Heuristic Solution for Constrained 7-DOF Motion Planning in 3D Scanning Application

Theodor Borangiu[a], Alexandru Dumitrache[a], Anamaria Dogar[a]

[a] *Centre for Research and Training in Industrial Control*
Robotics and Materials Engineering
University Politehnica of Bucharest, RO

Abstract

The laser scanning system presented here consists of a medium range triangulation based laser probe mounted on a 6-DOF articulated robotic arm, having the workpiece attached to a rotary table, which represents the 7^{th} degree of freedom. This article focuses on planning the motions of the rotary table and the robotic arm in order to follow a desired sequence of scanning toolpaths, which are defined in the scanned workpiece's reference frame.

Key words: motion planning, inverse kinematics, 7-DOF manipulator, 3D laser scanning
PACS: 45.40.Ln, 45.40.Bb

1. Introduction

This work is part of a project whose goal is to develop a 3D laser scanner system by using a 6-DOF vertical articulated robot arm to move a triangulation-based laser probe around the object of interest, which is placed on a rotary table. The manipulator has a spherical working envelope with a radius of 650 mm and the laser probe is able to measure distances from 70 to 250 millimetres with an accuracy of $30\,\mu\mathrm{m}$. An overview of the scanning system is illustrated in Fig. 1. The robotic arm will move around the workpiece being analyzed using computer-generated adaptive scanning paths, which are computed in real-time while the scanning sys-

Email addresses: borangiu@cimr.pub.ro (Theodor Borangiu), alex@cimr.pub.ro (Alexandru Dumitrache), dogar@cimr.pub.ro (Anamaria Dogar)
URL: www.cimr.pub.ro (Theodor Borangiu)

tem is discovering the features of the object. The resulting 3D model will be used for reproduction of the scanned parts on a 4-axis CNC milling machine.

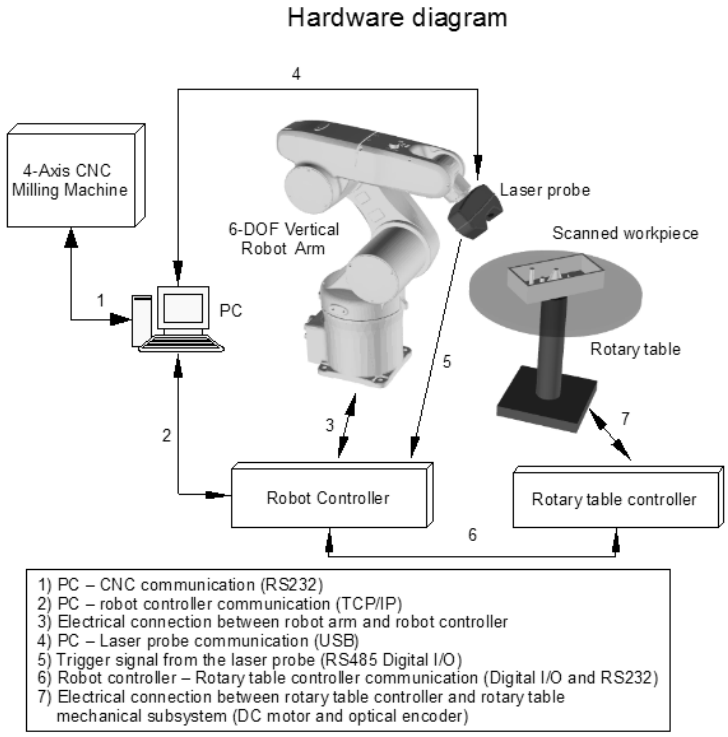


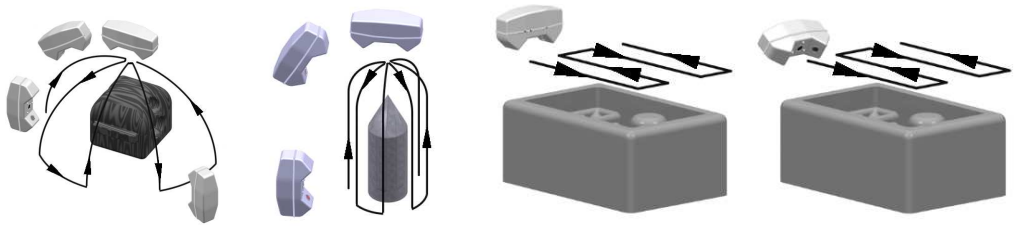
Figure 1: Main components of the scanning system

2. Problem description

The scanning trajectories are defined as motions in a reference frame attached to the workpiece. Some examples of scanning trajectories are given in Fig. 2a-d. The trajectories are discretized, i. e. described by a series of closely spaced locations (which include position and orientation), and the path between these locations is approximated with linear interpolation. The position change between two discrete steps is executed as a straight line motion. For the orientation change, the rotation matrix between two discrete steps is represented in axis-angle format, and the linear interpolation is performed on the rotation angle, keeping the rotation axis fixed.

The scanning trajectories are generated as sequences of motions, described by positions and orientations of the laser probe with respect to the workpiece. Since the robot arm has 6 degrees of freedom, this appears to be sufficient in order to achieve any desired position and orientation. The reason for using the rotary table is that the robotic arm alone is not able to look at the piece from the front side and then from the back side, especially for large workpieces. For the trajectories in Fig. 2 (a) and (b), it is clear that the workpiece has to be rotated on the table in order to complete the motion. For the trajectories in Fig. 2(c) and (d), if the object is small enough, it will be possible to perform the zig-zag path without rotating the table, but for larger objects, an out-of-range condition for the robot arm will occur, and that can be avoided by rotating the table.

A straightforward solution is to incorporate the rotary table motion in the scanning trajectory. This solution may be good for simple scanning patterns like those presented in Fig. 2, where the solution for the rotary table motion is obvious. However, as one of the aims of this work is to develop automated adaptive scanning procedures, by discovering the workpiece's concave regions which are hard to reach (Impoco et al., 2005), the resulting trajectories will be more complex. Also, the generation of adaptive scanning trajectories is not an easy task, so it was decided to make the task of rotary table movement completely automated.



(a) (b) (c) (d)

Figure 2: Examples of scanning patterns: (a) spherical; (b) cylindrical; (c) zig-zag; (d) zig-zag with tilted probe

The aim of this paper is to present an algorithm that determines automatically a suitable trajectory for the rotary table, when a scanning path around the workpiece is known. Input data for this problem consists of the scanning paths, which are a series of locations (positions and orientations), in the workpieces reference frame. The scanning system has to move continuously and synchronously the rotary table and the robot arm, such as the laser probe reaches the programmed locations and take measurements without stopping the system motion. Output data is a sequence

of joint values of the robotic arm and the angle of the rotary table, which give the desired location of the laser probe with respect to the workpiece. In other words, the problem presented here is the inverse kinematics (*IK*) for a 7-DOF kinematic chain. In addition, the computed solution has to satisfy the following requirements:

- minimize the accelerations and limit the speed of the rotary table;
- avoid collisions with any obstacles which may be within the manipulators range, or between the manipulator and the rotary table;
- flexible reach of the robotic arm and avoiding the proximity of singular configurations

3. Problem modelling and formalization

3.1. Kinematics model

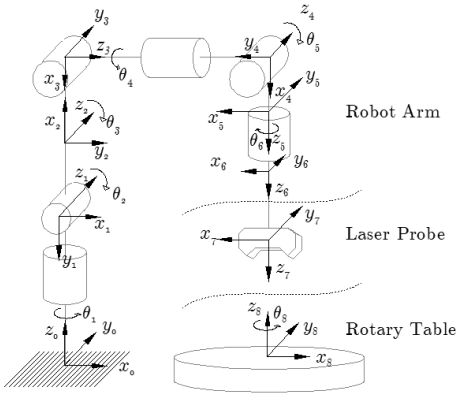
From a kinematics point of view, the 6-DOF robot arm and the rotary table are modelled using the Denavit-Hartenberg convention (Spong et al., 2002), as shown in 3(a) and 3(b). The rotary table may be considered fixed and the robotic arm rotating around the workpiece, hence the effect of the 7^{th} degree of freedom is applied before the other 6 links in the kinematic chain. The position of the rotary table relative to robot is modelled therefore as a link 0, or link R, where θ_R is the rotary angle variable.

The location of the laser probe with respect to the workpiece is specified as a homogeneous transformation matrix (HTM), which contains information about the cartesian position and the orientation. The orientation may be specified in yaw-pitch-roll angles or axis-angle formats, but it should be converted to the HTM format. The scanning trajectory is specified as a series of HTMs, each matrix $T_L^{(k)}$ corresponding to a discrete time step k.

The location of the laser probe with respect to the manipulator base is $T_M^{(k)}$, which is the product between the direct kinematics solution for the 6-DOF arm, and the tool transform T_{TL} which specifies the location of the laser probe with respect to the the robot arm flange, and is computed using a calibration procedure.

$$T_M^{(k)} = T_{DK}\left(\theta_1^{(k)} \dots \theta_6^{(k)}\right) \cdot T_{TL} \tag{1}$$

The location of the rotary table with respect to the robot base is $T_R^{(k)}\left(\theta_R^{(k)}\right)$, which is a translation which shows the position of the table with respect to manipulator,



(a) Reference frame assignment

Link	a_i [mm]	d_i [mm]	α_i [deg]	θ_i [deg]
Rot	–500	–200	0	$-\theta_R$
1	75	335	–90	θ_1
2	270	0	0	θ_2
3	–90	0	90	θ_3
4	0	295	–90	θ_4
5	0	0	90	θ_5
6	0	80	0	θ_6

(b) Denavit-Hartenberg parameters

Figure 3: Kinematics model of the scanning system

followed rotation around OZ:

$$T_R^{(k)}\left(\theta_R^{(k)}\right) = \mathcal{T}\left(-a_0, 0, -d_0\right) \cdot \mathcal{R}_Z\left(\theta_R^{(k)}\right) \tag{2}$$

The inverse kinematics for the 6-DOF manipulator is solved by its controller unit, and will be named *IK*$_6$:

$$\theta_{1\ldots6}^{(k)} = \mathit{IK}_6\left(T_{DK}^{(k)}\right) \tag{3}$$

Remark 1: The problem *IK*$_6$ is said to have a solution when there is a set of angles $\theta_1^{(k)} \ldots \theta_6^{(k)}$ which satisfy the equation:

$$T_{DK}^{(k)} = T_{DK}\left(\theta_{1\ldots6}^{(k)}\right) \tag{4}$$

and the angles are within the allowed range for the manipulator. If the mathematical condition can be satisfied, but the joint values are out of the allowed ranges, it will be considered that *IK*$_6$ has no solution.

3.2. Problem decomposition

The inverse kinematics of this 7-DOF chain can be solved in two steps:

1. Choose a suitable angle for the rotary table;
2. Solve the inverse kinematics for the 6-DOF manipulator and the angle chosen at step 1.

Supposing that at time step k, the laser probe has to be placed in the workpiece reference frame at the location $T_L^{(k)}$, and the angle $\theta_R^{(k)}$ was chosen, the position of the laser probe with respect to robot should be:

$$T_M^{(k)}\left(\theta_R^{(k)}\right) = T_R^{(k)}\left(\theta_R^{(k)}\right) \cdot T_L^{(k)} \tag{5}$$

Since IK_6 usually has a finite number of solutions, and when a robot configuration (LEFTY / RIGHTY, ABOVE / BELOW or FLIP / NOFLIP) is selected, a unique solution can be chosen, and the planning problem would be solved:

$$\theta_{1\ldots6}^{(k)} = IK_6\left(T_M^{(k)}\left(\theta_R^{(k)}\right) \cdot \left(T_{TL}\right)^{-1}\right) \tag{6}$$

Therefore, the 7-DOF planning problem is reduced to the planning of a single joint value, the rotary table angle $\theta_R^{(k)}$, for all time steps $k = \overline{1,n}$.

The rotary angle has to be choosen such as there exists at least one solution for the second step. Therefore, a very simple solution can be derived. The rotary angle can be modified, with fixed increments, e.g. 1 degree, until the IK_6 problem has a solution. This implies that every time the robot arm is out of range and cannot reach $T_M^{(k)}\left(\theta_R^{(k)}\right)$, it has to wait until the table rotates to an appropriate value of $\theta_R^{(k)}$, for which IK_6 has a solution. Using this trivial solution as a fall-back method guarantees the resolution-completeness (S. M. LaValle et al., 2006) of the planning solution.

3.3. The configuration map

The configuration space for this problem can be represented in its discrete form as a two dimensional image, or map, where the X axis corresponds to the discrete time $k = \overline{1,n}$, and the Y axis corresponds to the rotary angle θ_R ranging from -180° to 180° in $n+1$ equally spaced discrete steps. Since the rotary table can rotate continuously, without any limit on the number of complete rotations, the configuration map is periodic on the Y axis.

The mapping from the continuous rotary table angle θ_R to the discrete line index i in the map M_B is:

$$\begin{aligned} \mathrm{c2d}(\theta) &= \mathrm{round}\left(\frac{(\theta + 180\,^\circ)\cdot n}{360\,^\circ}\right) \mod n \\ \mathrm{d2c}(i) &= \frac{(i-1)\cdot 360\,^\circ}{n} - 180\,^\circ \end{aligned} \tag{7}$$

The pixel value at location (i, j) in the map M_B corresponds therefore to time step j and to rotary angle $\theta_R^{(i)}$, and it has the following meaning:

$$M_B(i,j) = \begin{cases} 0 \text{ (black): } \textit{IK}_6 \text{ has no solutions for } T_L^{(j)} \text{ and } \theta^* = \mathrm{d2c}(i) \\ 1 \text{ (white): } \textit{IK}_6 \text{ has solution for the above inputs.} \end{cases} \tag{8}$$

This image will be called the *binary configuration map*, and an example is shown in Fig. 4. It will be shown later how the configuration map can be used to impose additional constraints on the planned scanning path.

The white (allowed) regions on the map will be named C_{free} , while the black (forbidden) regions will be denoted as C_{obs}. When considering the continuous representation of the configuration space, C_{free} is an open set, while C_{obs} is closed (S. M. LaValle et al., 2006).

Given the mechanism geometry and limits, which are constant and determine whether *IK*$_6$ has a solution or not, the configuration map depends only on the set of desired scanning paths $T_L^{(j)}$.

Having defined the configuration map, the planning algorithm has to find a way through *obstacles* (not necessary physical obstacles, but forbidden areas on the map), from the starting rotary table angle, in the first column, to any position in the last column. The evolution of the path through the configuration map is the evolution of the rotary table angle θ_R so that the obstacles on the map are avoided. Since the rotary table rotates continuously, there is no need for the robot arm to wait, thus minimizing the scanning time.

3.4. The graylevel configuration map

As illustrated in S. M. LaValle et al. (2006), a shortest path solution through the configuration space is likely to touch the obstacles, which is not desirable. When the robot is close to the limits of C_{obs}, the robot is either close to the limits of its joints, or to the limits of its working envelope, or close to a singular point.

In order to obtain a planning algorithm that does not touches the obstacles, but maintains a sufficient distance, one has either to add borders to C_{obs}, or modify the interpretation of the map values in C_{free} to indicate the proximity of an obstacle.

The values of the map in C_{obs} remain zero, which means that these are forbidden states. The values in C_{free} will be in the $(0, 1]$ interval, showing that any of these states are allowed, but also indicating how desirable is the state. Therefore, states having higher values are more desirable than states having values close to 0.

Considering the joint limits of the robot for a given link j, it is possible to use a function which is equal to zero at the joint limits ($\theta_j^{\min} \leq \theta_j \leq \theta_j^{\max}$), and reaches its maximum value at the middle of the interval.

$$f_j(\theta_j) = \left(\sin \left(\frac{\theta_j - \theta_j^{\min}}{\theta_j^{\max} - \theta_j^{\min}} \cdot \pi \right) \right)^{\gamma_j} \tag{9}$$

A function showing the value of a given robot configuration can be constructed by multiplying the individual joint functions:

$$f_R\left(\theta_{1\ldots6}\right) = \prod_{j=1}^{6} f_j(\theta_j) \tag{10}$$

By adjusting the values of the exponents γ_j, it is possible to control the shape of the functions (Fig. 5b).

The *grayscale configuration map* will be represented as a 2D grayscale image, with real values between 0 and 1, like the example in Fig. 5a, having the following meaning:

$$M_G(i,j) = \begin{cases} 0 \text{ (black): } \textit{IK}_6 \text{ has no solutions for } T_L^{(j)} \text{ and } \theta^* = \mathrm{d2c}(i) \\ f_R\left(\theta_{1\ldots6}\right) : \textit{IK}_6 \text{ has one solution: } \theta_{1\ldots6} \end{cases} \tag{11}$$

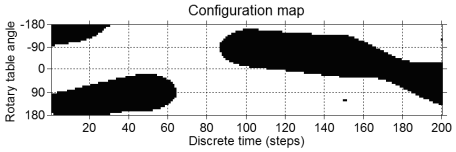
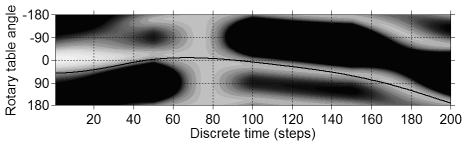
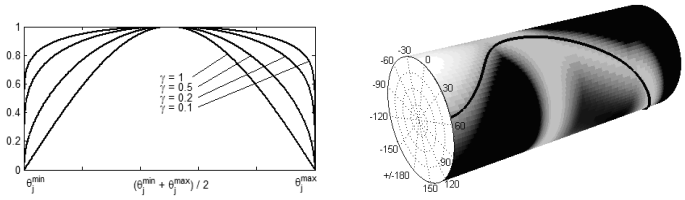


Figure 4: An example of binary configuration map



(a)



(b) (c)

Figure 5: An example of grayscale configuration map: (a) Plain view; (b) The value function for an individual joint; (c) Cylindrical view

(a) (b) (c) (d)

Figure 6: Robot arm and laser looking at a workpiece from a given pose T_L (eq. 12):
(a) Robot is near a "too close" condition (table rotated at $-25\,^\circ$)
(b) Robot is near a "too far" condition (table rotated at $-95\,^\circ$)
(c) Robot is not close to its limits (table rotated at $-60\,^\circ$)
(d) Value function $f_R(\theta,\ T_L)$ for $\theta = [-180\,^\circ,\ 180\,^\circ]$ and T_L from (12)

As the angle is represented on the Y axis, and the table is able to make an unlimited number of rotations, the path can wrap around on this axis, and the map can also have a cylindrical representation, like in Fig. 5b.

When searching a path on the M_G map, an algorithm has to take into account both the differential constraints (speed and accelerations) and the value function $M_G(i, j)$, which will allow finding a smooth path for the rotary table, which will not touch the configuration space obstacles.

In Fig. 6, the interpretation of a single column from the configuration map is detailed. The robot was requested to look at the center of the workpiece, at height $h = 50\,\mathrm{mm}$, using $yaw = 0\,^\circ$, $pitch = 100\,^\circ$ and $roll = 90\,^\circ$, from a distance of $250\,\mathrm{mm}$. The transformation T_L relative to the workpiece is:

$$T_L = \mathcal{T}(0, 0, 50)\mathcal{R}_Y(100\,^\circ)\mathcal{R}_Z(90\,^\circ)\mathcal{T}(0, 0, -250) \tag{12}$$

This transformation can be achieved using different values for the rotary table angle, as shown in the images (a)...(c) and on the graph (d). The angles on the graph for which the function f_R is equal to 0 correspond to out of range configurations for the robot. In Fig. 6d, the reader may observe the configurations from (a), (b) and (c) marked on the graph. The first two configurations have a low value of f_R , which means they are near an out of range configuration and it is desirable to be avoided. The configuration from (c) is near a local maxima of the function f_R , which means that it is much more desirable, as it is far from the joint limits. Moreover, the robot arm stands in a natural (or relaxed) joint configuration.

3.5. Computing the configuration maps

In order to compute the value of a point on the map $M_B(i, j)$ or $M_G(i, j)$, one needs to know whether the desired transformation $T_L^{(j)}$ can be reached by the robot arm, and for the grayscale map, one also needs to know the joint values of the robot arm, in order to compute the values of f_R in C_{free} . Therefore, the computer that runs the planning algorithm needs to know the inverse kinematics (IK) function for the 6-DOF manipulator.

One solution is to use the inverse kinematics routines from the robot controller. However, as the amount of IK calls would be very large, this solution is not practical. A better alternative will be to run the planner on a PC, which has much more processing power than the robot controller. A straightforward solution is to use a generic IK procedure, such as the one provided by Peter Corke's Robotic Toolbox (Corke, 1996). This approach has been implemented and found to be slow, as the procedure from the toolbox uses an iterative method, the pseudoinverse of

jacobian. The solution is to use a closed form IK (CIKS) for the 6-DOF robot arm.

The equations of the closed form $IK_6(T_{DK})$, which were derived using the kinematic decoupling method for manipulators with spherical wrist, presented in Borangiu et al. (2002), are given below. The values of the parameters were given in Fig. 3(b). The robot configuration has a LEFT arm, ABOVE elbow and NOFLIP wrist (see Borangiu et al. (2002) and Adept (2007)), therefore a single solution is computed.

The first 3 joints give the position of the wrist P, given by the cartesian vector $[P_{1,4},\ P_{2,4},\ P_{3,4}]$:

$$T_P = \mathcal{T}(0, 0, -d_6) \tag{13}$$

$$P = T_{DK} \cdot T_P \tag{14}$$

$$p_{wx} = P_{1,4}; \quad p_{wy} = P_{2,4}; \quad p_{wz} = P_{3,4}; \tag{15}$$

$$\theta_1 = \text{atan2}(p_{wy},\ p_{wx}) \tag{16}$$

$$L_1 = a_2; \quad L_2 = \sqrt{(a_3)^2 + (d_4)^2} \tag{17}$$

$$L_3 = \sqrt{\left(\sqrt{{p_{wx}}^2 + {p_{wy}}^2} - a_1\right)^2 + (p_{wz} - d_1)^2} \tag{18}$$

$$p = \frac{L_1 + L_2 + L_3}{2} \tag{19}$$

$$h = \frac{2\sqrt{p(p - L_1)(p - L_2)(p - L_3)}}{L_3} \tag{20}$$

$$\alpha_1 = \text{atan2}\left(p_{wz} - d_1,\ \sqrt{{p_{wx}}^2 + {p_{wy}}^2} - a_1\right) \tag{21}$$

$$\alpha_2 = \arcsin\frac{h}{L1} \tag{22}$$

$$\theta_2 = -(\alpha_1 + \alpha_2) \tag{23}$$

$$\theta_3 = \text{atan2}(-a_3,\ d_4) - \arccos\frac{{L_1}^2 + {L_2}^2 + {L_3}^2}{2\ L_1\ L_2} + 270\,^\circ \tag{24}$$

The remaining 3 joints give the orientation of the wrist. Let R_3^0 be the rotation matrix from the robot base to the 3^{th} link, R_6^0 the rotation from base to the 6^{th} link, and R_6^3 the rotation of the spherical wrist:

$$R_3^0 = \mathcal{R}_Z(\theta_1) \cdot \mathcal{R}_X(-90\,^\circ) \cdot \mathcal{R}_Z(\theta_2 + \theta_3) \cdot \mathcal{R}_X(90\,^\circ) \tag{25}$$

$$R_6^0 = P_{1..3,1..3} \tag{26}$$

$$R_6^3 = \left(R_3^0\right)^{-1}\ R_6^0 \tag{27}$$

Since $R_3^6 = \mathcal{R}_Z(\theta_4)\ \mathcal{R}_Y(\theta_5)\ \mathcal{R}_Z(\theta_6)$ from the spherical wrist geometry, $\theta_{4...6}$ are ZYZ Euler angles:

$$\theta_4 = \mathrm{atan2}\left(\left(R_6^3\right)_{2,3},\ \left(R_6^3\right)_{1,3}\right) \tag{28}$$

$$\theta_5 = \mathrm{atan2}\left(\sqrt{\left(\left(R_6^3\right)_{3,1}\right)^2 + \left(\left(R_6^3\right)_{3,2}\right)^2},\ \left(R_6^3\right)_{3,3}\right) \tag{29}$$

$$\theta_6 = \mathrm{atan2}\left(\left(R_6^3\right)_{3,2},\ -\left(R_6^3\right)_{3,1}\right) \tag{30}$$

The special cases when $\theta_5 = 0$ and $\theta_5 = 180\,^\circ$ are handled by choosing $\theta_4 = 0$. The equations (13) through (30) are not very simple, as they involve a lot of trigonometric function calls, but their biggest advantage over a generic iterative method is that they can be evaluated in $O(1)$ complexity. A C# .NET implementation of the above equations is able to compute about 250.000 IK solutions per second on a Pentium 4-M processor running at 2 GHz, which is more than sufficient for the planning task.

In order to test whether the robot is able to move the laser probe in the pose with respect to the workpiece, one will use the function "inrange" which will be defined below.

By combining equations (5) and (6), one can obtain the transformation T_{DK} required by IK_6 :

$$\theta_{1..6} = IK_6\left(T_R\left(\theta_R\right) \cdot T_L \cdot \left(T_{TL}\right)^{-1}\right) \tag{31}$$

The function *inrange* will be defined as:

$$\mathrm{inrange}(\theta,\ T_L) = \begin{cases} 0, & \text{if eq. (31) has no solution or it is out of range} \\ 1, & \text{otherwise} \end{cases} \tag{32}$$

In other words, the function *inrange* is the continuous counterpart of the binary configuration map M_B. The function will be called to compute the discrete map image M_B. The function which defines the grayscale map is:

$$f_R(\theta,\ T_L) = \begin{cases} 0, & \text{if eq. (31) has no solution or it is out of range} \\ f_R\left(IK_6\left(T_R(\theta_R) \cdot T_L \cdot \left(T_{TL}\right)^{-1}\right)\right) & \text{otherwise} \end{cases} \tag{33}$$

As a remark, the configuration maps MB and MG will not be computed for every pixel in the actual algorithm implementation. The maps are described and generated in order to illustrate the concept, and are displayed on the screen for debugging purposes. The planning algorithm will use directly the functions $inrange(\theta,\ T_L)$ or $f_R(\theta,\ T_L)$, and may cache the computed values in a matrix if it is going to reference them many times, for an increase in speed.

3.6. Specifying additional constraints using the configuration map

The configuration map can be used to specify additional constraints on the robot movement, for example, physical obstacle avoidance. There can be defined cartesian obstacles of an arbitrary shape (square, cylinder, sphere) having known dimensions, and multiply the function f_R with a new component, which is zero if the robot hits the obstacle, 1 if the robot is far from the obstacle, and has intermediate values if the robot is in the proximity to the obstacle, but not touching it. This function can be computed by a collision avoidance routine, and its results are integrated with the planner by using the above mentioned function. The same procedure can be used for avoiding singular configurations in the robot arm. These situations can be avoided by using a function that is close to 0 near a singularity, and 1 otherwise. There may be locations which can be reach both with or without a singular configuration of the robot arm; in this case, because the configuration value function has a very low value, the planner will avoid it and choose a non-singular solution. There may also be a situation where the singular configuration cannot be avoided, for example, when the laser probe looks down on the table and their Z axes are aligned. In this case, all the solutions are singular configurations and cannot be avoided. For this reason, the function which weights the singularities is not reccommended to be zero, and further care is required when generating the motion instructions for the robot. As a final remark, the constraints that can be specified using the configuration map are *static*, i. e. they are not related to the speeds or accelerations of the system; they only restrict the instantaneous positions of the robot.

3.7. Performance criteria

This section defines a cost function for evaluating a path computed by a planning algorithm, in order to formalize the requirements presented in Section 2.

The effects of a suboptimal path may be one of:

- Delays in the scanning process
- High accelerations and speeds for the rotary table

There are maximum limits on the absolute values of the rotary table angular speed and acceleration, $\omega_{\max}$ and $a_{\max}$. If a planned path exceeds these limits, the robot arm has to stop and wait for the table to rotate in a correct position, while the table is moving at its maximum speed or acceleration. This is the worst case possible, mainly because, due to a poor planned path, the scanning process is delayed. If the planned path does not exceed the speed and acceleration limits, the rotary table motion is smooth and no delays occur in the scanning process.

Therefore, the main objective of the planning algorithms presented here is finding a path that avoids the obstacles on the configuration map, and respects the speed and acceleration limits.

An essential feature of the planning algorithm will be its ability to run in real time, while the scanning process takes place. Therefore, the optimality of the path computed is less important, and for this reason, this paper will focus on faster, but suboptimal, heuristic algorithms.

Let n be the number of discrete time steps for a given planning problem. The initial conditions are the rotary table angle $\theta_R^{(0)}$ and the rotary table angular velocity $\omega_R^{(0)}$. The planner will compute the angles $\theta_R^{(1)}$ through $\theta_R^{(n)}$, which are obtained by integrating the angular velocities $\omega_R^{(0)}$ to $\omega_R^{(n-1)}$ or performing a double integration of the angular accelerations $a_R^{(0)}$ to $a_R^{(n-1)}$.

During the discrete moments k and $k+1$, which correspond to continuous time moments t_k and $t_{k+1} = t_k + \Delta t$, the table rotates using the constant acceleration $a_R^{(k)}$. Therefore,

$$\omega_R^{(k+1)} = \omega_R^{(k)} + a_R^{(k)}\ \Delta t \tag{34}$$

$$\theta_R^{(k+1)} = \theta_R^{(k)} + \omega_R^{(k)}\ \Delta t + a_R^{(k)}\frac{(\Delta t)^2}{2} \tag{35}$$

The penalty due to rotary table motion (high speeds or high accelerations) can be expressed as:

$$C_{a\omega} = \sum_{i=0}^{n-1}\left(k_a\left(a_R^{(i)}\right)^2 + k_\omega\left(\omega_R^{(i)} + a_R^{(i)}\ \Delta t\right)^2\right) \tag{36}$$

The coefficients k_a and k_ω are scalar weights for the angular acceleration and speed of the rotary table, and they may be used for tuning the algorithms.

The cost due to advancing on the gray region of configuration map can be defined as:

$$C_{adv} = \sum_{i=1}^{n} \left(1 - f_R\left(\theta_i,\ T_L^{(i)}\right)\right) \tag{37}$$

If the robot at step i can position the laser probe at the position $T_L^{(i-1)}$, but it cannot reach $T_L^{(i)}$ by rotating the table in the time Δt with respect to the maximum acceleration and speed, the scanning process has to wait until the rotary table comes into an acceptable position, so that the robot is able to reach $T_L^{(i)}$.

Therefore, a delay in the scanning process is introduced, and it will be named $d^{(i)}$. The delay depends on the amount of the rotation needed, and the maximum speed and acceleration limits. By using a trapezoidal acceleration profile, the delay can be computed. If the table rotation can be performed in the time Δt, no delay is introduced from time steps $i-1$ to i, and therefore, $d^{(i)} = 0$. By summing the delays for every time step and weighting them with the scalar k_d, the total delay cost for a given path is:

$$C_{delay} = k_d \sum_{i=1}^{n} d(i) \tag{38}$$

The total cost function of a given path will be:

$$C_{path}\left(\theta_R^{(1\ldots n)},\ \omega_R^{(0\ldots n-1)},\ a_R^{(0\ldots n-1)},\ d^{(1\ldots n)}\right) = C_{a\omega} + C_{delay} + C_{adv} \tag{39}$$

4. Algorithms

Any planning algorithm presented here will start from an initial condition vector of the rotating system, which contains the rotary table angle $\theta_R^{(0)}$ and the rotary table angular velocity $\omega_R^{(0)}$. The destination is not fixed, the planned path has only to reach the last column in the configuration map, at any rotary angle and angular velocity, $\theta_R^{(n)}$ and $\omega_R^{(n)}$. After the map is traversed, the scanning trajectory is completed and the rotary table may either decelerate and stop, or prepare for another scanning trajectory.

Related work in planning algorithms focus on two or more dimensions in the configuration space (Latombe et al., 1999), (M. Sharir, 2004). Successful solutions include roadmap-based planners, which use a graph that represents the

connectivity of the free configuration space, and potential field methods (Y. K. Hwang, N. Ahuja, 1992), which represent forbidden areas as repulsive forces and goals as attractive forces, allowing the planning problem to be solved using gradient-based optimization method. Recent approaces use genetic algorithms (Kazem et al., 2008), probabilistic methods and random sampling schemes (J. Barraquand et al., 1997).

The planning problem presented in this article is similar to a smooth path finding problem for a point moving in a 2D space; however, the second dimension represents the time, since the duration of the motion and the speed profile for the laser sensor with respect to the workpiece are known a priori. Therefore, the state-of-art algorithms for 2D problems cannot be applied directly, since obviously the system cannot turn back in time, but at every time step it advances on the map with one increment.

The planning problem has speed and acceleration constraints, and therefore it can be considered nonholonomic Tanner et al. (2000). A possible solution is to expand the configuration space with a third dimension representing the rotational speed of the table; in this way, the problem can be solved optimally using a Dijkstra-like algorithm S. M. LaValle et al. (2006) and Cormen et al. (2000).

4.1. Rotate when out of range

This algorithm, given in pseudocode below, is the simplest strategy for rotating the table. Whenever the trajectory hits an obstacle on the binary configuration map, the algorithm finds the minimum amount of rotation that gets out of the forbidden map area.

By using this solution, the path computed will touch the forbidden areas (Fig. 7), which is not desirable, and every time the table has to turn, the laser scanner will wait until the computed rotation angle is reached. This approach does not obey the prescribed timing characteristics of the scanning trajectory, but it allows the laser sensor to reach all the programmed locations and extract the 3D data successfully.

In this and all subsequent examples, the time step used is $\Delta t = 0.1\,\mathrm{s}$, therefore the test motion, which has 200 discrete steps, takes 20 seconds to complete, unless otherwise stated.

This is a trivial strategy which is actually used as the fall-back mechanism.

4.2. Local maxima search

This algorithm attempts to use the graylevel configuration map as a potential field, and at every time step, i. e. on every column on the map, steer the table towards a local maxima of the current column. The speed of steering, and also

Algorithm 1: Rotate when out of range

$\theta \leftarrow \theta_R^{(0)}$
for $j = 1$ **to** n **do**
  if inrange$(\theta, T_L^{(j)})$ **then**
    for $\varphi = 0$ **to** 180 **step** $\Delta\theta$ **do**
      if inrange$(\theta + \varphi, T_L^{(j)})$ **then**
        $\theta \leftarrow \theta + \varphi$
        break
      if inrange$(\theta - \varphi, T_L^{(j)})$ **then**
        $\theta \leftarrow \theta - \varphi$
        break
  $\theta_R^{(j)} \leftarrow \theta$

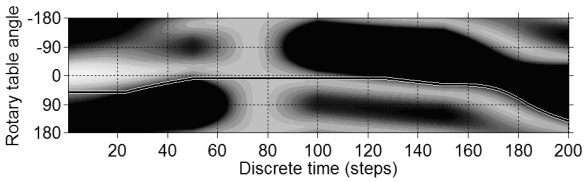


Figure 7: Sample solution for Algorithm 1. The planned path touches the edges of the obstacles.

the change in speed, are weighted according to the performance criteria specified previously, in order to ensure a smooth motion.

This approach works better than the first method; however, it is sensitive to local maximas in the graylevel map, which in many cases may lead to deadlock situations.

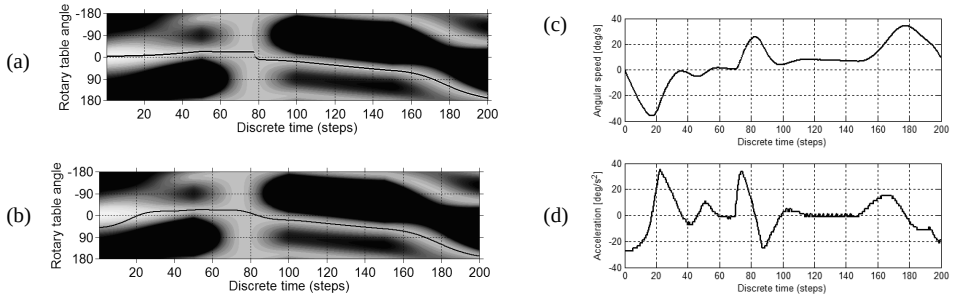


Figure 8: (a) Local maxima guide path
(b) Smoothed local maxima path
(c) Angular speed (smooth path) $[\,\mathrm{deg/s}]$
(d) Angular acceleration$[\,\mathrm{deg/s^2}]$

The guide path cannot be followed exactly, because it is possible to have sharp corners (see Fig. 8a) , which require high instantaneous acceleration rates. Instead, the path is smoothed (Fig. 8b,c,d) so that it can be followed much easier.

The experiments show that the results are much better than in the first case, as the planned path departs from the obstacles before touching them. For many practical cases, the planned path is smooth and does not introduce any delay in the scanning process. Since a local maxima can be found in constant time by searching over a finite set of angle values, and the smoothing can also be done with a finite number of computations at every step, this algorithm is able to per-

form the planning in real time. The main disadvantage of this algorithm is that the planned path may have high acceleration rates, sometimes much higher than necessary. Also, while the local maxima guide is not touching the obstacles, there is no guarantee that the smoothed path will have the same property. Should the smoothed path reach a forbidden area on the map, Algorithm 1 will provide a fall-back mechanism.

4.3. A Dijkstra-like algorithm

The Dijkstra algorithm finds the minimal cost path through a graph with positive weights (or costs) on its edges, from a given node to all the other nodes that can be reached from it Cormen et al. (2000). Given the discrete modelling of the planning problem, it can be formulated in terms of graph theory.

In order to be able to put constraints on the speeds and accelerations, a third dimension has to be added to the configuration space, the angular velocity. The graph nodes form a finite 3D matrix, as they map to a continuous 3D space, having on the first dimension the rotary table angle θ , on the second dimension the continuous time t , and on the third dimension, the angular velocity ω. A discrete representation (i, j, k) maps to an unique continuous configuration $(\theta_i,\ t_j,\ \omega_k)$. The reverse transform, from continuous to discrete, assigns a single discrete node (i, j, k) to a closed interval of continuous values:

$$\begin{pmatrix} \theta_i - \frac{\Delta\theta}{2} & \leq \theta < & \theta_i + \frac{\Delta\theta}{2} \\ t_j - \frac{\Delta t}{2} & \leq t < & t_j + \frac{\Delta t}{2} \\ \omega_k - \frac{\Delta\omega}{2} & \leq \omega < & \omega_k + \frac{\Delta\omega}{2} \end{pmatrix} \Rightarrow \begin{pmatrix} i \\ j \\ k \end{pmatrix} \tag{40}$$

where $i = c2d(\theta_i)$, $\theta_i = d2c(i)$ (see Eq. 7), and similar mappings are used for t and ω.

The method for obtaining a Dijkstra-like algorithm for a continuous problem is presented in S. M. LaValle et al. (2006).

From the node $(\theta_i,\ t_j,\ \omega_k)$, is it possible to advance using the acceleration a and reach the node corresponding to $(\theta_i + \omega_k\ \Delta t + a\frac{(\Delta t)^2}{2},\ t_j + \Delta t,\ \omega_k + a\Delta t)$. This is an edge in the graph having the cost $C^*_{edge} = k_a\ a^2$.

The costs put on velocities are modelled as costs into nodes. Every node corresponding to $(\theta_i,\ t_j,\ \omega_k)$ has a velocity cost equal to $k_\omega\ (\omega_k)^2$.

Also, every node has a cost corresponding to equation (37), which is equal to $1 - f_R(\theta_i,\ T_L^{(j)})$. Therefore, the cost of a node is:

$$C_{node} = k_\omega(\omega_k)^2 + 1 - f_R\left(\theta_i,\ T_L^{(j)}\right) \tag{41}$$

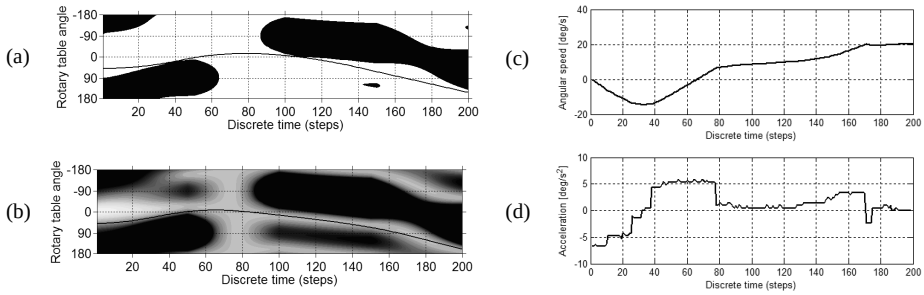


Figure 9: Examples for Dijkstra algorithm:
(a) Path computed for a binary configuration map
(b) Path computed for a grayscale configuration map
(c) Angular speed for the grayscale path $[\,\mathrm{deg/s}]$
(d) Angular acceleration for the grayscale path $[\,\mathrm{deg/s^2}]$

The costs on nodes on a graph can be transferred on the incoming edges, therefore the edge cost becoming:

$$C_{edge} = k_a\ a^2 + k_\omega(\omega_k)^2 + 1 - f_R\left(\theta_i,\ T_L^{(j)}\right) \tag{42}$$

If a delay has to be introduced, the edge cost increases with a component corresponding to C_{delay} from equation (38). This allows edges to traverse C_{obs}, in order to ensure that a solution is found even if C_{free} is not conex, but these edges should have a much higher cost than the others, so that this solution is selected as a last resort.

By summing the edge costs for an entire path, the cost function is equal to C_{path} from equation (39).

Because of this mode of cost assignment, the Dijkstra algorithm will minimize C_{path} , finding the optimal solution for the discretized problem.

An example of running the Dijkstra algorithm over a binary configuration map is given in Fig. 9. This is the smoothest possible solution for the rotary table (with respect to $C_{a\omega}$), but the path touches the obstacles, and therefore the robot arm will reach its joint limits, which is not desirable.

Running the same algorithm on the graylevel configuration map gives the result from Fig. 9b, where the planned path does not touch the edges of the obstacles, the motion is smooth and the acceleration rates are much lower than those in Fig. 8, obtained with the Local Maxima heuristic algorithm. As the discretization steps become smaller, the solution found by the Dijkstra algorithm converges to the optimal solution for the continuous problem.

The graph used in this algorithm can grow very large, as its size grows with the third power of the number of discretization steps of the 3D configuration space. If the space uses 100 steps on every dimension, the graph has 1,000,000 nodes, which is too much for a real time solution. However, the Dijkstra algorithm offers a reference trajectory, to which the solutions found by the other algorithms can be compared.

4.4. "Ray Shooting" heursitic algorithm

This algoritm attempts to provide a better solution than the local maxima heuristic, while retaining the possibility of real time planning. This algorithm does not have to find the optimal solution, but it has to compute a smooth motion, comparable to the one obtained with the Dijkstra algorithm in most practical situations.

At every time step k, the algorithm looks ahead p future time steps, that is, from $k+1$ through $k+p$. Over this range, it tries to perform a motion with constant angular acceleration, to ensure the smoothness of the planned path. The algorithm uses a finite set of acceleration values $a_j,\ \ j = \overline{1, n_a}$, and for every acceleration a_j , a possible path is evaluated, starting from the current state and spanning on the following p time steps. From the set of paths, the path having minimal cost value is chosen. This path corresponds to an acceleration equal to $a_{j,best}$, and the motion from time step k through time step $k+1$ will be performed using this acceleration. The planning algorithm advances to the next step and recomputes the paths, therefore at every time step it makes a decision. This means that at every time step there is a finite number of computations that have to be done, which make possible running the algorithm in real time.

Computing the path using a given acceleration is done by applying equations (34) and (35) over the desired range, and evaluating the cost is done using (39). The path computation stops if an obstacle is reached, and a cost penalty C_{pen} is added to the path, in order to favor the rays that did not hit any obstacle.

The accelerations considered in this algorithm form set of values, between $a_0 = a_{\min}$ and $a_{\max}$. The complete set has n_a elements $(p > 1)$:

$$A = \big[\underbrace{0,\ \pm a_0,\ \pm p\ a_0,\ \pm p^2\ a_0,\ \pm p^3\ a_0,\ ...}_{n_a \text{ elements}}\big] \tag{43}$$

The number of operations performed by this algorithm can be further limited by using a pruning scheme: if a ray is not going to obtain a better cost than the best path obtained so far at the current step, the computation stops.

Algorithm 2: "Ray Shooting" heuristic

$\theta \leftarrow \theta_R^{(0)}$
$\omega \leftarrow \omega_R^{(0)}$
for $j = 1$ **to** $n-1$ **do**
  for $i = 1$ **to** n_a **do**
    $C_i \leftarrow$ **evalpath**$(\theta,\ \omega,\ A_i,\ j+1,$ min
  $i \leftarrow \text{argmin}(C_i)$
  $a \leftarrow A_i$
  $\theta_{new} \leftarrow \theta + \omega\,\Delta t + a\,\frac{(\Delta t)^2}{2}$
  if inrange $\left(\theta_{new},\ T_L^{(j)}\right)$ **then**
    $\omega \leftarrow \omega + a\,\Delta t$
    $\theta_R^{(j)} \leftarrow \theta_{new}$
    $\omega_R^{(j)} \leftarrow \omega$
  else
    fall back using *Algorithm 1*

Function $C =$ **evalpath** $(\theta_0,\ \omega_0,\ a,\ j_{\text{ini}},\ j_{\text{fin}})$

$\theta \leftarrow \theta_0$
$\omega \leftarrow \omega_0$
$C \leftarrow 0$
for $j = j_{\text{ini}}$ **to** j_{fin} **do**
  $\omega \leftarrow \omega + a\,\Delta t$
  $\theta \leftarrow \theta + \omega\,\Delta t + a\,\frac{(\Delta t)^2}{2}$
  if inrange $\left(\theta,\ T_L^{(j)}\right)$ **then**
    $C \leftarrow C + k_a\,a^2 + k_\omega\,\omega^2 - f_R\left(\theta,\ T_L^{(j)}\right)$
  else
    $C \leftarrow C + C_{pen}$
    return

An example of running the Ray Shooting heuristic over the tested configuration map is given in Fig. 10. The figures show the path chosen until the current step, and the rays which determine the choice for the next step. In this example, the rays analyze the configuration map within 70 discrete time steps ahead.

In the first graph, Fig. 10a, the rays can "see" a solution for avoiding the obstacles by turning the table clockwise, i. e. θ_R decreasing, thus avoiding both obstacles on their "left" side. In the second graph, the rays begin to see an alternate solution, which would avoid the second obstacle on its "right" side, by changing the direction of the rotation of the table. Since the acceleration needed to change the rotation is smaller than the acceleration needed to avoid the second obstacle on the left, the planner chooses to reverse the rotation direction, resulting the solution from Fig. 10c.

In the last graph, one can view the planned solution, which is a smooth trajectory for the rotary table, and does not touch the obstacles, therefore the robot arm is not driven close to its joint limits. This solution satisfies the design requirements, has a shape similar to the one computed by Dijkstra algorithm, and it can be computed in real time.

The solution obtained by this algorithm is much smoother than the local maxima path, and is closer to the path computed by the Dijkstra algorithm. The acceleration rate for the two paths is comparable, as it can be observed by comparing Fig. 11 with Fig. 9. The planned path exhibits some small peaks in the acceleration, whose effects are minor and can be removed with a smoothing post-

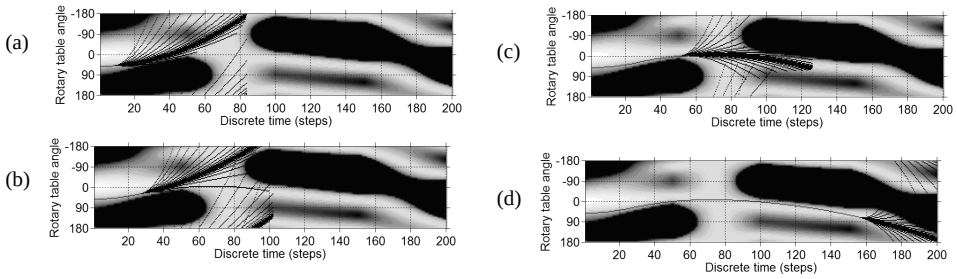


Figure 10: Ray Shooting example:
(a) The rays found a solution by avoiding both obstacles on the left side
(b) One of the rays avoids the second obstacle on the right side
(c) The algorithm decides to avoid the second obstacle on the right
(d) The algorithm has almost finished.

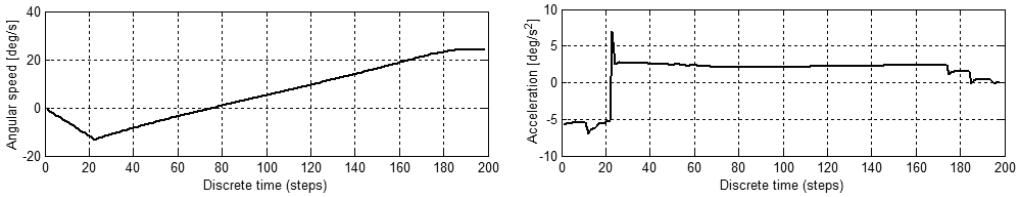


Figure 11: Speed and acceleration obtained by Ray Shooting algorithm

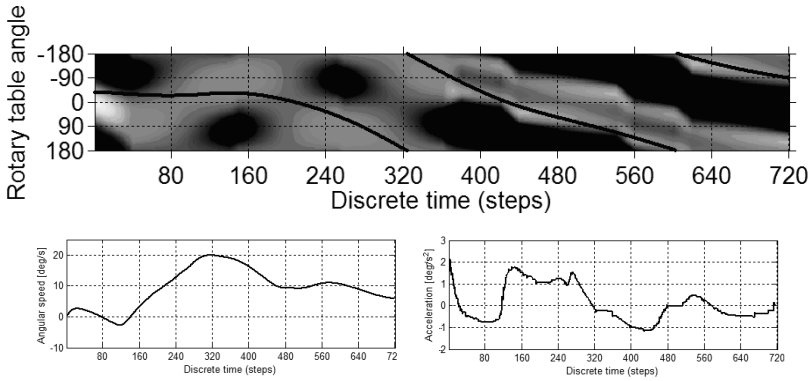


Figure 12: Another example of Ray Shooting algorithm. The planned path wraps around.

processing routine. There is also a higher tendency of the planned path being in the proximity of the obstacles, compared the Dijkstra solution.

In Fig. 12, another example of a path planned with the Ray Shooting algorithm is presented. The path starts from $38\,^\circ$ and ends at $637\,^\circ$, or $83\,^\circ$ when wrapped around, the total amount of rotation being $675\,^\circ$.

5. Conclusions

This article presented the problem of solving a redundant 7-DOF inverse kinematics problem, which usually has an infinite number of solutions. The motion planning problem has two constraints, the speed and the acceleration of the rotary table, which are used in order to achieve a smooth table motion, since there is no rigid mechanical fixture between the table and the scanned part.

The problem was modelled as finding the minimal cost path through a configuration space. The Dijkstra algorithm gave the optimal solution to the problem, however it was too slow in order to use it in a real time implementation. A heuristic algorithm was proposed, and it is able to compute a smooth path in real time, while respecting the imposed restrictions. The algorithm also has a fallback mechanism for difficult situations, which guarantees finding a solution if one exists.

The heuristic algorithm is implemented as a module for the laser scanning software, and it runs on the PC which deserves the 3D reconstruction system, performing the real-time planning of the rotary table and sending the planned result to the robot and rotary table controllers.

The proposed algorithm can also be used in other applications, which involve a robot moving on a 3D continuous path along a workpiece, performing a technological operation, i. e. welding, edge polishing, and there is a redundant degree of freedom whose motion has to be planned in order to achieve various constraints, such as collision and singularity avoidance, while maintaining a smooth trajectory.

Acknowledgements

This work is funded by the National University Research Council, in the framweork of the National Plan for Research, Development and Innovation.

References

Forward Kinematics: The Denavit-Hartenberg Convention. In: John Wiley and Sons, Inc., Robot Modeling and Control. 2005, p. 71-83.

S. M. LaValle, Planning Algorithms. Cambridge University Press, 2006.

P. I. Corke, A robotics toolbox for MATLAB, IEEE Robotics and Automation Magazine, Vol. 3, n. 1, pp. 24-32, March 1996.

Th. Borangiu, F. Ionescu, Robot Kinematics, In: AGIR and Editura Academiei Romne, Robot Modelling and Simulation, Bucharest 2002, p. 122-140

Adept Technology, Inc., Six-Axis Robot Configuration Singularities, March 2007

T. H. Cormen et. al., Single Source Shortest Paths, In: The MIT Press, Introduction to Algorithms, 2000, p. 527-531.

G. Impoco, P. Cignoni, R. Scopigno, A Six-Degrees-of-Freedom Planning Algorithm for the Acquisition of Complex Surfaces, International Journal of Shape Modeling, Vol. 11, no. 1, June 2005, p. 1-23.

Herbert G. Tanner and Kostas J. Kyriakopoulos, Nonholonomic Motion Planning for Mobile Manipulators, Proceedings of ICRA 2000

B. I. Kazem, A. I. Mahdi, A. T. Oudah, Motion Planning for a Robot Arm by Using Genetic Algorithm, Jordan Journal of Mechanical and Industrial Engineering, Vol. 2, No. 3, Sep. 2008, p. 131-136

J. Barraquand et al. (1997) A Random Sampling Scheme for Path Planning, The International Journal of Robotics Research, Vol. 16, No. 6, 759-774 (1997)

Micha Sharir, Algorithmic Motion Planning, in Handbook of Discrete and Computational Geometry Second Edition, CRC Press, Chapter 47, 2004

D. Halperin, L.E. Kavraki, J.C. Latombe, Robot Algorithms. In Algorithms and Theory of Computation Handbook, CRC Press, Chapter 21, pp. 21-1–21-21, 1999.

Y.K. Hwang, N. Ahuja, A potential field approach to path planning, IEEE Transactions on Robotics and Automation, Vol. 8, Issue 1, p.23-32