Proceedings of 16th Int. Workshop on Robotics in Alpe-Adria-Danube Region - RAAD 2007
Ljubljana, June 7-9, 2007

CONTINUOUS PATH FOLLOWING WITH APPLICATIONS IN DIFFERENT ROBOTIC TASKS

Theodor Borangiu
Andrei Nick Ivanescu
Anamaria Dogar
Alexandru Dumitrache
Andrei Rosu

University Politehnica of Bucharest,
Dept. of Automation and Applied Informatics
Bucharest, Romania

Abstract: This paper presents a theoretical approach and an implemented software program for a robot's end effector's path following. The reason for following a path may be avoiding obstacles or, more likely, moving the end effector along the contour of the part being processed in order to do various technological tasks (cutting, welding, painting, etc.). The complex path can be obtained in three ways: 1) by drawing the path in a CAD software package; 2) by using the robot's camera to acquire an image, and extract the path from it in real-time, without operator assistance; 3) by using a image acquired by some other means, and extracting the path from this image offline, before the robot program is executed. The implemented software is using algorithms for: contour extraction from binary images, polyline simplification, tool path reordering and optimization, drawing, scaling and fitting to paper. An application that uses the developed program in the process of making printed circuits boards by drawing the cable lay-up using a special marker, is presented in the final section of the paper.

Keywords: Robotics, Image Analysis, Education, Computer-Aided Design

1. INTRODUCTION

In some applications, the robot should not only reach a fixed number of pre-learned positions, but also should move along a complex path or trajectory. The reason for following a path by the end effector could be avoiding obstacles or, more likely, doing various technological tasks along the contour of a processed part.

Because a complex path has many points, it may be difficult to learn it using the teach pendant or by hard-coding each point manually in a robot program. There are the following alternatives:

1. Drawing the path in a CAD software package;
2. Using the robot's camera to acquire an image, and extract the path from it in real-time, without operator assistance;
3. Using an image acquired by some other means, and offline extracting the path from this image, before the robot program is executed.

In most cases, the first option may be preferred, because the path can be drawn with precision, as a CAD program lets the user specify the coordinates of the path segments. Moreover, a path drawn in a CAD environment is not constrained to lie in a fixed plane; it can be any 3D trajectory.

The second option should be used when the path is not precisely known at programming stage. From the image acquired by a camera it is possible to extract the contour of the objects and ask the robot to follow that contour, or to follow the path at a specified distance, so that a rounded-shape tool will move tangent to the object. This approach may be of little interest in industrial environments, because in such applications, the path is well-known from the design, but it will be a nice application for demonstrative and educational purposes.

The third approach allows the user to tweak the image offline until he/she gets the desired results. It

should be used when the path is not available in vectorial format, and the precision is not important. The path is also limited to a 2D plane.

Suppose a robot should follow a path made from basic elements: lines and arcs. It can be also considered more complex curves like splines, or math functions such as logarithms, exponentials, trigonometric functions and others. Because any continuous function can be linearized, i.e. approximated by small lines within a given tolerance, only straight lines were considered.

2. FOLLOWING A CONTINUOUS PATH DRAWN IN A CAD PACKAGE

If the end points of the segments are known the complex path following is resolved. It is discussed further the possibilities of extracting coordinates from a CAD drawing. There are several ways of doing this:

- Supposing the CAD package allows user-defined scripts or macros, we can write a script that iterates through all elements in the drawing, writing their coordinates to a text file, using a simple format like the one in the previous section. AutoCAD supports VBA (Visual Basic for Applications) and AutoLISP.

- Most CAD programs can save their drawings in DXF (Drawing Exchange Format). A DXF file is an ASCII file, and its structure is documented, so it is possible to write a parser that extracts the coordinates from these files.

- There are a lot of CAM (Computer Aided Manufacturing) software packages that create motion instructions for numerically controlled machines (CNCs). These programs output in ISO CNC language, also known as RS274 or G-Code. Because these programs usually output a small subset of the language, it will not be difficult to write a translator that converts G-Code to robot motion commands, or even an interpreter written in robot programming language that can execute G-Code files. It would also possible to write a post-processor plugin for a CAM program. The post-processor will save the tool path into a simple text file, which will be read by the robot.

In a CAD drawing, a path can be drawn as a polyline, which is a collection of simpler entities (lines, arcs and maybe spline curves) and which has a start point and an end point. The program that extracts coordinates from the polyline must linearize all the curves (arcs, splines).

3. FOLLOWING A PATH EXTRACTED FROM AN OFFLINE PROCESSED IMAGE. THE DRAWING ROBOT

Suppose having a binary image that should be drawn on paper using the robot. This is a very good educational application, and also a fun project, which show the students what a robot manipulator is capable of. The project will also allow analyzing the robot's accuracy and repeatability.

It is needed to extract the contours from the image and to tell the robot to move along these contours. For example, if the image of a gear is taken, the robot should draw it like this:

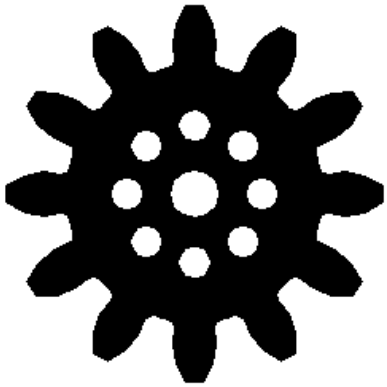

Fig. 1. Original binary image

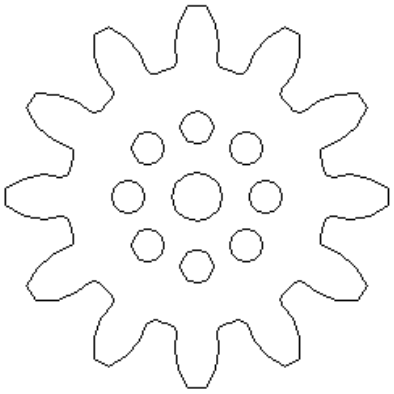

Fig. 2. Image to be drawn by the robot

There are algorithms used for:
- contour extraction from binary image
- polyline simplification
- toolpath reordering and optimization
- drawing scaling and fitting to paper.

3.1 Path optimization

In the contour following application, the order in which the contours are followed by the robot does not matter. It is optimal to reorder the contours so that the robot's motion between the paths is minimized.

This optimization problem is similar to the traveling salesman problem, which is NP-Complete, i.e. its solution cannot be found in polynomial time. For this reason, a simple heuristic algorithm that finds a good (but not optimal) solution is presented.

Every robot path can be considered as a polyline made from straight line segments, and is stored in the computer's memory as an array of vertices (points). Every polyline has starting point and an end point. Polylines may be open or closed.

Any polyline that has different start and end points is an open polyline. Open polylines with N vertices will be stored as an array of N vertices, where the first vertex in the array is the starting point, and the last vertex is the end point.

A polyline that has the same start and end points is a closed polyline. Closed polylines having N vertices will be stored as an array of N+1 vertices, with the first and the last vertices being the same. The start and the end point of a closed polyline are called the polyline's origin.

In order to minimize the robot motion between these polylines, the following actions can be performed:
- changing the order in which polylines are executed;
- reversing an open polyline, i.e. robot will start following it from its end point, backwards to its start point;
- changing the origin of a closed polyline;

Changing the direction in which the path is followed have no influence on the extra distance that the robot travels between polylines.

The following notations were used:
- $P = \{ p_1, p_2 \dots p_N \}$: a polyline represented by an ordered set of points (i.e. an array with N elements)
- reverse(P) = $\{p_N, p_{N-1} \dots p_1 \}$: a reversed polyline
- changeorigin(P, k) = $\{ p_k , p_{k+1} \dots p_{N-1}, p_1, p_2 \dots p_k \}$: a closed polyline with its origin changed to the vertex p_k . Note that p_k is duplicated, in order to show that the polyline is closed. Also, because p_N and p_1 are the same, only one of them is retained.
- $Q = \{P_1 , P_2 \dots P_M \}$: an array of polylines. The robot will follow P_1, then P_2, and the last polyline followed will be P_M.

We also used the Euclidean distance between two points.
The nearest neighbor algorithm:

Input: $Q = \{P^1 , P^2 \dots P^M \}$: an array of M polylines

Output: $R = \{ P^{1o} , P^{2o} \dots P^{Mo} \}$: an array of M polylines. The M polylines are built from the ones in Q; open polylines may be reversed, and closed polylines may have their origin changed. The order of polylines in Q and R may not be the same.

1. Choose a starting polyline (i.e. the leftmost): P^{1o} ; add it in R and remove it from Q;
2. $R = \{ P^{1o} \}$, $Q = Q - \{ P^{1o} \}$;
3. While Q is not empty:
    1. Let p_e be the end point for the current polyline.
    2. For each open polyline $P_k = \{ p_1^k, p_2^k \dots p_{Nk}^k \}$
        1. Compute $d_k^S = d(p_e, p_1^k)$ - distance from p_e to the start point of P_k
        2. Compute $d_k^E = d(p_e, p_{Nk}^k)$ - distance from p_e to the end point of P_k
        3. Compute $d_k = \min(d_k^S , d_k^E)$
    3. For each closed polyline $P_k = \{ p_1^k, p_2^k \dots p_{Nk}^k \}$
        1. Compute $d_k^i = d(p_e, p_i^k)$, $i = 1:N_k$ - distance from p_e to each point of P_k
        2. Compute $d_k = \min_i d_k^i$ and let $i_{min} = \arg\min_i d_k^i$
    4. Find the closest polyline P_j with respect to p_e – the one that has the smallest d_j.
    5. Remove P_j from Q
        $Q = Q - \{ P_j \}$
    6. Add P_j to the solution and update the end point p_e:
        1. If P_j is open:
            1. If $d_k^E < d_k^S$, P_j = reverse(P_j)
        2. Else (P_j is closed):
            1. P_j = changeorigin(P_j , i_{min})
        3. $R = R \cup \{ P_j \}$
        4. $p_e = p_{Nj}^j$

If the array insertions and deletions are made in O(1), the algorithm has a complexity of $O(NM^2)$ – where M is the number of polylines, and assuming every polyline has N vertices. For more speed, it's possible to skip changing origin for closed polylines; in this case, the complexity becomes $O(M^2 + MN)$.

3.2 Scaling and fitting the drawing to paper

In order to determine the size of the drawing executed by the robot, a millimeter to pixel ratio is needed to be determined. All the drawing elements were scaled using that ratio. In order that the drawing to be centered on the page, scaling was done in the following way:
Algorithm for scaling and centering the drawing on page:
Inputs: r = millimeter to pixel ratio
w, h: page size, in millimeters

$P = \{ p_i \}$: the set of all 2D points from which our drawing is made; $p_i = \langle x_i, y_i \rangle$, having coordinates expressed in pixels
Outputs: $Q = \{ q_i \}$: the points from P scaled and translated for centering the drawing on page.

1. Find the minimum bounding box for the drawing:
$$x_{\min} = \min_i x_i \, , y_{\min} = \min_i y_i$$
$$, x_{\max} = \max_i x_i \, , y_{\max} = \max_i y_i$$
2. Translate the drawing so that the center of the bounding box will be in origin
$$x_c = \frac{x_{\max} + x_{\min}}{2} \; ; y_c = \frac{y_{\max} + y_{\min}}{2}$$
$$p_i = \langle x_i - x_c \, , \; y_i - y_c \rangle \quad \text{for every } i$$

3. Scale the drawing using the millimeter to pixel ratio
$$p_i = \langle x_i * r \, , \; y_i * r \rangle \quad \text{for every } i$$

4. Center the drawing on page:
$$q_i = \langle x_i + w/2 \, , \; y_i + h/2 \rangle \quad \text{for every } i$$

By choosing different ratios, drawings of different sizes for the same object can be obtained. It would be a good idea to let the program determine automatically a ratio so that the drawing fits the page. The user have to specify a margin in which noting will be drawn. This is done by the following algorithm:

Algorithm for computing the millimeter-to-pixel ratio for fitting a drawing to a page
Inputs:
w, h: page size, in millimeters
m: desired margin, in millimeters
$P = \{ p_i \}$: the set of all 2D points from which our drawing is made; $p_i = \langle x_i, y_i \rangle$, having coordinates expressed in pixels
Output: r: the computed millimeter-to-pixel ratio

1. Find the minimum bounding box for the drawing:
$$x_{\min} = \min_i x_i \, , y_{\min} = \min_i y_i \, ,$$
$$x_{\max} = \max_i x_i \, , y_{\max} = \max_i y_i$$
$$w_d = x_{\max} - x_{\min}; \; h_d = y_{\max} - y_{\min}$$

2. Compute the ratio
$$w_p = w - 2m$$
$$h_p = h - 2m$$
$$r = \min(w_p / w_d, \; h_p / h_d)$$

3.3 User Interface

The above algorithms were implemented in a program that lets the user open an image and computes the robot path for drawing the image's contour. The user's operation is described as follows:

1) The user opens the input file, which is a bitmap image.

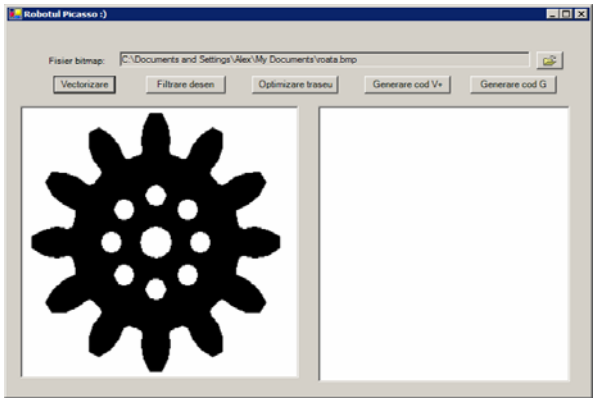

Fig. 3. Loading the image to be drawn

2) The contour extraction window: at this step, the image contour is detected with Moore-Neighbor algorithm.

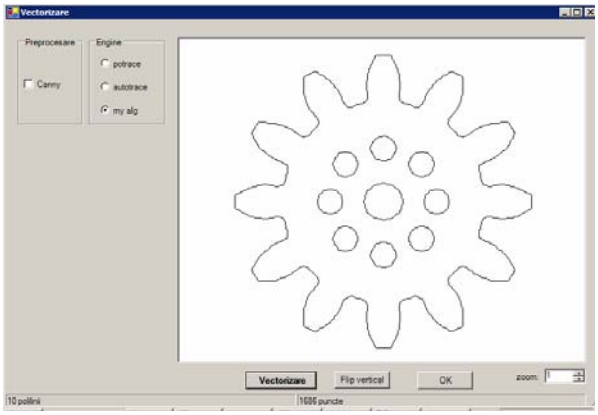

Fig. 4. The contour extraction step

3) Polylines are filtered using the Douglas-Peucker algorithm, using a tolerance set by user. Also, polylines that have the total length less than a specified value are removed.

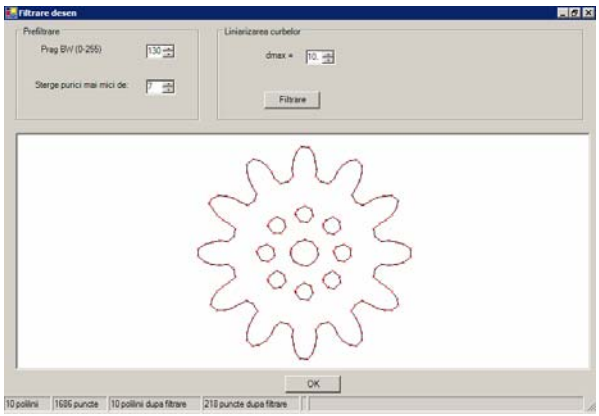

Fig. 5. Filtering the polylines

4) Polylines are optimized in order to minimze robot path:

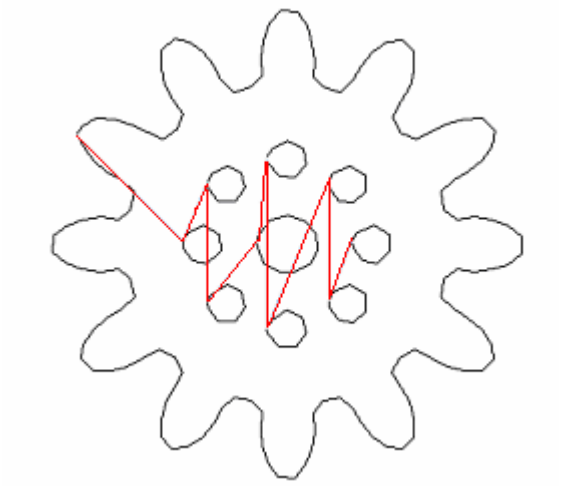

Fig. 6. Initial (unoptimized) polylines

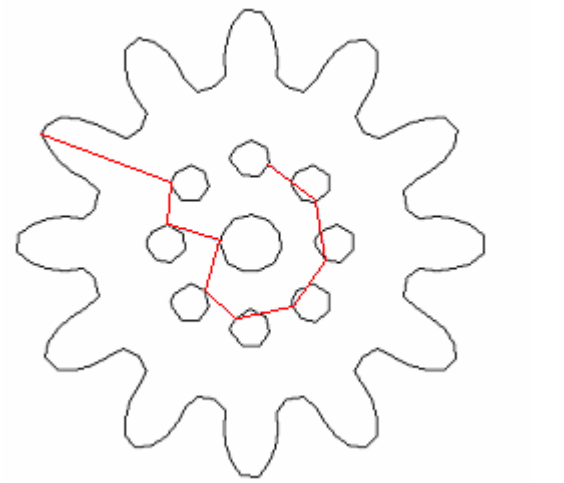

Fig. 7. Optimized polylines

5) User enters paper size, and various settings for executing the drawing.

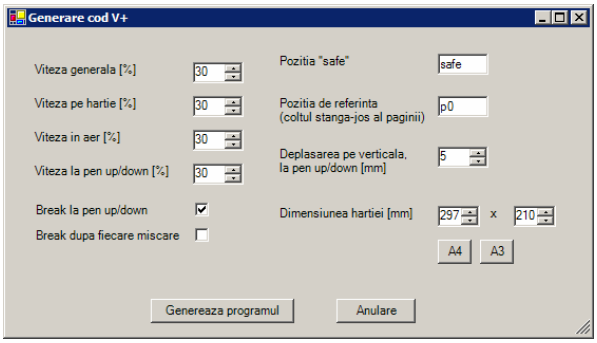

Fig. 8. Setting paper size and various parameters for robot motion

6) The program writes the file with the robot's path.

7) User executes the V^+ program that follows the path.

4. MAKING PRINTED CIRCUITS BOARD

The technological process of making printed circuits boards includes the chemical process of corrosion of the copper with ferric chloride. Initially any circuit is a textolite board with a thin copper folium. The connecting paths between the electronic components are printed on this board. This printing is done in order to protect the copper.

The printing is done using a special marker. The paint left by the marker will ensure the protector layer necessary for the interruption of the chemical reaction and preserving the paths. The marker is mounted in the end effector of the robot, and moved on a computed path.

After printing the board is eroded. The result of this stage is a board only with the printed path. The cleaning of the printing paint is done using isopropyl alcohol.

The next stage consists in drilling the electronic board in order to fix the electronic parts. This stage is done with a milling CNC machine. The holes coordinates are identified on an image in which they are marked with + symbol. The last stage is the mounting of electronic parts on the resulted PCB.

The image of the PCB is available in bitmap format. Using the path following algorithm there has been developed a new algorithm that generates the path that the marker will follow. The algorithm considers also the width of the marker and hatches the larger area of the PCB.

In the next figure the image of a test circuit with a resolution of 300 dpi is presented, the path represented with white colour is generated for a 10 pixels (0.85 mm) marker.

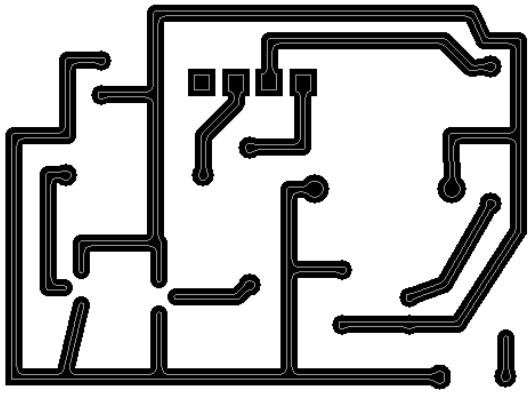

Fig. 9. The test circuit and the marker path

The algorithm can hatch the larger area of the PCB, when contour following is not enough for entire covering. The hatching is done by eliminating the contour covered in the precedent step of the

algorithm and applying the same algorithm until the area is covered.

The algorithm has been tested using the Adept Cobra 600 robot and the results are shown in the figures below.

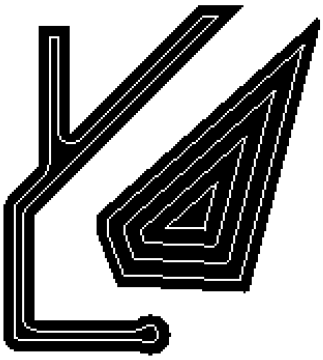

Fig. 10. The algorithm hatches the larger areas of the PCB

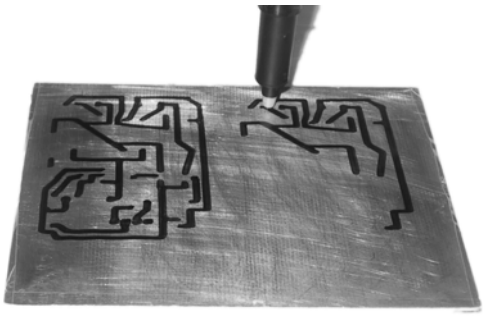

Fig. 11. The robot drawing a cable lay-up

In the case of the PCB with the line thinner than the existing marker a skeletonisation algorithm is used. The vectorization program can decide in which case is necessary to use one of the two implemented algorithm.

The two images can be obtained using dedicated designing programs like Orcad, Proteus.

CONCLUSIONS AND FUTURE WORK

This work has been done in the Robotics and AI Laboratory of the Faculty of Automatic Control and Computers. For the experiments, it was used a SCARA robot, Adept Cobra 600 TT. Future work includes using better algorithms for polyline simplification and testing real-time contour following procedures. A visual inspection procedure of the drawn cable lay-up will be also developed.

REFERENCES

Borangiu, Th. (2004). *Intelligent Image Processing in Robotics and Manufacturing*, Publishing House of Romanian Academy, Bucharest

O'Gorman, L. (1996). Subpixel Precision of Straight-Edge Shape for Registration and Measurements, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, **18**, *11*, 57-68