VOL 1 (2017) NO 4

e-ISSN : 2549-9904
ISSN : 2549-9610

INTERNATIONAL JOURNAL ON INFORMATICS VISUALIZATION

Specific Language for Robot Trajectory Generation

Kaloyan Yankov #

Faculty of Technics and Technologies, Trakia University, 8600 Yambol; 38 Graf Ignatiev str., Bulgaria
E-mail: kaloyan.yankov@trakia-uni.bg

***Abstract*— In this paper, a programming language for describing trajectories of the Mover 4 educational robot is discussed. The goal is to overcome the limitations of the programming tools provided by the manufacturer. Object-oriented structures of trajectories in the joint space and three-dimensional space are formulated. The model of the trajectory in the joint space is represented by the value of the joint, its velocity and acceleration, and the inertial tensor of the configuration from the respective joint to the end-effector. The inertia tensor is necessary to calculate joint forces and moments. A point from the trajectory in three-dimensional space is defined by the Cartesian coordinates of the end-effector, its orientation with the Euler angles and its velocity. Language offers spatial primitives to describe trajectories formed by segments, circle arcs, and cubic splines. Each primitive has a method of generating intermediate points. The language will allow the study of kinematic and dynamic capabilities in tracking trajectories.**

Keywords— **kinematics, robotics, robot language, robot path, simulation**

I. INTRODUCTION

The Robot Mover4, Commonplace Robotics GmbH, is a four joint robot for use in education, entertainment and research environments [1]. The accompanying CPRog software allows to control the robot. CPRog offers two editors to program the movement of the robot. The first is a graphical editor 'GraphEdit'. It is a puzzle-oriented programming language. It is addressed at users who have no basic knowledge of programming languages and programming. The second option is 'TextEdit'. This is a tabular programming language. Regardless which of the two programming tools is selected, the options for motion generation are equivalent. The advantage of TextEdit is that it allows the creation of larger programs than GraphEdit. The programs are saved on an HDD in an XML file. With the language the following operators can be set:

- ✓ Straight line motion from the current position to the target point. The end-effector velocity is defined in mm/sec.
- ✓ Interpolation of the axes from the current position to the target position in joint coordinates. Velocity is defined as percentage of the maximum joint rotational velocity.

These options are too poor because they restrict the robotics training process - it is not possible to demonstrate fully kinematic and dynamic robot control. Control functions can be enhanced by creating a language for describing the robot trajectories. Many programming languages are known - specifically designed for this purpose or as an upgrade of an existing procedural language [2], [3]. The use of a universal programming language and its upgrading with appropriate data structures and methods is the easiest way, as there is no need to create a language compiler.

The purpose of this work is to formulate a language for describing a limited class of trajectory paths on a MOVER 4 robot. The object-oriented graphics upgrade of Borland Delphi will be used [4], and the graphics primitives expanded with data structures specific of the robot trajectory. The generated motion program will be interpreted in an XML file to be read and executed by the robot. Such language will allow the study of kinematic and dynamic capabilities in tracking trajectories.

II. METHODS AND ALGORITHMS

A. Kinematic Model of Robot

The robot manipulation system (MS) consists of successively connected kinematic pairs which form an open kinematics chain (OKC) with a first link fixed to a selected coordinate system and the last link, end-effector (EE) that performs the desired target movements. Kinematic pairs are the independent parameters that uniquely define the position of MS in space. They are called generalized coordinates and define the vector:

$$q = \left[q_1, q_2, \ldots q_n\right]^T \text{, n-number of joints} \qquad (1)$$

q is called configuration of the MS.

The vector of the generalized velocities is:

$$\dot{q} = \left[\dot{q}_1, \dot{q}_2, \dot{q}_3, \ldots, \dot{q}_n\right]^T \qquad (2)$$

The generalized accelerations are:

$$\ddot{q} = \left[\ddot{q}_1, \ddot{q}_2, \ddot{q}_3, \ldots, \ddot{q}_n\right]^T \qquad (3)$$

The change of generalized coordinates $\boldsymbol{q}$ in accordance with the constructive limits determines the range of permissible configurations:

$$\left| \begin{array}{l} Q_0 = \left\{ q : q_i^{\min} \le q_i \le q_i^{\max} \right\} \\ Q_1 = \left\{ \dot{q} : \dot{q}_i^{\min} \le \dot{q}_i \le \dot{q}_i^{\max} \right\}, \quad i = 1,2,\ldots,n \\ Q_2 = \left\{ \ddot{q} : \ddot{q}_i^{\min} \le \ddot{q}_i \le \ddot{q}_i^{\max} \right\} \end{array} \right. \qquad (4)$$

Equations (1) - (4) define the kinematic model of the robot. The kinematic model is necessary to solve the inverse kinematics and to control the movement of the manipulation system to perform the technological operation. The coordinates $\boldsymbol{M}$ of the end-effector characteristic point are given by the equation:

$$M\left(x, y, z, \alpha, \beta, \gamma\right) = F\left(q\right) \qquad (5)$$

where:

F - continuous nonlinear function;

$\left(x, y, z\right)$ - the position in Cartesian coordinates of the end-effector;

$\left(\alpha, \beta, \gamma\right)$ - the orientation in Euler angles of the end-effector.

Equation (5) defines the forward kinematics of the robot. The solution of forward kinematics for the continuous set $\boldsymbol{Q_0}$ defines workspace $\boldsymbol{D}$ of the EE:

$$D = \left\{ M : M = F\left(q\right), q \in Q_0 \right\} \qquad (6)$$

The inverse kinematics problem consists of determining the generalized coordinates $\boldsymbol{q_G}$ where the end-effector will reach goal coordinates $\boldsymbol{G}$ in $\boldsymbol{D}$:

$$q_G = F^{-1}\left(G\right), \quad G\left(x, y, z, \alpha, \beta, \gamma\right) \in D \qquad (7)$$

In general, the inverse kinematics has no unique solution. For MOVER 4, the solution is implemented in the accompanying CProg software and therefore will not be the subject of the current work.

B. Trajectory Model in Joint Space

The instantaneous configuration of the MS is determined by the following data:

- Values $\boldsymbol{q}$ of joint coordinates - determine the Cartesian position of the EE in $\boldsymbol{D}$ - Eq.(5);
- Joint velocities and accelerations $\left(\dot{q}, \ddot{q}\right)$ - determine the velocity and acceleration of the EE. They are necessary element in the planning and execution of technological operations.
- Inertial tensor - needed to calculate joint forces and moments [5], [6].

For the current configuration, a class is defined with data for the four joints of MOVER 4:

```
TConfiguration = class(TObject)
  Value,                                  // joint values
  Velocity,                               // joint velocities
  Acceleration  : real [1..4]             // joint accelerations
  InertiaTenzor : real [1..4][1..4]       // mass parameters
  NextConfiguration : ^TConfiguration     // pointer to the next
end;                                      configuration
ConfigurationList: List of TConfiguration;  // dynamic list of
                                            configurations
```

This structure defines a limitation for modeling OKC up to 4 degrees of freedom. It can be overcome by increasing the size of the arrays used or by applying dynamic structures to represent the kinematic chain [7], [8]. Consecutive configurations representing the trajectory of motion in joint coordinates are organized into a dynamic list ***ConfigurationList***. When generating a robot motion control program, the configuration list is created either off-line or during on-line task execution. In both cases, EE should follow strictly defined points in the space in order to realize the purpose of the movement. Therefore, it is necessary to formulate tools for describing trajectories in the three-dimensional space (3D).

C. Trajectory Model in Cartesian Space

Trajectory $\boldsymbol{\Gamma}$ in the workspace $\boldsymbol{D}$, consisting of $\boldsymbol{r}$ base points numbered in sequential order is:

$$\Gamma = \left\{\Gamma_0, \Gamma_1, \Gamma_2, \ldots, \Gamma_r\right\} \subset D \qquad (8)$$

where:

Γ_0 is a starting point;

$\Gamma_1, \Gamma_2, \ldots \Gamma_{r-1}$ are intermediate points;

Γ_r - target endpoint.

Each point $\boldsymbol{\Gamma_i}$ of the trajectory is described by the vector:

$$\Gamma_i = \left\| \begin{array}{c} p_i \\ \theta_i \\ v_i \end{array} \right\| \qquad (9)$$

where:

$p_i = \left\| x_i \quad y_i \quad z_i \right\|^T$ are the Cartesian coordinates;

$\theta_i = \left\| \alpha_i \quad \beta_i \quad \gamma_i \right\|^T$ is the orientation in Euler angles of the EE;

v_i - the velocity in the i-th trajectory point.

The new class that define the trajectory points is descendants of geometric classes of object oriented two-dimensional graphics Pascal_2D [4]. Base point Γ_i of the trajectory is defined as the successor of the KPoint class:

```
TPathPoint =class(KPoint)          // descendant of KPoint
   private                            // 3D-coordinates
      AX,AY,AZ                        // Euler angles
      Velocity : real;                // joint velocity
      NextPoint : ^TPathPoint      //  pointer to the next point
   public                          // methods
      Create;                         // Creates new point
      GET;                            // Get point coordinates
      SET;                            // Set point coordinates
end;
TrajectoryList: List of TPathPoint   // dynamic list of path points
```

The Cartesian coordinates of the point are inherited element of the ancestor class KPoint. The class TPathPoint is complemented by the Euler angles and joint velocities. A pointer to the next element allows the creation of the dynamic path list. GET-methods, according to the requirements of the object-oriented programming realize the class interface, providing data encapsulation. They ensure access to the parameters of the concrete object in the program. SET methods give a possibility to modify the properties of generated point exemplars.

D. Cartesian Trajectory Primitives

The sequence (8) can contain a significant number of points. If each one of them must be explicitly entered, this would create problems for the operator. A possible approach is the choice of several trajectory primitives, each of which is defined by the required minimum number of base points. With appropriate interpolation, intermediate points can be generated. Complex trajectories can be created by combining primitives.

The formulation of the classes is determined by the number of points defining a segment of the trajectory.

1) *Two base points (Γ_S, Γ_E):* They define a straight-line segment in 3D. The new class is the successor to TPathPoint :

```
TPathLine =class(TPathPoint )
   Private
      EndPoint : TPathPoint;          // last point
      NP : integer;                   // number of points
   public
      LinePath(var TrajectoryList);   // creates list of points
   . . . . . .
end;
```

The ***LinePath*** method generates a number of ***NP*** intermediate points that are added to the ***TrajectoryList*** queue. Intermediate points are obtained by linear-spacing interpolation. At start point Γ_S :

$$\Gamma_S = \left\| p_S \quad \theta_S \quad v_S \right\|^T$$

And end point Γ_E

$$\Gamma_E = \left\| p_E \quad \theta_E \quad v_E \right\|^T$$

intermediate coordinates Γ_i , are calculated by:

$$\Gamma_i = \Gamma_S + \left(\Gamma_E - \Gamma_S\right)\frac{i}{r},\ i = 1,2,...,r \tag{10}$$

Where r is the number of intermediate intervals.

2) *Three base points (Γ_S, Γ_M, Γ_E):* They define a circle in 3D. The generated trajectory starts from the first point and ends with the third set point. The class definition is:

```
TPathCircle = class(TPathPoint )
   Private
      MidPoint, EndPoint :TPathPoint;   // middle, last point
      NP : integer;                     // number of points
   public
      CirclePath(var TrajectoryList);   // create list of points
   . . . . . .
end;
```

The task of generating circle points lying in a plane xOy is trivial. If these points are in three-dimensional space, the problem becomes much more complex. In literature, the task of circular interpolation is encountered in CNC programming [9], [10]. One of the possible ways is to use the Newton-Raphson method to solve the nonlinear equations describing the trajectory [11]. Other algorithms are based on the assumption that the rotation axis is known for the three base points using the Rodrigues' rotation formula [12]. In the present work, an algorithm based on the matrix data presentation and using the already developed matrix operations in the CINDY system will be proposed [8]. The basic idea of the algorithm is to find spatial transformations with which the three points are translated in the ***xOy*** plane, and the center of the circle they define coincides with the origin. So intermediate points can be generated, which in the plane is trivial. By applying the inverse transformations the spatial coordinates of the points from the desired trajectory will be obtained.

The algorithm for generating the intermediate points consists of the next steps.

Step 1. Calculation the normal vector $\vec{n}^{\mu}$ to the plane μ, determined by points (p_S,p_M,p_E) -Figure 1:

$$\begin{aligned}
&\mu = \left(p_S, p_M, p_E\right)\\
&\vec{n}^{\mu} = \left(\vec{p}_M - \vec{p}_S\right)\times\left(\vec{p}_M - \vec{p}_E\right) = \vec{a}\times\vec{b} \Rightarrow\\
&\vec{a}\times\vec{b} = \left(\begin{vmatrix} a_Y & a_Z \\ b_Y & b_Z \end{vmatrix}, \begin{vmatrix} a_Z & a_X \\ b_Z & b_X \end{vmatrix}, \begin{vmatrix} a_X & a_Y \\ b_X & b_Y \end{vmatrix}\right) \Rightarrow\\
&\vec{n}_X = \left(n_X^{\mu},\ n_Y^{\mu},\ n_Z^{\mu}\right)
\end{aligned} \tag{11}$$

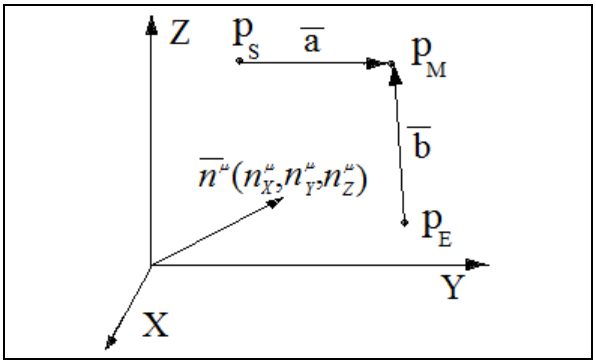

Fig. 1. Normal vector to the plane μ(p_S,p_M,p_E)

Step 2. Two rotations are determined using the normal vector $\vec{n}_\mu$. The first rotation is around axis ***Ox*** at angle ***α*** to parallelism of μ and ***Oy*** (Figure 2). The projection ***d*** of the normal vector $\vec{n}_\mu$ on a plane ***yOz*** is:

$$d^2 = \left(\vec{n}_Y^\mu\right)^2 + \left(\vec{n}_Z^\mu\right)^2 \tag{12}$$

The angle ***α is*** defined by the trigonometric functions:

$$\cos\alpha = \frac{\vec{n}_Z^\mu}{d}; \quad \sin\alpha = \frac{\vec{n}_Y^\mu}{d} \tag{13}$$

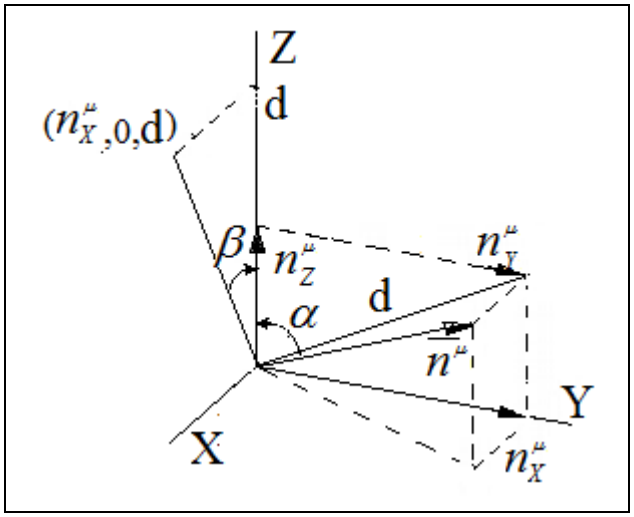

Fig. 2. Rotations necessary to coincide the vector $\vec{n}_\mu$ with the Z axis

The transformation matrix R_X describing the rotation is:

$$R_X = \begin{Vmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{Vmatrix}$$

The substitution of trigonometric functions with their equivalent expressions from (13) gives:

$$R_X = \begin{Vmatrix} 1 & 0 & 0 & 0 \\ 0 & \vec{n}_Z^\mu/d & -\vec{n}_Y^\mu/d & 0 \\ 0 & \vec{n}_Y^\mu/d & \vec{n}_Z^\mu/d & 0 \\ 0 & 0 & 0 & 1 \end{Vmatrix} \tag{14}$$

After this rotation, the normal vector will lie in the ***xOz*** plane and its coordinates are $\left(\vec{n}_X^\mu, 0, d\right)$.

The second rotation is about the axis ***Oy*** at the angle ***β***:

$$\cos\beta = d; \quad \sin\beta = \vec{n}_X^\mu \tag{15}$$

The transformation matrix R_Y is:

$$R_Y = \begin{Vmatrix} \cos\beta & 0 & -\sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{Vmatrix}$$

We again replace trigonometric functions with their equivalent metric expressions from (15):

$$\mathrm{R_Y} = \begin{Vmatrix} d & 0 & -\vec{n}_X^\mu & 0 \\ 0 & 1 & 0 & 0 \\ \vec{n}_X^\mu & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{Vmatrix} \tag{16}$$

With these two transformations, plane ***μ*** becomes parallel with ***xOy***. As shown, the values of the trigonometric functions are not calculated, but the projections of the normal vector are used, which leads to the acceleration of the computational algorithm. The projections $\tilde{p}_S, \tilde{p}_M, \tilde{p}_E$ in complex form of the three base points on ***xOy*** are already known.

Step 3. The center ***C(c_x,c_y,0)*** of the circle passing through them is calculated by the formula [13]:

$$C(c_X, c_Y, 0) = \frac{\begin{vmatrix} \tilde{p}_S \cdot \bar{\tilde{p}}_S & \tilde{p}_S & 1 \\ \tilde{p}_M \cdot \bar{\tilde{p}}_M & \tilde{p}_M & 1 \\ \tilde{p}_E \cdot \bar{\tilde{p}}_E & \tilde{p}_M & 1 \end{vmatrix}}{\begin{vmatrix} \tilde{p}_S & \bar{\tilde{p}}_S & 1 \\ \tilde{p}_M & \bar{\tilde{p}}_M & 1 \\ \tilde{p}_E & \bar{\tilde{p}}_E & 1 \end{vmatrix}} \tag{17}$$

Where $\bar{\tilde{p}}_S, \bar{\tilde{p}}_M, \bar{\tilde{p}}_E$ are the corresponding complex-conjugated forms of the coordinates.

Step 4. With T_{XY} - translation the center is placed in the coordinate origin:

$$T_{XY} = \begin{Vmatrix} 1 & 0 & 0 & -c_X \\ 0 & 1 & 0 & -c_Y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{Vmatrix} \quad (18)$$

Step 5. Thus, after the transformations the coordinates of the point ***p(x,y,z)*** in ***3D*** will be obtained in the ***xOy*** plane and the new coordinates ***p*(x*,y*,0)*** are calculated with:

$$\begin{Vmatrix} x^* \\ y^* \\ 0 \\ 1 \end{Vmatrix} = R_X.R_Y.T_{XY}. \begin{Vmatrix} x \\ y \\ z \\ 1 \end{Vmatrix} \quad (19)$$

Where :

$$R_X.R_Y.T_{XY} = \begin{Vmatrix} d & 0 & -\vec{n}_X^{\mu} & -c_X d \\ -\frac{\vec{n}_X^{\mu}\vec{n}_Y^{\mu}}{d} & \frac{\vec{n}_Z^{\mu}}{d} & -\vec{n}_Y^{\mu} & \frac{c_X\vec{n}_X^{\mu}\vec{n}_Y^{\mu}}{d} - \frac{c_Y\vec{n}_Z^{\mu}}{d} \\ \frac{\vec{n}_X^{\mu}\vec{n}_Z^{\mu}}{d} & \frac{\vec{n}_Y^{\mu}}{d} & \vec{n}_Z^{\mu} & -\frac{c_Y\vec{n}_Y^{\mu}}{d} - \frac{c_X\vec{n}_X^{\mu}\vec{n}_Z^{\mu}}{d} \\ 0 & 0 & 0 & 1 \end{Vmatrix} \quad (20)$$

Step 6. The start point and the end point define the angle of rotation φ around the origin:

$$\cos\varphi = \frac{\vec{p}_S^{\,*}.\vec{p}_E^{\,*}}{\left|\vec{p}_S^{\,*}\right|.\left|\vec{p}_E^{\,*}\right|} = \frac{x_S^*.x_E^* + y_S^*.y_E^*}{\sqrt{\left(x_S^*\right)^2 + \left(y_S^*\right)^2}.\sqrt{\left(x_E^*\right)^2 + \left(y_E^*\right)^2}} \quad (21)$$

In a selected number of ***r*** intermediate points, the sampling angle is $\delta = \varphi / r$, and the transformation matrix is:

$$R_\delta = \begin{Vmatrix} \cos\delta & -\sin\delta & 0 & 0 \\ \sin\delta & \cos\delta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{Vmatrix}$$

Step 7. The coordinates of the intermediate points in the space will be obtained by the inverse transformation:

$$\begin{Vmatrix} x_i \\ y_i \\ z_i \\ 1 \end{Vmatrix} = R_\delta T_{XY}^{-1}.R_Y^{-1}.R_X^{-1}. \begin{Vmatrix} x_i^* \\ y_i^* \\ z_i^* \\ 1 \end{Vmatrix}, \quad \mathrm{i} = 1,2,...,\mathrm{r} \quad (22)$$

Where:

$$T_{XY}^{-1}R_Y^{-1}R_X^{-1} = \begin{Vmatrix} d & -\frac{\vec{n}_X^{\mu}\vec{n}_Y^{\mu}}{d} & \frac{\vec{n}_X^{\mu}\vec{n}_Z^{\mu}}{d} & c_X \\ 0 & \frac{\vec{n}_Z^{\mu}}{d} & \frac{\vec{n}_Y^{\mu}}{d} & c_Y \\ -\vec{n}_X^{\mu} & -\vec{n}_Y^{\mu} & \vec{n}_Z^{\mu} & 0 \\ 0 & 0 & 0 & 1 \end{Vmatrix} \quad (23)$$

Due to the uniform curvature of the circular trajectory, it is appropriate to apply a proportional law to set the velocity and orientation of EE and Eq.10 is applied.

3) *Four and more base points:* At four and more points the interpolation between each base point is a cubic spline:

$$c(t) = a_1.t^3 + a_2.t^2 + a_3.t + a_4$$

The presence of four parameters in the equation requires to have at least four base points [14]. The class definition is:

```
TPathSpline = class(KPoint)
  private
     BasePoints: TrajectoryList;        // list with base points
     NP : integer;                      // number of intermediate
  public                                // points
     SplinePath(var TrajectoryList);
  . . . . . .
end;
```

For ***i***-th interval $t_i t_{i+1}$ (***i***=1,...,r-1) the piece-wise cubic polynomial spline is:

$$c_i(t) = \frac{S_i\left(t_{i+1} - t\right)^3 + S_{i+1}\left(t - t_i\right)^3}{6h_{i+1}} + \left[\frac{t_i}{h_{i+1}} - \frac{S_i h_{i+1}}{6}\right].\left(t_{i+1} - t\right) + \left[\frac{t_{i+1}}{h_{i+1}} - \frac{S_{i+1} h_{i+1}}{6}\right].\left(t - t_i\right) \quad (24)$$

Where: $c_i = [p_i, \theta_i, v_i]$;

$h_{i+1} = t_{i+1} - t_i$;

$S_i = f\left((t_i), c'(t_i), c''(t_i)\right)$ - solution of the linear system that satisfies the continuity requirements for spline function.

The formula (24) is applied to each of the coordinate (9).

Spline interpolation is very well suited to describe robot trajectories because it provides position, velocity and acceleration as smooth functions of time. Spline interpolation is unsuitable when base points are located approximately in a straight line because oscillations are generated.

E. Generation of XML file

The syntax requirements that the XML-file must meet are described in the manual [1]. The motion trajectory created by the language is in a ***TrajectoryList*** dynamic list.

The header of the XML-file is in Figure 3.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- values in mm and degree -->
<Program>
      <Header ProgramName="program name"
       Kinematic="CPRFour" LastChangeDate="date2"
       SetUpDate="date1" Author="author names" />
</Program>
```

Fig. 3. The header of the XML-file

Another important element in the syntax of XML-language is the command for straight-line motion from the current position to the (***x,y,z***) coordinate with velocity ***v*** (Fig.4).

```
<Linear Nr="integer" x="float" y="float" z="float"
vel="integer" acc="integer" smooth="bool" Descr=""/>
```

Fig. 4. Syntax of the linear motion command

The algorithm for generating the XML trajectory consists of sequential scan the structure ***TrajectoryList*** and creating a string according to the XML-syntax. The pseudo-Pascal representation of the algorithm is shown on Fig.5.

```
procedure PathInXML(const TrajectoryList );
  var PathItem : TPathPoint ; // current path point
      SX,SY,SZ,SV : string;     // string presentations
      Command : string;         // generated command line
      I : integer;              //   counter of the points
  begin
      PathItem:=TrajectoryList ;
        repeat
            Inc(I);
            with PathItem do
            begin
                  SX:=floattostr(Get(X));
                  SY:=floattostr(Get(Y));
                  SZ:=floattostr(Get(Z));
                  SV:=floattostr(Get(Velocity ));
            end;
            Command:=concat( '<LinearNr=',inttostr(I),
             'x=',SX, 'y=',SY, 'z=',SZ,'vel=',SV,
            smooth="true" Descr=""/> )
            Writeln(''Splinepath.xml', Command );
            PathItem :=PathItem.^NextPoint ;
         until (PathItem = nil);
  end; // PathInXML
```

Fig. 5. Algorithm for generation the XML file

III. Results and Discussion

CINDY system is used to create the robot path and to check its validity. A trajectory of five base points is given (Table I). Spline interpolation has been applied to generate the EE trajectory. Between each two base points, 100 intermediate ones are generated. Due to a large number of generated commands, only a part of the program in XML format is shown in Fig. 6.

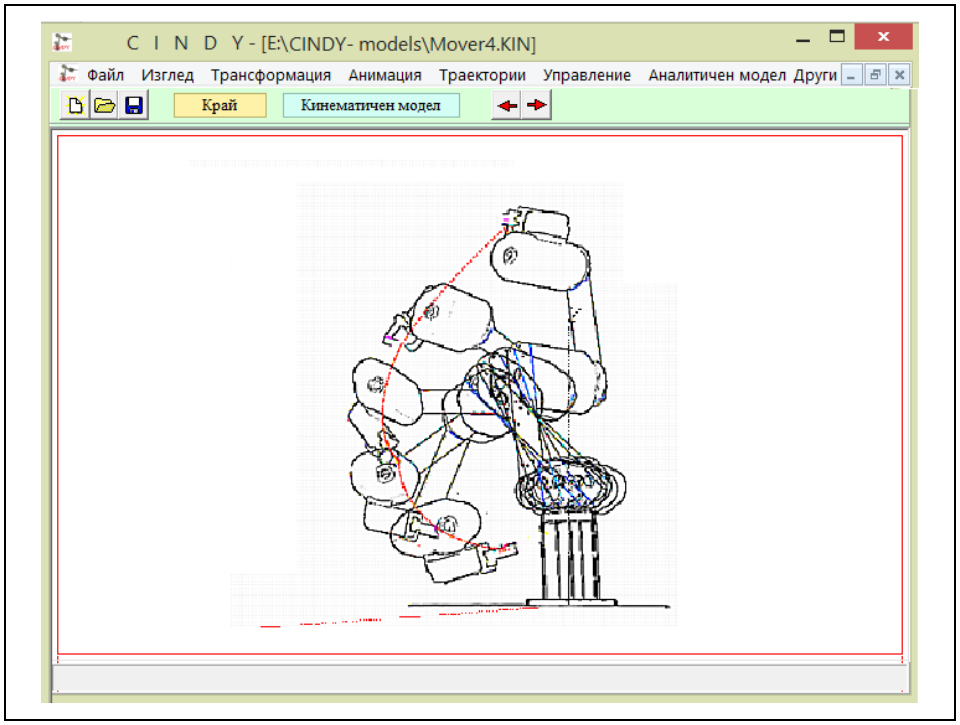

Fig.7 Some configurations of MOVER 4 during the spline path motion

TABLE I.
BASE POINTS FOR SPLINE GENERATION

t	X	Y	Z	Velocity [mm/s]
0	-43.8	152.7	57.5	0
10	238.4	255.8	308.8	45
20	260.0	-72.4	583.3	70
30	155.2	-276.6	45.4	40
40	-76.0	-326.1	195.3	0

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- values in sm and degree -->
<Program>
    <Header ProgramName="Spline path"
  Kinematic="CPRFour" LastChangeDate="2017-07-21"
  SetUpDate="2017-03-13" Author="Kaloyan Yankov"/>
<Linear Nr="1" x="-43.8" y="152.7" z="57.5" a="" b=""
       c="" vel="0" acc="0" smooth="true" Descr=""/>
<Linear Nr="2" x="-40.35" y="155.83" z="59.25"
       vel="0.47" acc="0.118" smooth="true" Descr=""/>
<Linear Nr="3" x="-36.89" y="158.94" z="60.59"
       vel="0.93" acc="0.235" smooth="true" Descr=""/>
<Linear Nr="4" x="-333.44" y="162.01" z="62.74"
       vel="1.39" acc="0.353" smooth="true" Descr=""/>
<Linear Nr="5" x="-29.99" y="165.04" z="64.49"
       vel="1.86" acc="0.470" smooth="true" Descr=""/>
<Linear Nr="6" x="-26.55" y="168.05" z="66.25"
       vel="2.33" acc="0.588" smooth="true" Descr=""/>
. . . . . .
<Linear Nr="399" x="-73.39" y="-325.86" z="191.37"
      vel="0.391" acc="-0.099" smooth="true" Descr=""/>
<Linear Nr="400" x="-76.00" y="-326.10" z="195.30"
       vel="0" acc="0" smooth="true" Descr=""/>
</Program>
```

Fig.6. Robot program in XML format

Fig. 7 shows the simulation of the movement of MOVER 4 with program CINDY, tracking part of the trajectory. CINDY allows visualizing the graphs of change of the kinematic parameters of MS both in Cartesian coordinates and in the space of the generalized coordinates.

On Fig. 8 are the graphs of the change of the coordinate of the end-effector respectively in X, Y and Z axis.

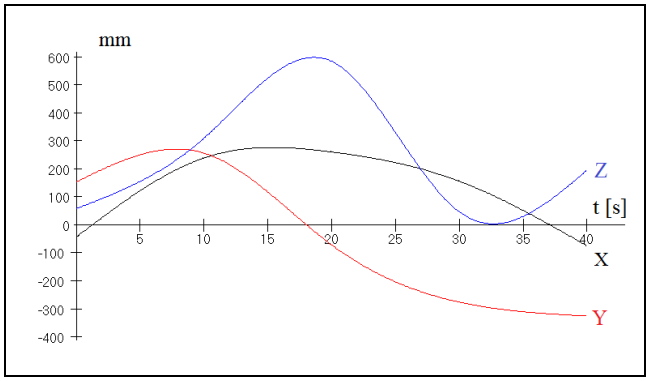

Fig. 8. Cartesian coordinates of the robot path

Fig. 9 represents the variation of the velocity of EE decomposed on the coordinate axes.

The changes of the four generalized coordinates are in Fig. 10, and in Fig. 11 are their velocities.

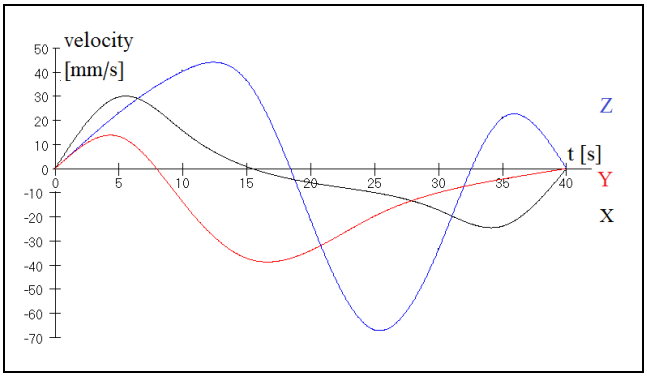

Fig. 9. Cartesian velocities of the end-effector

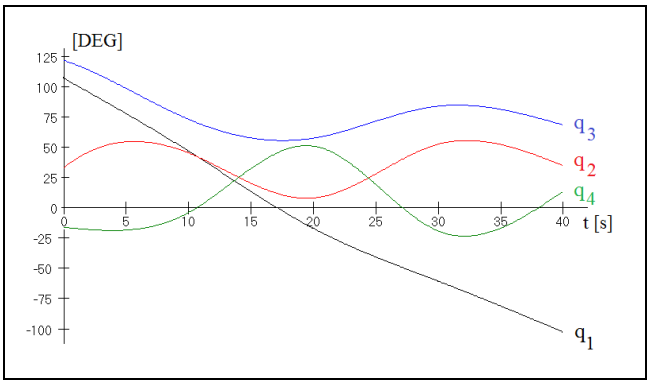

Fig. 10. Joint coordinates

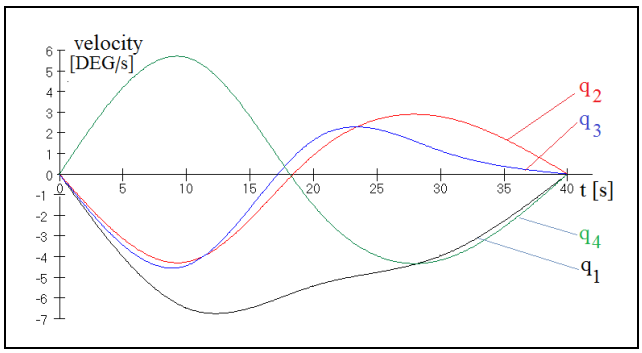

Fig. 11. Joint velocities

IV. CONCLUSIONS

In this paper a simple language for describing a robot technology paths is presented. Two kinds of trajectory model are formulated. The first is an object-oriented model in the joint space. The model includes values, velocities and accelerations of joint coordinates and inertia tensor of the configuration. The inertia tensor is necessary for calculating joint forces and moments and thus to model the dynamics of the robot.

The second trajectory model is in three-dimensional space. A point from the trajectory is defined by the Cartesian coordinates of the end-effector, its orientation

with the Euler angles and its speed. The means for trajectory description are three spatial primitives: segments, circle arcs, and cubic splines. Each primitive has a method of generating the intermediate points. Complex paths can be created with an arbitrary combination of primitives. For creating, tuning and analyzing the programmed trajectories, the CINDY simulation system is used.

The language will be used to explore the kinematics and dynamics of a MOVER 4 robot following different paths.

ACKNOWLEDGMENT

This study was supported by Grant 3ФТТ/30.04.2015 "Identification and Simulation of Second-order Dynamic Models" from the Faculty of Technics and Technologies, Trakia University - Yambol, Bulgaria.

REFERENCES

[1] *User Guide Mover4*, Version (SW V902-08-008, HWE 2MV23 HWM V05 DOC V14), Commonplace Robotics GmbH, Germany, 2015.

[2] I. Pembeci and G. Hager. (2001) A Comparative Review of Robot Programming Languages. [Online]. Available: http://www.cs.jhu.edu/CIRL/publications/pdf/pembeci-review.pdf

[3] F. Wahl and M. Thomas, "Robot Programming - From Simple Moves to Complex Robot Tasks," in *Proc. of First Int. Colloquium "Collaborative Research Center 562–Robotic Systems for Modelling and Assembly*", 2002, Braunschweig, Germany, pp.249-259.

[4] K. Yankov, "Object Oriented Two-Dimensional Graphics in a Borland Delphi Environment," in *Proc. SAER'99*, 1999, pp. 169-173.

[5] K. Yankov, "Inertial Parameters of the Robot Mover 4," in *Proc. InfoTech-2016*,sept.2016, pp.316-325.

[6] A. Dharmawan, A. Ashari and A. E. Putra, "Mathematical Modelling of Translation and Rotation Movement in Quad Tiltrotor," *International Journal on Advanced Science, Engineering and Information Technology*, vol.7(3), pp.1104-1113, 2017.

[7] K. Yankov, "Kinematic simulation in robotics," Ph.D.-thesis, Technical University, Sofia, Bulgaria, Jan. 1992.

[8] K. Yankov, "Computer Simulation of Industrial Robots," in *Proc. ACMBUL'92 "Computer Applications"*, okt. 1992, paper 33.

[9] A. Lubbe, "Mathematical Basis For Three Dimensional Circular Interpolation on CNC Machines," *The South African Journal of Industrial Engineering*, vol.8 (2), pp 47-59, 1997.

[10] H. Liang, "Minimum Error Tool Path Generation Method and an Interpolator Design Technique for Ultra-precision Multi-axis CNC Mashining," Ph.D.-thesis. Concordia University, Monreal, Quebec, Canada, July 1999.

[11] L. Maisonobe. (2007) Finding the circle that best fits a set of points. [Online]. Available: http://www.spaceroots.org/documents/circle/circle-fitting.pdf

[12] D. Rogers and J. A. Adams, *Mathematical Elements for Computer Graphics*, 2nd ed., McGraw-Hill Publishing Company, 1990.

[13] A. Nicolae, "Determinant Identities and the Geometry of Lines and Circles," *The Journal of "Ovidius"*, University of Constanta, Versita, vol. 22 (2), pp. 37-49, 2014.

[14] I. Faux and M. Pratt, *Computational Geometry for Design and Manufacture*, 2nd ed, G.M.Bell,Ed., New York, USA: Ellis Horwood Ltd, 1981.