GENERATE THE COMPLEX TRAJECTORY THROUGH DIRECT FUNCTION COMPUTATION INTERPOLATING ALGORITHMS

Liliana Marilena Matica, Horea Oros

Faculty of Electrotechnics and Informatics, University of Oradea, lmatica@uoradea.ro

Faculty of Sciences, University of Oradea, horos@uoradea.ro

Abstract Formulas and implements methods of those formulas to generate the points co-ordinates from a complex trajectory are analyzed. The interpolation algorithms analyzed are named ,,direct function computation". Those algorithms can implement a feedback command to traverse the complex trajectory considering the current position of the robot to determine the next position. The implemented programmes have a real time evolution.

1. INTRODUCTION

There are industrial robots with trajectory that is important all time movement, e.g. industrial robots for painting or welding. Therefore it is necessary a compromise between the numbers of external information and required internal information to traverse the trajectory. The compromise is accomplish by decomposing the trajectory in simple, circular or linear segments. The co-ordinates of the segment extremity are the external information. The intermediary points co-ordinates are computed by interpolating along traversing the trajectory.

2. PROGRAMME LANGUAGE MOVEMENT COMMANDS FOR INDUSTRIAL ROBOTS

The interpolating module is called when the trajectory is specified all time movement. The types of movement for an industrial robot can be classified as: 1 point to point - unimportant trajectory, only the final position is important, such point welding or manipulation of the objects; 2 with articulation interpolating - this type command the movement for each articulation, the resulted trajectory is unknown in the working space of the industrial robot; 3 with Cartesian interpolating - the trajectory is exact and it is defined in a Cartesian space Oxyz with three axes. The trajectory co-ordinates are converted

in movement for each articulation using the geometric model of the industrial robot.

The programming language of the industrial robots VAL II has instruction that select one of this type of movement described. So [1] there are the instructions:

$$MOVE \ < location >$$

to define a point to point movement with articulation interpolating between current position and final position specified in the instruction by the argument;

$$MOVES \ < location >$$

the movement trajectory is a linear one, to the final position specified in the instruction by the argument;

$$APPRO \ < location >, \ < distance >$$

the programmed movement is with articulation interpolating to the specified position, the Oz axis movement is specified by argument distance;

$$APPROS \ < location >, \ < distance >$$

is the Cartesian version of the previous instruction;

$$DEPART \ < distance >$$

in Oz axis direction movement with an articulation interpolating;

$$DEPARTS \ < distance >$$

is the Cartesian version of the previous instruction. The language LM defines two types of movement: the free movement with a non-defined trajectory and the Cartesian movement with Cartesian interpolating. Such instructions are [1]:

$$MOVE \ CUBE \ VIA \ POS1 \ CARTESIAN, \ POS2, \ POS3 \ TO$$

$$PUT\_CUBE \ CARTESIAN \ WITH \ SPEED = 0.8;$$

Those successive specifications command a Cartesian interpolating movement from the initial position, named $POS1$, then point to point movement without defined trajectory, and, at the end, a Cartesian interpolating movement from $POS3$ to PUT_CUBE.

Other programming language, also final effect movement oriented, contain diverse instructions regarding the movement. The $ROBEX - M$ language

contains an interesting instruction that programmes a movement with the same orientation of the tool [1]:

$$COMOVE$$

constant orientation $MOVE$.

The language $ROBEX$ defines relative movement regarding the current position of linear three axes Oxyz or circular axes associates with the linear axes [1], so there are instructions:

$$GODLTA/\ dx,\ dy,\ dz\ [, EVENT,\ a[,\ ELSE,\ m]]$$

go to a specified distance,

$$TURN/XYROT, ww1, [XYROT, ww2, [XYROT, ww3]][, EVENT, a[ELSE, m]]$$

$$TURN/YZROT, ww1, [YZROT, ww2, [YZROT, ww3]][, EVENT, a[ELSE, m]]$$

$$TURN/ZXROT, ww1, [ZXROT, ww2, [ZXROT, ww3]][, EVENT, a[ELSE, m]]$$

Language ACL programmes a movement according to linear or circular trajectory through $MOVEL$ and $MOVEC$ instructions.

3. INTERPOLATING ALGORITHMS FOR DIRECT FUNCTION COMPUTATION

Some of the most efficient interpolating algorithms are the direct function computation algorithms [3]. Those algorithms work with trajectory analytical expression, the line analytical expression for a line segment and the circle analytical expression for a circle arc. Algorithms are based on the trajectory propriety that the points on one side of trajectory give a sign to analytical expression and the points on other side give the opposite signs to the same trajectory analytical expression. The programmed movement must give alternative sign to the trajectory analytical expression.

For a Cartesian space trajectory we must analyze the sign of the expression $sign(\frac{\partial f}{\partial x}dx + \frac{\partial f}{\partial y}dy + \frac{\partial f}{\partial z}dz)$, where the trajectory is defined by the equation $z - f(x, y) = 0$ *or* $F(x, y, z) = 0$.

It is easy the analyze the algorithms for a Cartesian plane movement but the principle is the same for a Cartesian space movement.

The formulas, for current point $P(x_i, y_i)$ deviation computation, are:

- for a step δx on Ox direction $F(x_i + \delta x, y_i) - F(x_i, y_i) = \Delta F(x)$;
- for a step δy on Oy direction $F(x_i, y_i) - F(x_i, y_i + \delta y) = \Delta F(y)$;
- for a step on Ox and Oy directions $F(x_i + \delta x, y_i) - F(x_i, y_i + \delta y) = \Delta F(x, y)$.

OCTANT	1	2	3	4	5	6	7	8
direction	u x	y	y	-x	-x	-y	-y	x
direction u+v	x+y	x+y	-x+y	-x+y	-x-y	-x-y	x-y	x-y

Table 1 The change of variable for linear interpolating octant algorithm.

The sign analysis is necessary in order to determine next point co-ordinates $P(x_i + 1, y_i + 1)$. For a linear segment with analytical trajectory equation $ax + by + c = 0$ the steps deviation can be determined by formulas

$$\Delta F(x) = a\delta x,\ \Delta F(y) = b\delta y,\ \Delta F(x, y) = \Delta F(x) + \Delta F(y).$$

For space trajectory movement it can be considered a simplification that commands a linear or circular plane Oxy trajectory movement and a constant movement Oz-axis direction. For a circle segment, an arc with analytical equations $x_2 + y^2 - R^2 = 0$ the step deviation are

$$\Delta F(x) = 2x_i\delta x + \delta x^2,\ \Delta F(y) = 2y_i\delta y + \delta y^2,\ \Delta F(x, y) = \Delta F(x) + \Delta F(y).$$

The direct function algorithm is: for a linear rising analytical trajectory function, a positive deviation demand a step to Ox-axis direction, a negative deviation a step to Oy-axis direction. For a circle segment, trigonometric sense, a positive deviation demand a step to Ox-axis direction. The step dimension is defined by the speed value. For any kind of analytical trajectory function the algorithm can work. It must determine the sign of the function for points on one side of the trajectory by considering one example of co-ordinates point. The alternating sign principle determines the step demanded for going to the side with opposite sign. This method is very useful, it implements a real time feedback of the movement command. The next command is computed based on the current point co-ordinates, irrespewctive the fact the current point co-ordinates are the precedent computed point co-ordinates or not. The algorithm can be improved by considering the sign of a simplified expression, named a „discriminant". For a linear segment, with the final point $F(x_F, y_F)$ the algorithm must analyze the sign of the expression $D_D = x_F y_i + y_F x_i$. For a circular segment the „discriminant" is $D_C = x_i^2 + y_i^2 - (x_0^2 + y_0^2)$, where (x_0, y_0) are the co-ordinates of the initial point considering the circle center as the origin of the axes. In order to simplify the computation it is recommended to make a change of variables such that for a linear segment we consider a new origin on the beginning segment point and for circular segment a new origin on the circle center.

Working performance have named the interpolating algorithm method of octants [2]. For linear interpolating the algorithm considers two movement

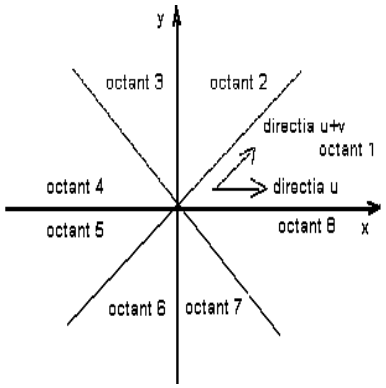

Figure 1 Octants definition.

OCTANT	1	2	3	4	5	6	7	8
direction v	y	-x	-x	-y	-y	x	x	y
direction u+v	-x+y	-x+y	-x-y	-x-y	x-y	x-y	x+y	x+y

Table 2 The changes of variables for circular interpolation, octants methods.

direction u and u+v according with the change of variable presented n Table 1, depending on the number of octant where the linear segment is placed.

The algorithm analyzes the sign of expression $F(u,v) = u_F v_i - v_F u_i$. The algorithm is: positive deviation demand direction u movement command. For opposite sign the movement command is to u+v direction. The sense and the step value are defined according to the definition of the octant, (fig. 1). There are defined eight octants. Circular interpolation, trigonometric sense works with change of variables presented in Table 1.

4. COMPUTATION EXAMPLES

The step value for each sample period can be computed depending speed value. The speed variation at the beginning and the end of the movement is programmed according to inertia reasons. Table 3 shows movement commands for a linear segment $A(0,0)$ $B(5,6)$, „discriminant" method. The step value is 1.

The same linear segment can be interpolated with octant method, the commands are described in Table 4. The linear segment is in octant 2, so $u_F = y_F = 6,\ v_F = x_F = 5$. For the two methods see fig. 2.

A circular interpolation example, method "discriminant" is analyzed in Table 5. The beginning point is $A(13,0)$ and the end point is $B(5,12)$.

NR.	D=	C-DA U = Y	C-DA V = X	OX	OY
1	0	*		0	0
2	-6		*	1	0
3	-1		*	1	1
4	4	*		1	2
5	-2		*	2	2
6	3	*		2	3
7	-3		*	3	3
8	2	*		3	4
9	-4		*	4	4
10	1	*		4	5
11	-5		*	5	5
12	0	stop		5	6

Table 3 Computation example, linear interpolation, "discriminant" method.

NR.	D=	C-DA U = Y	C-DA V = X	OX	OY
1	0	*		0	0
2	-5	*	*	0	1
3	-4	*	*	1	2
4	-3	*	*	2	3
5	-2	*	*	3	4
6	-1	*	*	4	5
7	0	stop		5	6

Table 4 Computation example, linear interpolation, octants method.

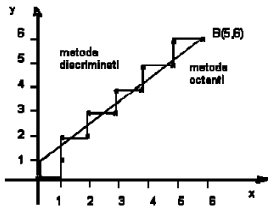

Figure 2 Linear interpolation, examples.

NR.	D=	C-DA OX	C-DA OY	OX	OY
1	0	*		13	0
2	-25		*	12	0
3	-24		*	12	1
4	-21		*	12	2
5	-16		*	12	3
6	-9		*	12	4
7	0	*		12	5
8	-23		*	11	5
9	-12		*	11	6
10	1	*		11	7
11	-20		*	10	7
12	-5		*	10	7
13	12	*		10	9
14	-7		*	9	9
15	12	*		9	10
16	-5		*	8	10
17	16	*		8	11
18	1	*		7	11
19	-12		*	6	11
20	11	*		6	12
21	0	stop		5	12

Table 5 Computation example, circular interpolation, "discriminant" method.

The same circular segment can be generated by octants method. At the beginning, the arch is in the second octant.

Beginning with the 11th step, the arch is in the first octant, the variables must be changed.

In these examples, the current co-ordinate was considered the computed coordinate, while in real mode utilization, the current coordinate must be read by transducers. The analyzed function sign is computed with the aid of these current values of the coordinates.

5. CONCLUSION

The direct function computation interpolating methods are simple methods, that have the advantage to implement a feed-back adjust of the movement on a complex trajectory. That adjust is the effect of the direct function sign computation using the current point coordinates to determine the next step in the movement. From all these algorithms the octants algorithm is more indicated because it determines the speed variation less than 9% .

NR.	D=	C-DA	C-DA Y = V	OX	OY
		X = U	Y = V		
1	0		*	13	0
2	1	*	*	13	1
3	-21		*	12	2
4	-16		*	12	3
5	-9		*	12	4
6	0	*	*	12	5
7	-12		*	11	6
8	1	*	*	11	7
9	-5		*	10	8
10	12	*	*	10	9
		X = V	Y = U		
11	12	*		9	10
12	-5	*	*	8	10
13	1	*		7	11
14	-12	*	*	6	11
15	0	stop		5	12

Table 6 Circular interpolation, octants method.

References

[1] Lecocq, H., *Langage de programmation de la robotique*, Univ. Liege, 1998.

[2] Matica, L. M., Abrudan-Purece, A., *Asupra implementării generării traiectoriilor complexe prin algoritmi de interpolare*, Al XV-lea Simpozion Naţional de Robotică, ROBOTICA 2000, Oradea, 2000, 225-228.

[3] Matica, L. M., *Sisteme informatice industriale*, Universitatea Oradea, 2001.

[4] Stoicu-Tivadar, L., *Programarea roboţilor industriali şi a maşinilor unelte cu comandă numerică*, Universitatea ,,Politehnica" Timişoara, 1996.