The Robot Toolbox for Matlab

G. PAJĄK and I. PAJĄK
University of Zielona Góra
Institute of Computer Science and Production Management
Licealna 9, 65-417 Zielona Góra, POLAND
e-mail: g.pajak@iizp.uz.zgora.pl; i.pajak@iizp.uz.zgora.pl

In the paper a Matlab toolbox for modeling and simulation manipulators described by Denavit-Hartenberg parameters is presented. This package can be used to realistic visualization of robot motion necessary in research, didactic process or during design of production cell. Available functions allow to show realistic model of mechanism easily based on DH parameters only. The toolbox also provides a set of tools for creating any polyhedrons and functions to manipulate them in 3D space. Using those objects it is possible to create own appearance of the manipulator and its workspace. Functions implementing classical methods of trajectory planning in configuration space allow to calculate manipulator trajectory which can be shown as an animation. The main functions and computer examples illustrating the features of this toolbox have been presented.

Key words: robot toolbox, manipulator modeling, simulation.

1. Introduction

The visualization of manipulators and tasks realized by them plays an important role in many applications. The validation of research results is not always possible by performing real experiment using real robot in its environment. Even if a real manipulator is available simulation tools allow to verify and compare obtained solution using many different mechanisms in any designed workspace. Moreover, while teaching the basis of robotics work with real robots it is not always possible, so simulation tools are needed. In virtual environment student can learn issues related to description of position and orientation in 3D space, convention used to describe the manipulator kinematics and, finally, build any mechanism. Additionally, computer modeling is useful during programming industrial robot tasks. The simulation tools allow to design and verify a production cell in virtual environment without occupation of mechanical robots and other equipments. In the applications presented above performing a simulation using simplified models may by insufficient, often it is important that manipulator and objects in its environment look like a real workspace.

There are a lot of tools which can be used in the applications presented above. The robot manufactures provide their own software allowing to simulate a robotic process in 3D space (Fanuc (2011), ABB (2011)). The main drawback of such tools is their narrow specialization, they support own robots only. This feature limits usage of this software in research and educational applications. There are also more versatile solutions which allow to simulate a broader class of mechanism. López-Nicolás et al. (2009) presented applications package for creating models of robots and their workspace. The robots created in this way may be programmed and shown in a graphical simulator. Cakir and Butun (2007) created an educational tool for interactive simulation 6-DOF industrial robotic arms. The program is written for manipulators with revolute joints only and that it is its main drawback. Gourdeau (1997) and Bruyninckx (2001) developed independently object oriented programming toolboxes in C++ for synthesis and simulation of robotic manipulator models. Using those tools it is possible to build any robotic system but knowledge of C++ programming is

necessary. Knowing LISP programming language is needed in order to use window-based robot simulation tool developed by Bingul et al. (2002).

The tools presented above are standalone applications or libraries for certain programming languages. However, in robotics, both research and education are often carried out using complete environments for high-level programming and it is convenient that the simulation tool is integrated with such an environment. The one of the most popular applications is MathWorks product Matlab (MathWorks (2010)) but, unfortunately, it does not have specialized toolbox for robotics. There are some independent libraries covering that gap, such as Robotic Toolbox for Matlab (Corke (1996)) and SpaceLib (Legnani (2006)). Both libraries provide many useful tools necessary for robotic modeling and simulations, but they show poor graphics capabilities and do not allow to create the realistic models of mechanisms. Kucuk and Bingul (2010) presented ROBOLAB toolbox which allows users to compute forward and inverse kinematics, calculate manipulator trajectories using classical methods and view robot animations. The main disadvantage of that tool is that it allows to make calculations for 16 predefined 6-DOF serial manipulators only, and to create their own appearance is not possible.

In this paper the Matlab toolbox for modeling and simulation manipulators described by Denavit-Hartenberg parameters is presented. The main advantage of that toolbox in comparison to toolboxes mentioned above is possibility to make simulation of any robot manipulator. Furthermore, a user can easily show manipulators using default 3D appearance presenting action of mechanism in realistic manner or can define his/her own model of manipulator corresponding to the real construction. The toolbox also provides a set of basic blocks, tools for creating any polyhedrons and functions for manipulating position and orientation of those objects in 3D space. Using previously created elements it is possible to define a manipulator appearance and its workspace. Additionally, the manipulators motions can be calculated using one of the classical methods of trajectory planning in the configuration space.

2. Description of the manipulator and its workspace

2.1. Position and orientation in 3D space

In the work, we assumed each solid is described in its own local coordinate system. Hence, the position and orientation of that coordinate system sets the location of the solid precisely. To determine position of the local coordinate system in any reference system 3×1 translation vector describing position of the origin of the local system is required. To describe orientation solid coordinate system in the reference system an 3×3 orientation matrix may be used. In compact notation translation vector and orientation matrix may be combined together in 4×4 homogeneous transformation matrix (Craig (1993)) as follows:

$$ {}^{R}_{S}\mathbf{T} = \begin{bmatrix} {}^{R}_{S}\mathbf{R} & {}^{R}\mathbf{P}_{ORG} \\ \mathbf{0}_{1\times3} & 1 \end{bmatrix}, \tag{2.1} $$

where: ${}^{R}_{S}\mathbf{T}$ - 4×4 homogeneous transformation matrix describing position and orientation of solid local system $\{S\}$ in the reference system $\{R\}$; ${}^{R}_{S}\mathbf{R}$ - 3×3 rotation matrix describing orientation of solid local system $\{S\}$ in the reference system $\{R\}$; ${}^{R}\mathbf{P}_{SORG}$ - 3×1 translation

vector describing position of origin of solid local system $\{S\}$ in the reference system $\{R\}$; $\mathbf{0}_{1\times3}$ - 1×3 zeros vector.

Dependencies given by (2.1) are a convenient and efficient way to represent a sequence of transformations and they are basic form of the description of position and orientation in the presented toolbox.

2.2. Kinematics of the manipulator

In robotics applications a Denavit-Hartenberg convention is commonly used to describe the manipulator kinematics. To determine DH parameters assigning a coordinate systems to each joint of manipulator is needed. There are two different methodologies defining the way of coordinate system attachment. In the first one, described by Paul (1981) and Fu, et al. (1987), origin of coordinate system $\{i\}$ is located on the axis of joint $i+1$. In the second one, described by Craig (1993), origin of coordinate system $\{i\}$ is located on the axis of joint i. In presented toolbox the second convention named "modified Denavit-Hartenberg" is used to describe manipulator kinematics. In that case z-axis of the coordinate system $\{i\}$ is placed along the axis of joint *i*, the x-axis is parallel to the normal to axes of joint *i* and *i*+1, y-axis completes the right-handed coordinate system. The DH parameters in this convention are defined as follows:

link length (a_i) – distance between Z_i and Z_{i+1} along X_i,

link twist (α_i) - angle between Z_i and Z_{i+1} around X_i,

joint offset (d_i) - distance between X_{i-1} and X_i along Z_i,

joint angle (θ_i) - angle between X_{i-1} and X_i around Z_i.

In the fig. 2.1 the method of coordinate system attachment and DH parameters is presented.

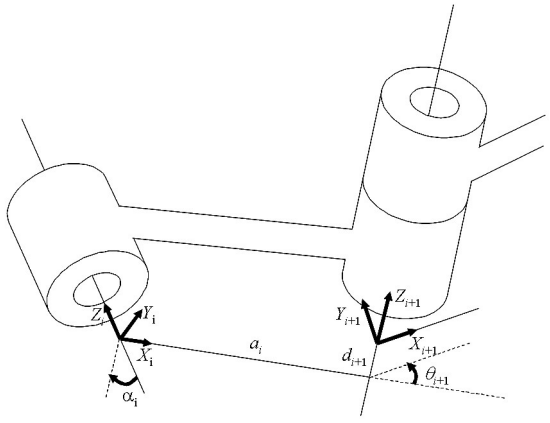

Fig. 2.1. Kinematic parameters determined according to the modified DH convention

3. Overview of basic functions

3.1. Position and orientation

The method of description of position and orientation used in presented toolbox has been described in section 2.1. The homogeneous transformation matrix (2.1) may be defined using the function

```
T = rp2t(R, P)
```

where: `T` - 4×4 homogeneous transformation matrix ; `R` – rotation matrix; `P` – translation vector.

To specify orientation, three functions defining rotation matrices around x-axis, y-axis and z-axis have been implemented:

```
R = rotx(angle)
R = roty(angle)
R = rotz(angle)
```

where: `R` – rotation matrix; `angle` – angle of rotation around respective axis.

The composition of several transformations may be accomplished by multiplication of transformations or rotations matrices. To determine the inverse transformation (2.1) a suitable function, which preserves a structure of homogeneous transformation, has been implemented:

```
T = tinv(T)
```

where: `T` - 4×4 homogeneous transformation matrix.

3.2. Coordinate systems objects

The toolbox provides a set of functions for creating and working with coordinate systems. Those functions allow to understand the description of the position and orientation in 3D space by homogenous transformation (2.1). A coordinate system object may be created by calling following function:

```
cs = csys(T)
```

where: `cs` - structure representing a coordinate system; `T` - 4x4 transformation matrix determining position and orientation.

Changing position and/or orientation of the coordinate system is possible by using function:

```
cs = csys_move(cs, T, kind)
```

where: `cs` - structure representing a coordinate system; `T` - 4x4 transformation matrix determining changes of position and orientation; `kind` – an optional parameter specifying the method of calculating new position and orientation, if `kind` equals 0 then `T` is new position and orientation (default value), if `kind` is equal to 1 then `T` describes changes of position and orientation according to global coordinate system, if `kind` equals 2 then `T` describes changes of position and orientation according to current coordinate system.

Each coordinate system defined by `csys` can by plotted in a Matlab figure window and then it may be removed using functions:

```
cs = csys_plot(cs)
cs = csys_delete(cs)
```

where: `cs` - structure representing a coordinate system.

Additionally, a coordinate system object has a set of properties, which define its appearance (color, length of axis, description etc.). Those properties can be determined while creating coordinate system or by using appropriate function. The description of all properties is included in the documentation of the toolbox.

3.3. Blocks objects

The blocks are basic objects in presented toolbox, they are used to define the appearance of manipulators and may be used to create a robot workspace. Each block is approximated by polyhedron, the primary function allowing to define any object of this type is:

```
b = block(p, f)
```

where: `b` - structure representing a block, `p` - a column matrix containing the x, y, z coordinates for each vertex of the block; `f` - a cell array containing connection matrices specifying which vertices in the `p` are connected.

Using of the above function requires determination of each point and each edge of the block, so it is laborious. Therefore, additional set of functions defining elementary blocks has been implemented: `cone`, `cube`, `cuboid`, `cylinder`, `polyline`, `prism`, `pyramid`, `sphere`, `tetrahedron`. Additionally, the complex blocks can be created by merging a previously defined blocks using the function `block_merge`. Detailed description of those function is included in the documentation.

On the block objects similar operations available to coordinate systems objects can be performed. These are the implemented functions: `block_move`, `block_plot`, `block_delete`, which enable changing position and orientation, plotting and removing the block from a figure window respectively. Those functions can be used in the same way as their equivalents defined for coordinate systems. Additionally, local coordinate system attached to a block can be plotted using function:

```
b = block_plot_csys(b)
```

where: `b` - structure representing a block.

Similarly to coordinate systems, block objects have a set of properties defining their appearance. All properties are described in toolbox documentation in a detailed manner.

3.4. Manipulators objects

A set of functions defining and operating on manipulator objects are of crucial importance for presented toolbox. The basic function which allows to define manipulator is:

```
r = robot(dh, joints, 'PropName', PropValue, ...)
```

where: `r` - structure representing a manipulator; `dh` - $4 \times n$ matrix containing DH parameters set according to convention presented in section 2.2, n - number of joints; `joints` - n-element character array containing joints' types, a single joint can be defined as rotate (`'r'`) or prismatic (`'p'` or `'d'` depending on the type of the link – see section 4.2); `PropName`, `PropValue` – optional parameters, properties specifying the appearance of the manipulator.

For objects of that type optional properties set during their defining are particularly important. They determine the way to plot the manipulator object in the Matlab figure window, the essentials of them are listed below:

`RobotBase`	transformation matrix determining position and orientation of robot base,
`RobotGrasper`	transformation matrix determining position and orientation of the grasper coordinate system,
`RobotRange`	matrix containing ranges of motion in individual joints,
`RobotModel`	method of plotting the manipulator in default 3D view, there are four allowed values: `'config'`, `'configex'`, `'range'`, `'rangeex'` (they will be explained in section 4).

After creating, the manipulator object is in the configuration resulting from its DH parameters. That configuration can be changed using function:

```
r = robot_config(r, conf)
```

where: `r` - structure representing a manipulator; `conf` – n-element vector containing new configuration of the robot.

The same function may be used to read the current configuration if it is be called without `conf` parameter. In that case an output value will be a vector containing a current configuration.

Each manipulator object defined by `robot` function can be plotted in Matlab figure window using function:

```
r = robot_plot(r, view)
```

where: `r` - structure representing a manipulator; `view` – method of plotting the manipulator object, if `view` equals `'line'` each nonzero length and offset of the manipulator is plotted as a line, if `view` is equal to `'mesh'` the manipulator is plotted as wireframe 3D object, if `view` equals `'surf'` the manipulator is plotted as a set of patches.

While creating the manipulator object gets its default 3D appearance dependent on DH parameters, types of joints and values of properties. In that case each nonzero length and offset is represented by a single cylinder. There is also possibility to define own appearance of the manipulator. For this purpose, a shape of each joint using the block objects should be determined. Definition of the joint appearance can be done by function:

```
r = robot_arm(r, nr, block, ...)
```

where: `r` - structure representing a manipulator; `nr` – number of joint; `block, . . .` – set of blocks defining the arm appearance, position and orientation of each block are specified in joint local coordinate system.

Regardless of how the manipulator is plotted, the local coordinate systems of individual manipulator joints may be shown using function `robot_plot_csys`. The manipulator and its coordinate systems can be removed form Matlab figure window by function `robot_delete`.

The presented toolbox also provides the functions simulating manipulator motions. The first function creates a special panel in Matlab figure window which allows to change a configuration of the manipulator using set of controls (each control corresponds to single configuration variable). To display this panel a following function is used:

```
robot_panel(r)
```

where: `r` - structure representing a manipulator plotted in figure window.

The second function enables to show manipulator realizing the pre-planned trajectory:

```
r = robot_motion(r, q, delay)
```

where: `r` - structure representing a manipulator object; `q` - matrix containing pre-planned trajectory; `delay` - pause between animation steps.

3.5. Trajectory planning

Three classical methods of trajectory planning in the configuration space have been implemented. The first one allows to generate trajectory described by cubic polynomials:

```
[Q,V,A,T,c] = trj_poly3(T, Q0, Qt, V0, Vt)
```

where: `T` – vector containing a set of time instants for which trajectory will be generated; `Q0`, `Qt` - initial and final configuration of the manipulator; `V0`, `Vt` – optional initial and final velocity; `Q` - trajectory for each time instant and each joint; `V`, `A` - the instantaneous velocities and accelerations; `c` – matrix containing polynomials' coefficients describing calculated trajectory.

If it is important to determine an acceleration at the beginning and at the end of the motion 5-th order polynomials can be used:

```
[Q,V,A,T,c] = trj_poly5(T, Q0, Qt, A0, At, V0, Vt)
```

where: `A0`, `At` – initial and final acceleration; other parameters have similar meaning as `trj_poly3` parameters.

The third trajectory planning method implemented in the toolbox gives a trajectory described by linear segment with parabolic blends:

```
[Q,V,A,T,c] = trj_line(T, Q0, Qt, A0, At, V0, Vt)
```

where all parameters have similar meaning as those presented above.

4. Examples

In that section definition of two chosen manipulators and simulation of certain manipulator task will be presented. The properties determining the manipulator appearance and definition of the

shape specified by user will be discussed. Finally, usage of implemented functions to solve the trajectory planning task and simulation of resulting movements of manipulator will be shown.

4.1. Planar 3R manipulator

A planar manipulator of three-revolute kinematic pairs whose DH parameters are given in the table below will be defined.

Table 1. DH parameters of 3R manipulator

i	α_{i-1}	a_{i-1}	d_i	θ_i
1	0	0	0	θ_1
2	0	0.4	0	θ_2
3	0	0.36	0	θ_3

Additionally, the manipulator is equipped with a grasper 0.15 units length. Grasper segment is shifted along z-axis of the coordinate system connected to the last joint by -0.025 units.

Using the function `robot` such a manipulator can be defined in the following way:

```
r = robot([dh(0,0,0,-pi/4);dh(0,0.4,0,pi/2);...
           dh(0,0.36,0,-pi/4)], ['r', 'r', 'r'],...
          'RobotBase', rp2t(rotx(-pi/2),[0;0;0]),...
          'RobotGrasper', [0.15; 0; -0.025],...
          'RobotGrasperSize', 0.05,...
          'RobotLineWidth', 2);
```

First parameter in the above function call is an array containing DH parameters (joints angles are set respectively to $-\pi/4$, $\pi/2$, $-\pi/4$). The second parameter specifies type of manipulator joints, in that case all the joints are revolute. The following parameters are properties determining additional features of manipulator. `RobotBase` transforms the base of the manipulator in such a way as to ensure a natural position of the robot in Matlab figure window. `RobotGrasper` defines the length and location of the manipulator grasper according to above specification, a non-zero value of `RobotGrasperSize` causes drawing a default grasper located in the origin of the local coordinate system. `RobotLineWidth` determines the line width that the manipulator will be plotted in `'line'` view. The manipulator defined above, plotted using function `robot_plot`, has been shown in fig. 4.1.

The same manipulator can be plotted in default 3D view, but due to too large default values of parameters specifying sizes of the blocks defining manipulator appearance the change of their values is required. It can be done by redefinition of the manipulator object or by modification of the existing object using function `robot_set`:

```
r = robot_set(r, 'RobotCylinderRadius', 0.04);
```

It can be necessary to show local coordinate systems of individual manipulator joints. Default value of the parameter specifying the axis length is also too large for the manipulator defined in that example, so it has to be changed:

```
r = robot_set(r, 'CSysAxisLength', [0 0.2 0 0.2 0 0.2]);
```

The manipulator with its local coordinates systems plotted using functions:

```
r = robot_plot(r, 'surf');
r = robot_plot_csys(r, 1);
```

is shown in fig. 4.2.

This manipulator has been plotted using default value of the property `RobotModel` (i.e. `'config'`), to improve the readability of the drawing another model can be used:

```
r = robot_set(r, 'RobotModel', 'cofigex');
```

In that model each joint is marked with additional cylinder placed along axis of the joint, it can be seen in fig. 4.3.

Default 3D views may differ from the required appearance of the manipulator, so it is possible to create own 3D model of mechanism. For this purpose shape of each joint has to be determined by blocks objects (section 3.3). The Matlab code defining the own appearance of the manipulator considered in that section has been presented below. The manipulator plotted in `'surf'` view is shown in fig. 4.4.

```
 1 par = {'BlockColor',[0.75,0.75,0.75],'BlockEdgeColor','none'};
 2 r1  = 0.048;  r2 = 0.036;  r3 = 0.012;
 3 l1  = 0.4;    l2 = 0.36;   lt = 0.15;
 4 w   = 0.05;   nr = 25;

 5 b1 = block_move(cuboid(0.2,0.2,0.025,par{:}),rp2t(rotx(pi/2),[0;0.095;0]));
 6 p = [[r1;0.07] arc([r1;0],[-r1,0],r1,nr) [-r1;0.07]];
 7 b2 = block_move(prism(p,w,par{:}), [0;0;-w]);
 8 r = robot_arm(r, 0, b1, b2);

 9 p = [arc([0;-r1],[0;r1],r1,nr) arc([l1;r2],[l1;-r2],r2,nr)];
10 r = robot_arm(r, 1, prism(p,w,par{:}));

11 p = [arc([0;-r2],[0;r2],r2,nr) arc([l2;r3],[l2;-r3],r3,nr)];
12 r = robot_arm(r, 2, block_move(prism(p,w,par{:}), [0;0;-w]));

13 b = block_move(cylinder(r3,lt,nr,par{:}), rp2t(roty(pi/2),[-lt;0;0]));
14 r = robot_arm(r, 4, b);
```

The definition of parameters determining features of the blocks and manipulator dimensions are given in lines 1-4. Base of the manipulator (lines 5-8) is composed of two blocks: flat cuboid and rounded prism. The first (lines 9-10) and second (lines 11-12) joint is a prism rounded on both sides. The manipulator operates with a cylindrical tool, which is placed in the grasper coordinate system (lines 13-14). All rounded elements have been defined using toolbox function `arc` which creates an arc of a given radius that passes through two points.

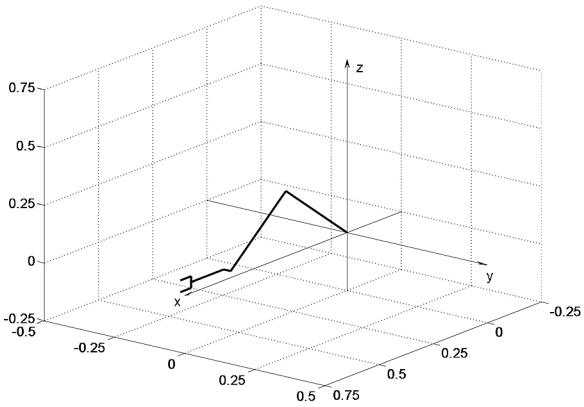

Fig. 4.1. Manipulator in `line` view

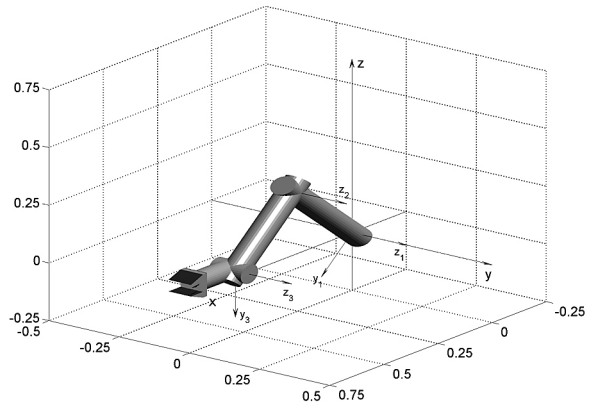

Fig. 4.2. Default 3D view, model `config`

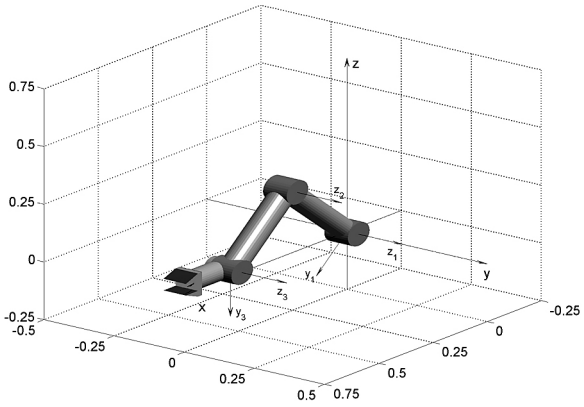

Fig. 4.3. Default 3D view, model `configex`

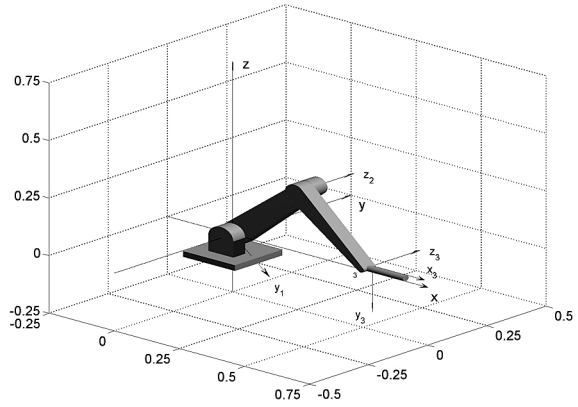

Fig. 4.4. User defined 3D view

4.2. Cartesian manipulator (3P)

A manipulator of three mutually perpendicular prismatic joints equipped with a grasper 3 units length will be discussed. Its DH parameters are given in the table below.

Table 2. DH parameters of 3P manipulator

i	α_{i-1}	a_{i-1}	d_i	θ_i
1	0	0	d_1	0
2	$\pi/2$	0	d_2	$-\pi/2$
3	$-\pi/2$	2	d_3	0

Using the function `robot` such a manipulator can be defined in the following way:

```
r = robot([0 0 5.0 0; pi/2 0 7.0 -pi/2; -pi/2 2 3.0 0],...
          ['d'; 'd'; 'p'],...
          'RobotBase', rp2t(rotx(-pi/2),[0;0;0]),...
          'RobotGrasper', rp2t(roty(-pi/2),[0; 0; 3]),...
          'RobotGrasperSize', [0.75,1],...
```

```
'RobotRange', [-3.5, 8.5; 1, 11; 0, 6],...
'RobotCylinderRadius', 0.5,...
'RobotModel', 'rangeex');
```

In the first argument in the above function call DH parameters have been set with the initial configuration: 5.0, 7.0, 3.0. The second parameter specifies type of manipulator joints. In that case all joints are prismatic, but two different types of the link have been used – it is related to the method of plotting the manipulator in default 3D view (property `RobotModel`). If prismatic joint is marked as `'p'` the block connected to this joint is a movable element. If prismatic joint is marked as `'d'` the block connected to this joint is a fixed element and the next segment of kinematic chain moves along it. The property `RobotRange` sets the ranges of motions for each joint, and in that case it also specifies the dimensions of the manipulator blocks in default 3D view.

The model `range/rangeex` used above is more natural for manipulator with prismatic joints. In default model `config` manipulator arms change their dimensions during the reconfiguration. 3P manipulator plotted using function `robot_plot`, in default 3D view discussed here has been shown in fig. 4.5. and 4.6 in models `rangeex` and `configex` respectively.

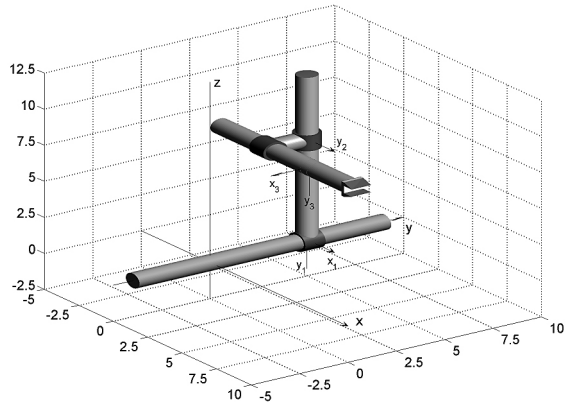

Fig. 4.5. 3P manipulator, model `rangeex`

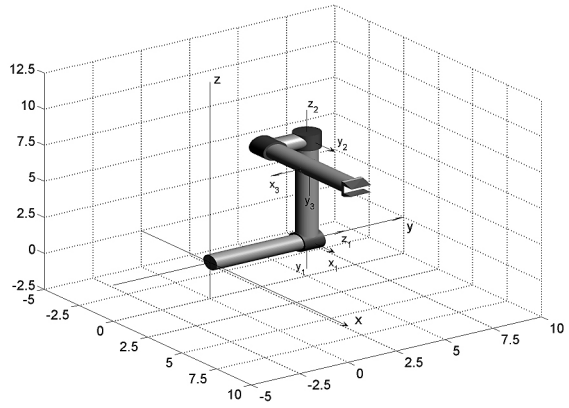

Fig. 4.6. 3P manipulator, model `configex`

4.3. Trajectory planning

The manipulators defined above can be used to simulation of their motions. In this example 3R planar manipulator defined in section 4.1. will perform the task of movement from initial $[-2.04,\ 2.51,\ 1.10\ ]^T$ to final $[-1.17,\ 1.7,\ 1.04\ ]^T$ through its intermediate configuration $[-1.79,\ 1.63,\ 1.54\ ]^T$ (manipulator avoids the obstacle). The Matlab code realizing this simulation has been presented below.

```
1 r = robot([dh(0,0,0,-2.04); dh(0,0.4,0,2.51); dh(0,0.36,0,1.1)],...
2           ['r', 'r', 'r'], 'RobotBase', rp2t(rotx(-pi/2),[0;0;0]),...
3            'RobotGrasper', [0.15; 0; -0.025], 'RobotGrasperSize', 0.05,...
4            'RobotCylinderRadius', 0.025, 'RobotModel', 'configex');
5 axis([-0.25 0.75 -0.5 0.5 -0.25 0.75])
6 csys_plot(csys(eye(4), 'CSysAxisLength', axis));
7 r = robot_plot(r, 'surf');
```

```
 8 b = block_plot(block_move(cuboid(0.05,0.2,0.1),[0.3;0;0]), 'surf');

 9 t1 = 0:0.1:2;  t2 = 2:0.1:4;
10 q1 = trj_line(t1, [-2.04;2.51;1.10], [-1.79;1.63;1.54], 1, 1);
11 q2 = trj_line(t2, [-1.79;1.63;1.54], [-1.17; 1.7; 1.04], 1, 1);

12 r = robot_motion(r, [q1 q2]);
```

In lines 1-4 there is the definition of manipulator. The workspace including the global coordinate system, the manipulator in the initial configuration and the obstacle (cuboid) are defined in lines 5-8. The linear trajectory with parabolic blends has been calculated in lines 10-11 – the first call of the function `trj_line` computes trajectory from initial to intermediate configuration, the second call computes trajectory to the final configuration. In the end, the simulation of the motion is performed in line 12. The manipulator motion (five selected steps) and calculated trajectories for individual joints have been show in fig. 4.7 and 4.8.

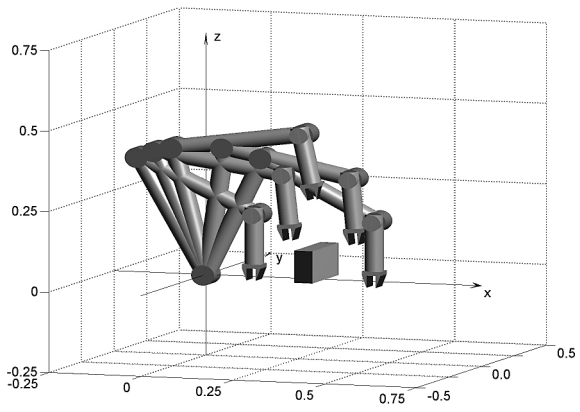

Fig. 4.7. Motion of the manipulator

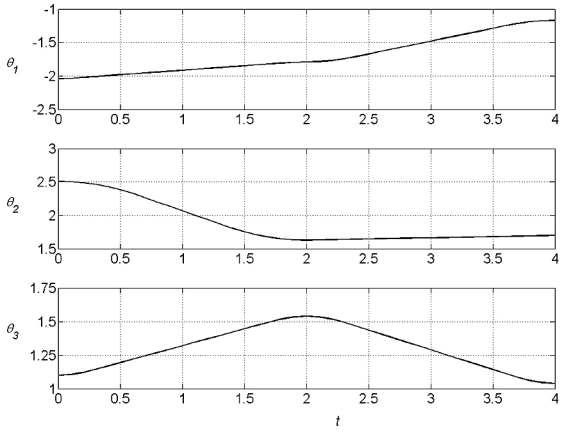

Fig. 4.8. Trajectories for individual joints

5. Conclusions

In the paper a Matlab toolbox for visualization the manipulators and their workspaces in the realistic manner has been presented. To show manipulator in its default 3D appearance only Denavit-Hartenberg parameters are required, it is also possible to create a model of mechanism corresponding to its real appearance using a set of polyhedrons. Additionally, the classical methods of trajectory generation in configuration space have been implemented and resulting motions may be shown using provided functions. The toolbox can be useful for educational purposes, for presentation research results and during designing and verifying the production cells.

The toolbox presented in the paper includes only a set of the basic functions necessary to modeling the manipulators in virtual environment and it is still being developed. In the next versions improvement connections between a structure representing a manipulator and its graphical representation will be implemented. Possibility of simultaneous simulation of multiply robots and other objects with collision detection will also be added. Additionally, a new class of objects representing the mobile robots will be introduced. That robots type according to the authors knowledge has not been implemented in other Matlab toolboxes.

The Robot Toolbox is freely available as the .zip archive file from its website: `http://www.uz.zgora.pl/~gpajak/rtoolbox`. After downloading the archive has to be

unzipped to any folder on the computer and path to this folder must to be added to the Matlab search path. The documentation for each toolbox function is available through the Matlab function `help`, all provided examples may be run after calling function `demo_robot`.

Nomenclature

- a_i - manipulator link length of the *i*-th joint
- d_i - manipulator *i*-th joint offset
- ${}^{R}\mathbf{P}_{SORG}$ - translation vector describing position of origin of coordinate system $\{S\}$ in the reference system $\{R\}$
- ${}^{R}_{S}\mathbf{R}$ - rotation matrix describing orientation of coordinate system $\{S\}$ in the reference system $\{R\}$
- ${}^{R}_{S}\mathbf{T}$ - homogeneous transformation matrix describing position and orientation of coordinate system $\{S\}$ in the reference system $\{R\}$
- α_i - manipulator link twist of the *i*-th joint
- θ_i - manipulator *i*-th joint angle

References

ABB (2011): *RobotStudio: of offline robot programming for ABB robots*. - http://www.abb.com.

Bingul Z., Koseeyaporn P., Cook G. E. (2002): *Windowsbased robot simulation tools*. - 7th International Conference on Control, Automation, Robotics and Vision, Singapore.

Bruyninckx H. (2001): *Open robot control software: the OROCOS project*. - IEEE International Conference on Robotics and Automation (ICRA), pp. 2523–2528.

Corke P. I. (1996): *A robotics toolbox for MATLAB*. - IEEE Robotics and Automation Magazine, vol. 3, No.1, pp. 24-32.

Craig J. J. (1989): *Introduction to Robotics*. - Cambridge, Massachusetts: Addison Wesley.

Fanuc (2011): *ROBOGUIDE: a family of offline robot simulation software*. - http://www.fanucrobotics.com.

Fu K. S., Gonzalez R. C., Lee C. S. G. (1987): *Robotics. Control, Sensing, Vision and Intelligence*. – New York: McGraw-Hill.

Gourdeau R. (1997): *Object oriented programming for robotic manipulators simulation*. - IEEE Robotics and Automation Magazine, vol.4, No.3.

Kucuk S., Bingul Z. (2010): *An off-line robot simulation toolbox*. - Computer Application in Engineering Education, 18, pp. 41-52, DOI 10.1002/cae.20236.

Legnani G. (2006): *SPACELIB: a software library for the Kinematic and dynamic analysis of systems of rigid bodies*. - U. di Brescia. http://www.ing.unibs.it/~glegnani/

López-Nicolás G., Romeo A., Guerrero J. J. (2009): *Project Based Learning of Robot Control and Programming*. - ICEE & ICEER, pp. 1-8.

MathWorks, Inc. (2010): *Matlab Programming Fundamentals* – Natick, Massachusetts.

Paul R. P. (1981): *Robot Manipulators: Mathematics, Programming and Control*. - Cambridge, Massachusetts: MIT Press.