\ v)$: the shortest distance from u to v after the iteration k

When there are obstacles on the way of chess piece, the weight of that direction is infinity. Illustration of this case is shown in *Figure 5*.

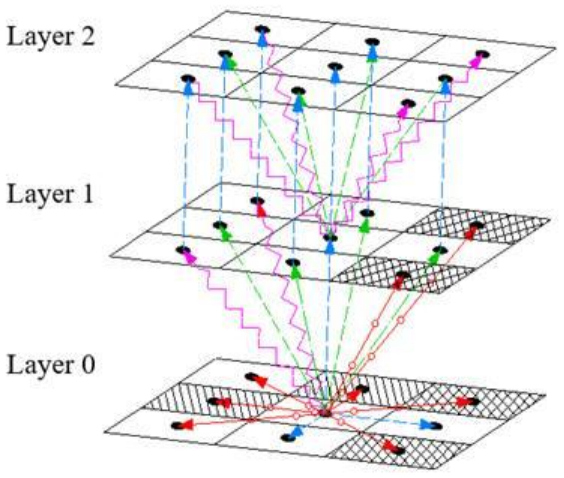
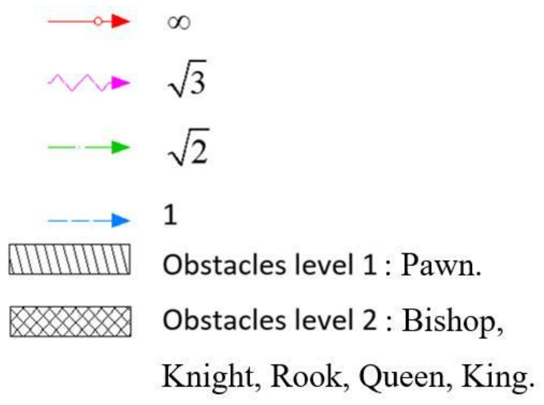


Figure 5: The weight when there are obstacles

With:


∞
√3
√2
1
Obstacles level 1 : Pawn.
Obstacles level 2 : Bishop, Knight, Rook, Queen, King.

Simulation:

Simulation in Simulink Environment

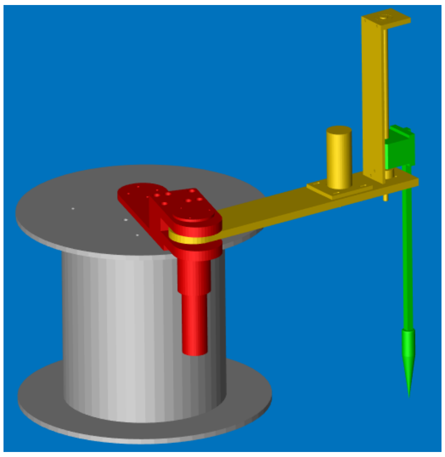


Figure 6: SCARA robot arm in Simulink environment

This section will conduct 3D-modeling and simulation under an optimization trajectory by Dijkstra's algorithm. The simulation will verify the robot kinematic calculations in previous section. Besides, this progress provides an intuitive view of the trajectory response-ability before controlling on real model. A Simulink function was applied to import the model to the Simulink environment (*Figure 6*). After that, linear control blocks and central control blocks were built and applied kinematic equations to simulate. Forward and inverse kinematic equations were presented in [1]. All kinematic link between these blocks are shown by the *Figure 7*.

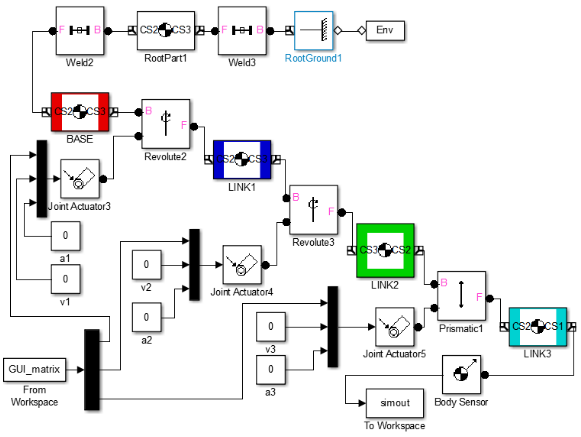


Figure 7: Block diagram of control system in Simulink environment

Simulation results:

In the first case: moving the Pawn from E2 to E4, the results are shown in *Figure 8*, *Figure 9*, *Figure 10* and *Figure 11*.

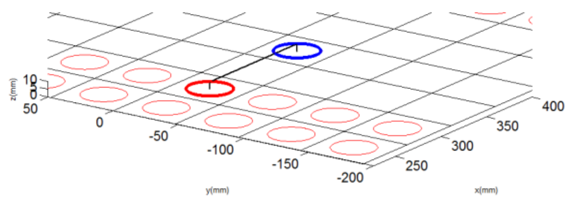


Figure 8: Simulation trajectory of robot

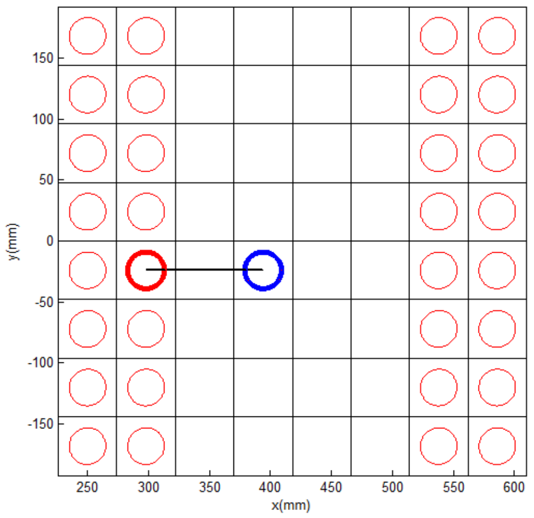


Figure 9: xy plan in simulation

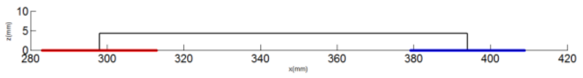


Figure 10: xz plan in simulation

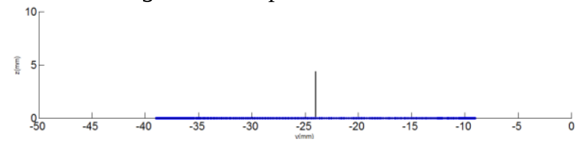


Figure 11: yz plan in simulation

In this case, motion of the Pawn was a straight line on layer 0 because there is no obstacle between D2 and D4. In addition, to avoid friction with chessboard, chess piece was lifted 5mm higher than chessboard. Total time to complete this move is 21.2s.

The second case: moving the Knight from G1 to F3, the results are shown in *Figure 12, Figure 13, Figure 14 and Figure 15*.

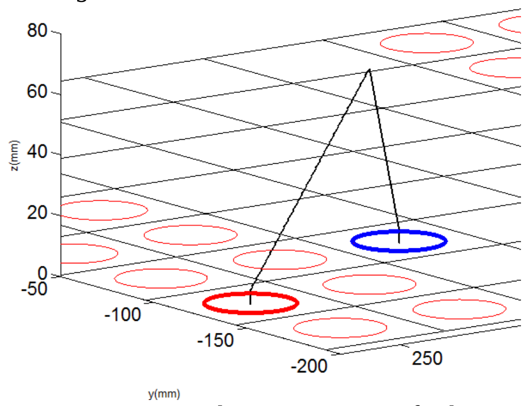


Figure 12: Simulation trajectory of robot

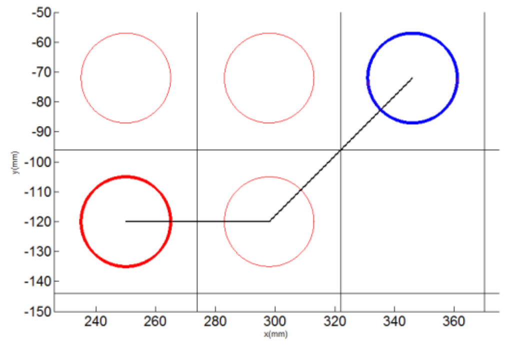


Figure 13: xy plan in simulation

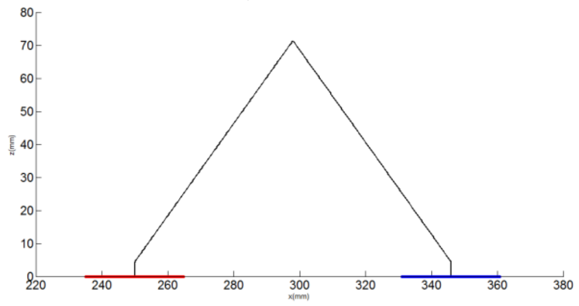


Figure 14: xz plan in simulation

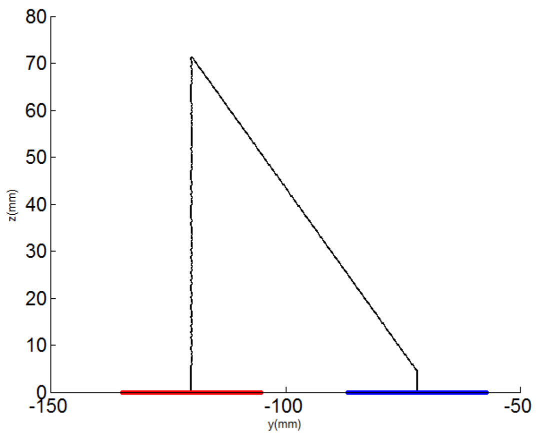


Figure 15: yz plan in simulation

In this case, there are obstacles between G1 and F3, therefore, the Knight must conduct a jump overhead the Pawn.

In xy plan, the reason why the Knight do not move directly from G1 to F3 is that there are only 64 notes (proportional to 64 chess cells), hence, it is compulsory that the trajectory must pass over G2 position. Total time to complete this move is $37.4s$.

Experimental model:

Control system

The control system in *Figure 16* was designed to control three DC Servo motors. This research built a communication program in the computer via Matlab. The starting and ending position of the chess will be inputted to the program. Dijkstra's algorithm chooses the shortest paths, then the program using the inverse kinematic equations to get the output parameters $\Delta\theta_1, \Delta\theta_2, \Delta d_3$, encodes them into a frame and transfer to the microcontroller via the RS232 standard.

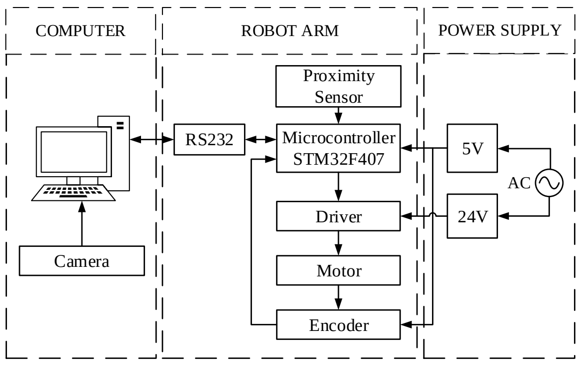


Figure 16: Control system diagram

The essence of completing a trajectory is motion control problem (point to point). Trajectory will be divided into several segments. In order to complete an element as a line, three motors of robot need to operate simultaneously and finish at the same time. That work is also known as synchronous operation of three motors.

When these analysed signals were received from the computer, the microcontroller will generate PWM based on the synchronization algorithm (Figure 17) to control the three motors for completing these displacement. Generally, pulse of each motor is scaled according to the maximum frequency (where the motor can perform). In other words, the more pulse motor has, the larger frequency it will be.

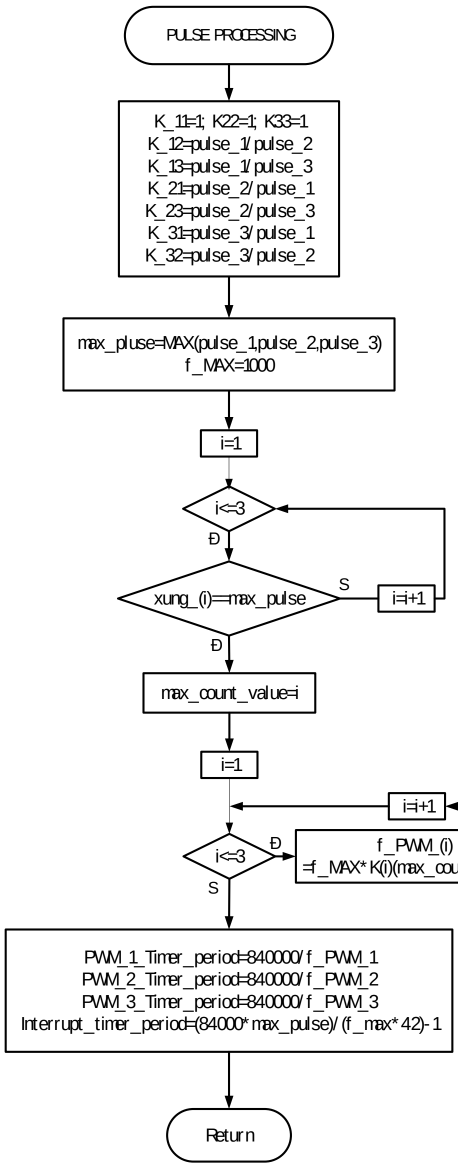


Figure 17: The synchronization algorithm

Experiment and results

After designing and simulating, the experimental model in *Figure 18* was built and installed with the control system. Besides, a graphical user interface is built by Matlab to import data, analyze and transmit these processed signal (*Figure 19*).

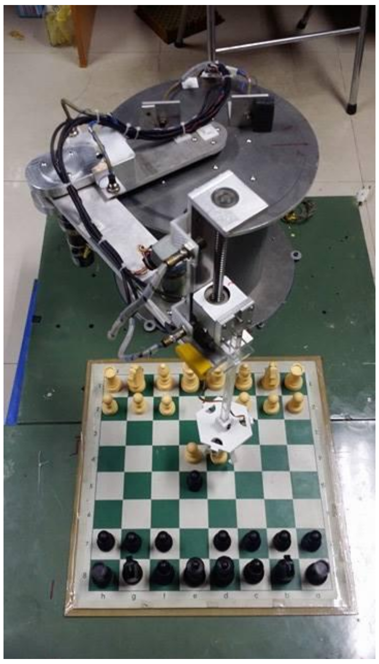
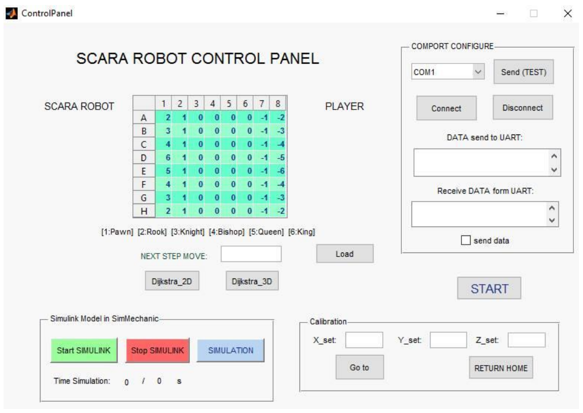


Figure 18: Experimental model



ControlPanel

SCARA ROBOT CONTROL PANEL

SCARA ROBOT

PLAYER

	1	2	3	4	5	6	7	8
A	2	1	0	0	0	0	-1	-2
B	3	1	0	0	0	0	-1	-3
C	4	1	0	0	0	0	-1	-4
D	6	1	0	0	0	0	-1	-5
E	5	1	0	0	0	0	-1	-6
F	4	1	0	0	0	0	-1	-4
G	3	1	0	0	0	0	-1	-3
H	2	1	0	0	0	0	-1	-2

[1:Pawn] [2:Rook] [3:Knight] [4:Bishop] [5:Queen] [6:King]

NEXT STEP MOVE:

COMPORT CONFIGURE: COM1

DATA send to UART:

Receive DATA form UART:

☐ send data

Simulink Model in SimMechanic

Time Simulation: 0 / 0 s

Calibration

X_set: Y_set: Z_set:

Figure 19: The main control interface

The obtained results:
To evaluate the operation of robot arm, pulse of three encoder was responded and calculate by forward kinematic equation. Position of end-effector will be compared with simulation results.
For the first case: moving the Knight from cell E2 to E4, the results are shown in *Figure 20, Figure 21, Figure 22* and *Figure 23*.

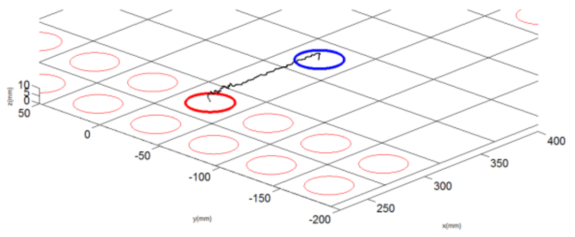


Figure 20: Real trajectory of robot

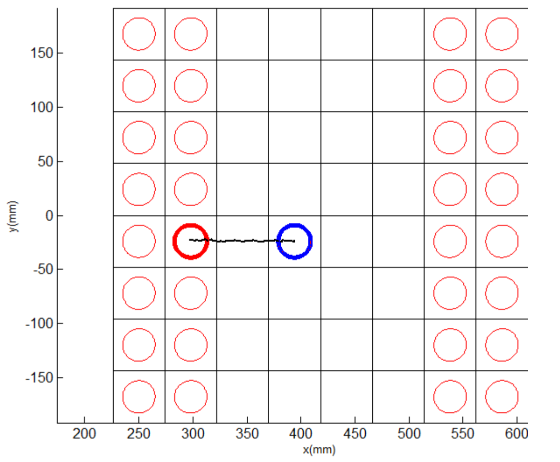


Figure 21: xy plan

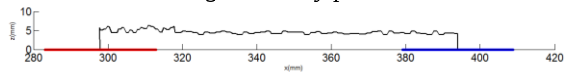


Figure 22: xz plan

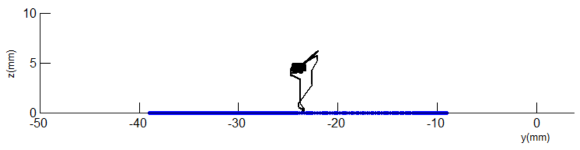


Figure 23: yz plan

The second case: moving the Knight from cell G1 to F3, the results are shown in *Figure 24, Figure 25, Figure 26 and Figure 27.*

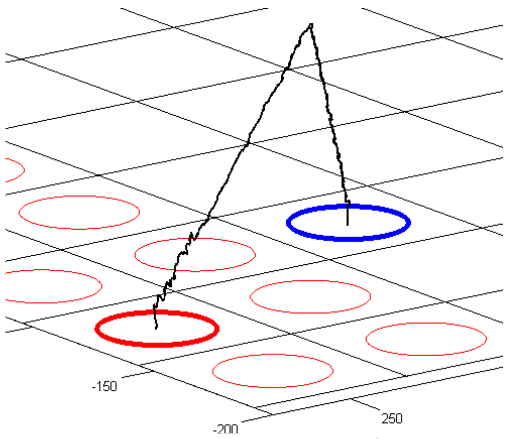


Figure 24: Real trajectory of robot

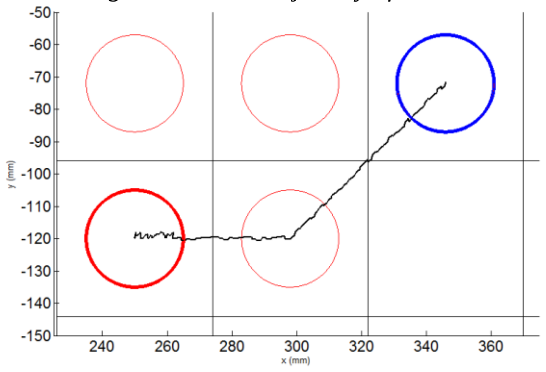


Figure 25: xy plan

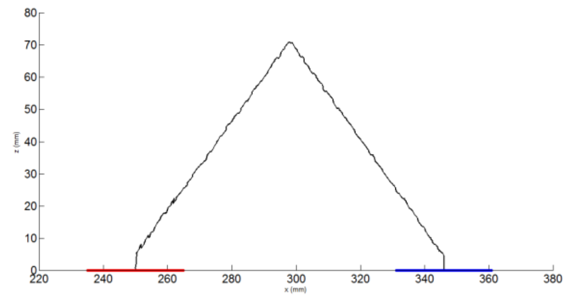


Figure 26: xz plan

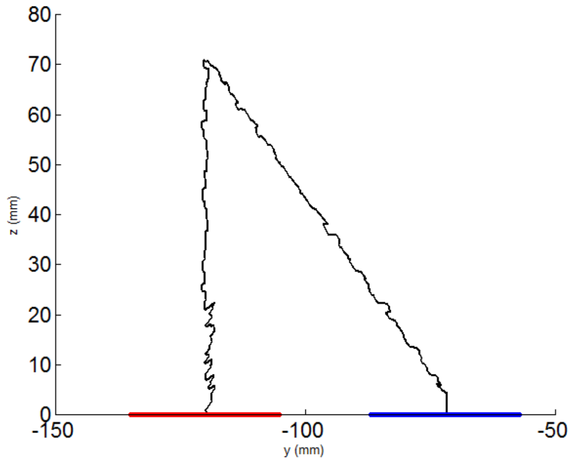


Figure 27: yz plan

As can be seen from the figures, the real trajectory is not stable, especially in the beginning period. The main causes are: differences between three encoders of motors, rounding of values when calculating forward kinematic equation, PID controllers is not really optimized. Besides, errors in mechanical transmission also affect to the accuracy of the system.

Conclusion:

The paper gives an overview of applying Dijkstra algorithm to trajectory optimization for an automatic chess-playing SCARA. The system was also experimented with human opponents in real matches. Results can be improved by increasing the quantity of nodes on chessboard as well as the number of layer. However, the drawback of this action is that the calculating time will be longer.

References:

[1] N.D. Anh, L.T. Nhat, T. Van Phan Nhan (2015) “Design and control automatic chess-playing robot arm”, Lecture Notes in Electrical Engineering (Vol 371), 485-496

[2] David Galva ˜o Wall, John Economou, Hugh Goyder, K Knowles, Peter Silson and Martin Lawrance, “Mobile robot arm trajectory generation for operation in confined environments”, Journal Systems and Control Engineering (Vol 229), 215-234

[3] Zhang, J. and Wang, Y. (2008), “The construction of the digital campus URP system”. Journal of Beijing University of Posts and Telecommunications (social science edition), 72–75

[4] Yun Long Yi and Ying Guan (2012), “A Path Planning Method to Robot Soccer Based on Dijkstra Algorithm”. Advances in Electronic Commerce, Web Application and Communication (Vol 2), 89-95

[5] T.V. Hoai, “Shortest path”, lecture in “Discrete mathematics”, Master course (2009-2010).

[6] Marco Ulricha, Gregor Lux, Leila Jurgensen, Gunther Reinhart, “Automated and Cycle Time Optimized Path Planning for Robot-Based Inspection Systems”, 6th CIRP Conference on Assembly Technologies and Systems, 377-382.