A New Trajectory Generation Method Using Vision for a Robot Manipulator

[1] J.A. Soto, [2] J. E. Vargas y [1] J.C. Pedraza

[1]Centro de Ingeniería y Desarrollo Industrial, Unidad de Investigación y Postgrado
[2] Universidad Anáhuac – México Sur, Facultad de Ingeniería

Abstract

In this work a new proposal method is presented to generate close paths for a robot manipulator. Through a specific image, which contains an object, an image processing is applied to obtain the boundaries of the image, with this boundary and applying the inverse kinematics the path that the robot should follow, is created, so that finally the path can be simulated and created in a five degrees of freedom manipulator.

1. Introduction

The robotics, artificial vision and their applications are some of the greatest fields for research. Investigations on this fields promise advanced developments and novelties in many aspects. Applications of projects which combine robotics with artificial vision are more easily found nowadays, they are also more interesting and complex.

The main idea of this job is to generate a practical application, using a five degrees of freedom robot manipulator and a digital camera. Within the possible applications there are for example: cutting, brazing and drawing. The generation of the path could be extrapolated to any other kind of robot or machine that can follow the path. The steps to follow during the development of this job are the next ones:

- Obtain an image and realize the preprocessing.
- Generate the path of the object over parametric curves using cubic schemers (splines) to soften the boundary.
- With the inverse kinematics and the parametric curves, generate the path for the manipulator robot.
- Realize the simulation of the path tracking, using OpenGL libraries.

2. Path Generation

There are two basic ways for programming the path in a manipulator robot, the explicit programming and the implicit programming.

The explicit Programming divides itself in gestual programming and textual programming. In the gestual programming the robot is taught by taking it to the interest position, keeping it and defining speed and acceleration, this type of programming is also called on-line programming, many times is necessary to stop the process in wich the manipulator will act to be able to program it. In textual programming tha calculus for the inverse kinematics are made carefully, to generate the path, considering position, speed, acceleration and workspace, this kind of programming is also known as out-line programming and it is not necessary to stop the process in which the manipulator will act.

The implicit programming is when the robot has pre-defined tasks and with the help of a sensors system and the correct programming, it makes the robot to seem intelligent. This programming falls into artificial intelligence and is the most advanced in what manipulators programming refers to.

2.1 Obtaining the image and preprocessing

The image processing is basic but not less important in this work. The process of capture of image and its processing have a great number of methodologies and algorithms hardly studied. The knowledge of the application and how to use these tools are the key to have a successful and functional result.

First the image is obtained with any camera, if possible, having a good lighting and a good contrast between the background and the desired object (figure 1, real image). The image is transformed into a gray scale, if it is necessary. After this, it is binarized and finally the boundary is generated, storing the data in two vectors(x,y), which represent the real path of the image. In figure 1 the results of this process are showed.

The algorithm to follow the boundary, is an own contribution, it consists en assigning a searching address (clockwise), search for a start point, follow each one of the boundary points and store them in a pair of vectors (x,y), which represent us the location on the plane and are necessary to obtain the splines.

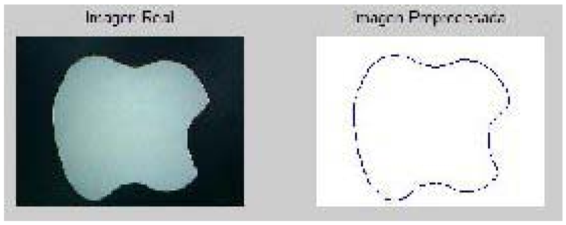

Figure 1. Processing Results

2.2 Boundary Softening with Splines

Once the boundary of the images is obtained in vectors (x, y), the boundary may have non-desired irregularities, because of the defects in the image obtaining or during the preprocessing, which may have some noise, that is why sometimes is necessary to soften the boundary of the image.

An option to soften the image is using cubic schemers, which represent the path. The process to soften the path includes knowing the complexity of the same, selecting the points which best represent the desired boundary and applying the cubic schemers.

In figure 2, the selected points to soften the boundary are represented. For this example the initial length of the vector is 1339 points, 35 points are selected and it is how the points can be observed in the same figure 2, the results are interesting.

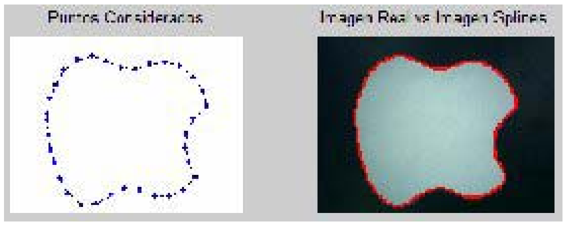

Figure 2. Results after softening

The splines not only help us softening the boundary, but also, it reduces the memory size used and the most interesting, because there are cubic polynomials, the first and second derivative for each one of the points can be obtained, this makes easier the job to obtain the inverse kinematics for the robot.

2.3 Position in the workspace

After the obtaining the path from the image, the next step is locate it in the workspace of the robot, for this case the path is 2D, that is why it is located in the xy plane at a constant altitude z=300 mm, (scaling the PIXELES of the image in millimeters). In figure 3 the path located in the workspace of the manipulator is showed.

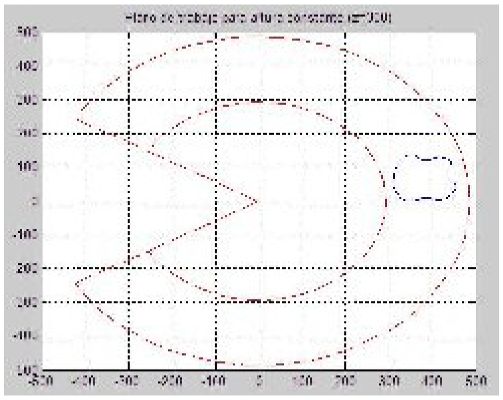

Figure 3. Path in the workspace.

2.4 Kinematics of the Manipulator

To develop the kinematics of the manipulator there are used homogeneous transformed matrixes, figure 4 shows an image of the robot. The robot is drawn on OpenGL and the manipulator Mitsubishi Melfa RV-2AJ was taken as a base form. The robot counts with 5 degrees of freedom, the five degrees are rotational and each one has their rotation restrictions, these restrictions and the speeds are showed in figure 5.

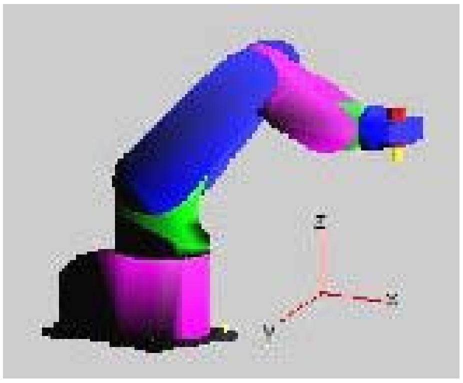

Figure 4. Robot Configuration

Table 1. Restrictions for the Manipulator

Articulación	Limite de Rotación Deg.	Velocidad Máxima Deg./s
q1	-150 a +150	180
q2	-60 a +120	90
q3	-110 a +120	135
q4	-90 a +90	180
q5	-200 a +200	210

The inverse kinematics, for this example, simplifies because the tool direction on the plane is constant on the negative z axis direction, therefore the direction of the 4th link will be always orthogonal to the z axis, simplifying the kinematics considerably. But it is not always like that, that is why it is necessary to consider all the transformation matrixes. Continuously the transformation matrixes are showed for the direct kinematics, represented by *A*, with a sub index that represents the axis for initial coordinates and a super index that represents the axis for final coordinates.

$$A_0^1 = \begin{bmatrix} \cos(q1) & 0 & sen(q1) & 0 \\ sen(q2) & 0 & -\cos(q1) & 0 \\ 0 & 1 & 0 & L1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (1)$$

$$A_1^2 = \begin{bmatrix} \cos(q2) & -sen(q2) & 0 & L2\cos(q2) \\ sen(q2) & \cos(q2) & 0 & L2sen(q2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (2)$$

$$A_2^3 = \begin{bmatrix} \cos(q3) & -sen(q3) & 0 & L3\cos(q3) \\ sen(q3) & \cos(q3) & 0 & L3sen(q3) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (3)$$

$$A_3^4 = \begin{bmatrix} \cos(q4) & 0 & sen(q4) & 0 \\ sen(q4) & 0 & -\cos(q4) & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (4)$$

$$A_4^5 = \begin{bmatrix} \cos(q5) & -sen(q5) & 0 & 0 \\ sen(q5) & \cos(q5) & 0 & L4 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (5)$$

$$A_5^6 = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & L5 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (6)$$

The direct kinematics is obtained by multiplying each one of the transformation matrix, to obtain a matrix which goes from the coordinates 0 axis to the coordinates 6 axis, presented as $A0^6$, as the formula number 7 shows. The direct kinematics is necessary to proof the calculus of the inverse kinematics.

$$A_0^6 = T = A_0^1 A_1^2 A_2^3 A_3^4 A_4^5 A_5^6 \qquad (7)$$

For the calculus of the inverse kinematics, first the path that the robot is going to follow needs to be known, which we know. In function of the path the orientation of the coordinates 6 axis (or the tool) relatively with the coordinates 0 (the base), is oriented together with each point of the path position (px, py, pz), witch form the matrix of the desired transform Td which is showed in formula number 8.

$$T_d = \begin{bmatrix} nx & ox & ax & px \\ ny & oy & ay & py \\ nz & oz & az & pz \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (8)$$

Where the vectors *n*, *o* and *a* represent the orientation of the coordinates 6 axis relatively to axis 0 and the vector p the position in the space equally referenced to the 0 axis. To solve the inverse kinematics it is just enough to equate the desired transformation matrix with the original matrix, Td=T, and solve for the angles q1, q2, q3, q4 and q5. As it has been mentioned, this simplifies, because the tracking of that path, the orientation of the tool remains the same.

Once we know the angles, the maximum speeds need to be considered for each articulation and, in function of this speeds, take the robot from its position start point to the path start point, taking care of collisions. In the same form the maximum speed of path tracking is in function of the articulations speeds.

Figure 5 shows the table of maximum speeds for the robot for each articulation and figure 6 shows a block diagram, where the steps to achieve the simulation are showed.

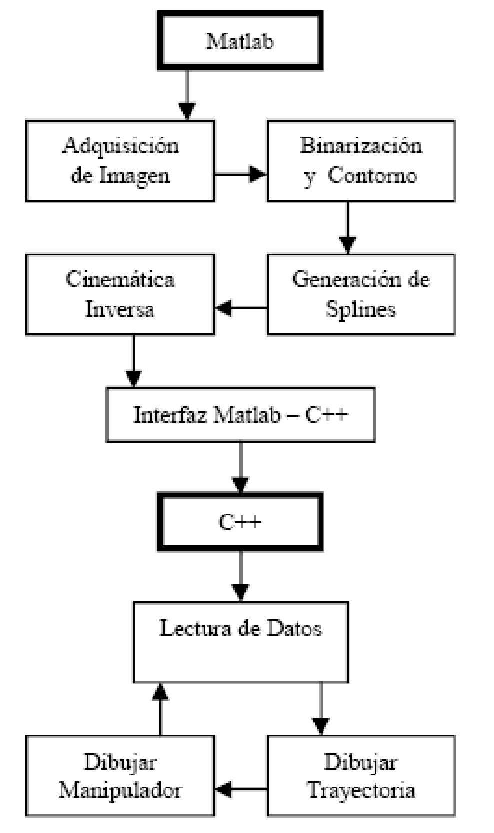

Figure 6. Block Diagram of the system

3 Analysis of the results

For some engineering designs and normal calculus is necessary to manage matrixes. In this proposal we manage matrixes for the image processing and for the manipulator kinematics. Matlab has a great number of functions for matrixes operations; besides, there are many different ways to realize interfaces between Matlab and different languages, such as C++.

Using these tools the simulation over C++ was developed. First all the calculus necessary for the Matlab are done, to finally simulate with the OpenGL libraries.

The results, after running the simulation are very interesting. Figure 6 shows an instant in which the robot is tracking this path, the simulation was done over C++ with OpenGL libraries.

In figure 7, the same analysis is done for a different figure, a bit more complex and just as the previous one, there was no problem to track the path.

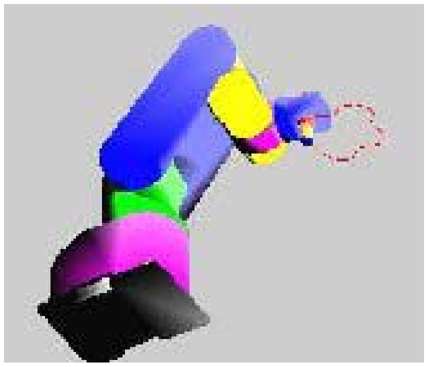

Figure 7. Path tracking

In this form almost all kinds of path in 2D can be tracked, as long as it is inside the workspace of the manipulator

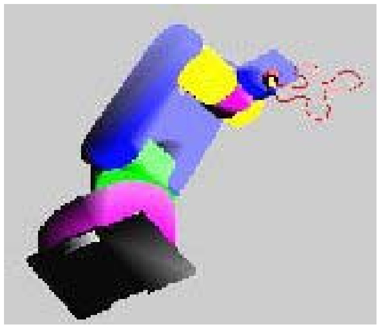

Figure 8. Path tracking

4 Conclusions

The advances of this work, until the day the article was written, are promising and it is expected that into the practice, with the real robot, these results are checked.

As long as the article was written, it could be observed that with a simple image processing along with a mathematical tool, such as the splines, was a powerful result for soft path generation for a manipulator robot.

Since the kinematics of the manipulator is in function of the splines obtained from the processing, and since the splines are cubic polynomials they can be treated without big problems, the next point that this work is pretending to develop, is the tracking of the same paths, but this time facing the tool to the normal direction to the tangent of each point of the path, which modifies and makes the manipulators kinematics more complex.

Finally it is desired to work in the intersection, capture and manipulation of motion objects, initially over a transporting band (at an unchanging height) and after that scaling it to any point in the manipulator's workspace.

References

[1] Myler H, Weeks A, *"Computer Imaging Recipes in C"*, Prentice-Hall, USA, 1993.

[2] Barrientos A, Balaguer C, *"Fundamentos deRobótica"*, Mc Graw Hill, 1st Edition, España, 1997.

[3] Neider J, Davis T, *"ReedBook"*, Addison Wesley, 2nd Edition, USA, 1997.

[4] Wright R, Sweet M, *"OpenGL Super Bible"*, Waite Group Press, 2nd Edition, USA, 1999.

[5] Bay Y, *"Applications Interface Programming Using Multiple Languages"*, Prentice Hall, USA, 2003.