An Application Example of Webots in Solving Control Tasks of Robotic System

Petar Mandić
PhD Student
University of Belgrade
Faculty of Mechanical Engineering

Mihailo Lazarević
Full Professor
University of Belgrade
Faculty of Mechanical Engineering

This paper presents some of the capabilities ofWebots-a robotics simulation software. Using Webots as the development environment one can obtain model, program as well as simulate robots. First, key features and components of Webots are described and then it is presented how to construct a model of robotic system in it. Then, a control system is designed based on mathematical model of robot system in order to solve the problem of positioning of the end-effector. Results obtained in Webots enviroments are compared with those from Matlab/Simulink, so one can confirm the control system design procedure and accuracy of physics simulation. An example of more complex task that the robot manipulator needs to execute is given in the remainder of this paper. It is a so called Tower of Hanoi problem, where is particularly solved the inverse kinematics problem in detail. A part of C programming code which has been used for controlling the robot in Webots is explained at the end.

Keywords: *simulation software-Webots, robot control, simulation.*

1. INTRODUCTION

Robots today are making a considerable impact on many aspects of modern life, from manufacturing to healthcare. Mobile robots, underwater and flying robots, robot networks, surgical robots, and others are playing increasing roles in society. Unlike the industrial robotics domain where the workspace of machines and humans can be segmented, applications of intelligent machines that work in contact with humans are increasing, which involve e.g. haptic interfaces and teleoperators, cooperative material-handling, power extenders and such high-volume markets as rehabilitation, physical training, entertainment. In that way, robotic systems are more and more ubiquitous in the field of direct interaction with humans, in a so called friendly home environment. For example, providing contact sensing on the whole body of a robot is a key feature to increase the safety level of physical human-robot interaction.

Moreover, robots are also used in bioengineering research, for example, robots and computers control the skin-making process, which takes place in a sterile, climate-controlled setting. One of these robotic systems capable of operating in human friendly environments is *NeuroArm robot*. It is an integral part of the Laboratory of Mechanics at Faculty of Mechanical Engineering in Belgrade (Figure 1). This robotic arm possesses seven degrees of freedom, which is described with six parameters for rotating and one parameter for translating (Figure 2). Within NeuroArm Manipulator System there are a rich set of options that enable scientists and engineers to configure your robot that will meet the needs [1,2]. There are 4 basic methods of control given robotic system: Windows GUI Tool, Open source serial protocol & C library, Universal real time behaviour interface, and Webots simulation package. In the remainder of this article, attention will be given to the Webots simulation package. Namely, with Webots users can design 3D models of robots and test their behaviour in realistic simulation environment.

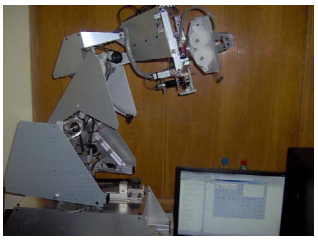


Figure 1. NeuroArm robot system

Simulations play an important role in robotics research. In comparison with real robot investigations, simulations are easier to set up, less expensive, faster, and more convenient to use, and allow the user to perform experiments without the risk of damaging the robot. Building up a new robot model and setting up experiments only takes a few hours and control programs can be tested extensively on a host computer. Simulators are especially preferred when using time consuming algorithms for robot controllers.

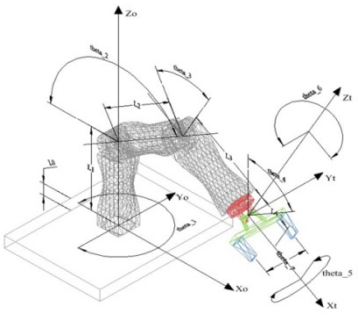


Figure 2. Model of NeuroArm with 7 degrees of freedom

Received: September 2012, Accepted:December 2012
Correspondence to: Petar Mandić, M.Sc.
Faculty of Mechanical Engineering,
Kraljice Marije 16, 11120 Belgrade 35, Serbia
E-mail: petarmandic84@gmail.com

Webots is a three dimensional mobile robot simulator. The Webots simulation software provides the user with a rapid prototyping environment for modelling, programming and simulating mobile robots [3,4]. With Webots the user can design complex robotic setups, with one or several, similar or different robots, in a shared environment. The properties of each object, such as shape, colour, texture, mass, friction etc., are chosen by the user. Robots can have different locomotion schemes: wheeled robots, legged robots or flying robots. They may be equipped with a number of sensor and actuator devices, such as distance sensors, servos, touch sensors, cameras, emitters, receivers etc.

Simulating complex robotic devices requires precise physics simulation. Webots relies on ODE (Open Dynamic Engine) to perform accurate physics simulation. Webots robotics simulation software is used in more than 930 universities and research centres worldwide for both research and educational purposes and among them is Laboratory of Mechanics at Faculty of Mechanical Engineering,University of Belgrade.

A Webots simulation is composed of the following items:

1. a Webots world file (.wbt) that defines one or several robots and their environment.

2. one or several controller programs for the above robots.

3. optionally, Webots may contain Supervisor, a privileged type of Robot that can execute operations that can normally only be carried out by a human operator and not by a real robot, for example moving the robot to a random position, making a video capture of the simulation, etc.

A world file is a 3D description of the properties of robots and of their environment. It contains a description of every object: position, orientation, geometry, physical properties etc. Worlds are organized as hierarchical structures where objects can contain other objects. For example, a robots can contain two wheels, a distance sensor and a servo which itself contains a camera, etc.

A controller is a computer program that controls a robot specified in a world file. Controllers can be written in any of the programming languages supported by Webots: C, C++, Java, Python or Matlab. When a simulation starts, Webots launches the specified controllers, and it associates the controller processes with the simulated robots. A majority of Webots controllers are written in C programming language. The C API (Application Programming Interface) is composed of a set of about 200 C functions that can be used in C or C++ controller code.

Matlab programming language is also of great importance for us, because it allows easy matrix manipulation, plotting of functions and data, implementation of algorithms etc.

Webots GUI (Graphical User Interface) is composed of four principal windows (see figure 3): the 3D window that displays and allows to interact with the 3D simulation, the Scene Tree which is a hierarchical representation of the current world, the Text Editor that allows to edit source code, and finally, the Console that displays both compilation and controller outputs.

More information about Webots, how to design a 3D robot and its environment, write a robot controller etc., can be found in [5].

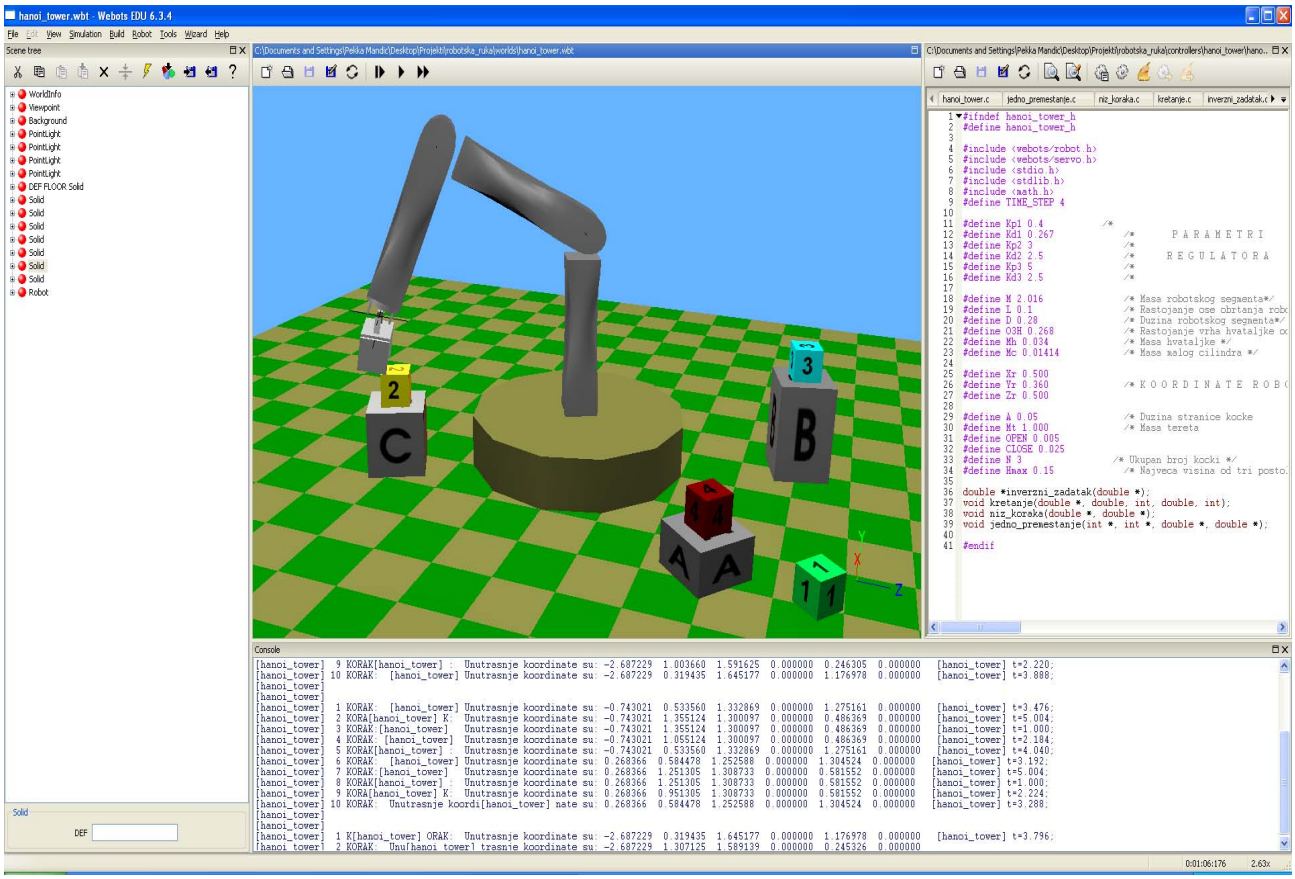


Figure 3. Webots Graphical User Interface

2. WEBOTS MODEL OF ROBOTIC SYSTEM

In this section we will describe how to create a 3D model of robot manipulator in Webots.As mentioned earlier, Webots world file contains all the information related to your simulation, i.e. where are the objects, how do they look like, how do they interact with each other etc. A world is defined by a tree of nodes. Each node has some customizable properties called fields. The Webots Node Chart outlines all the nodes available to build Webots worlds. For more information on the available nodes and world file format, see [6]. All objects in Webots have a hierarchical structure. Figure 4 shows the hierarchy relations between all nodes of the NeuroArm robot manipulator.

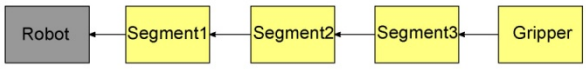


Figure 4. Model node hierarchy

For example, if we want to add a second link (Segment2) to the robot, we need to create a Servo node in the children field of the first link. The Webots Servo node models a servo motor, which is adequate for the simulation of robot's joints. It adds a joint (one degree of freedom) in a mechanical simulation. Hence, our 3D model of robot manipulator will have as much Servo nodes as it has degrees of freedom. Servo may be either rotational or linear, and it also simulates a position sensor. Also, Webots allows the addition of spring and damping behaviour to the Servo.

Servo node contains customizable fields which allow us to change characteristics of a servo device (see Figure 5). Here will be mentioned some of those fields.

Translation and rotation fields define the translation and rotation from the parent coordinate system to the children's coordinate system. In our example (Segment2) these fields define the relative position of the second link with respect to the first link. In children field we define:

- geometry and appearance of the second link
- next component of the robot structure (Segment3).

Field name contains the appropriate name of the second link.

The boundingObject specifies the geometrical primitives used for collision detection. If this field is NULL (deactivated), then no collision detection is performed and second link can pass through any other object, e.g. floor, obstacles and other robots.

Physics field is used to model the physical properties the second link. This field specifies the mass, the center of gravity and the mass distribution, thus allowing the physics engine to create a body and compute realistic forces. If the physics field is NULL then Webots simulates this object in kinematics mode.

The type field is a string which specifies the servo type, and may be either rotational or linear.

The maxVelocity field specifies both the upper limit and the default value for the servo velocity.

The maxForce field specifies both the upper limit and the default value for the servo motor force. The motor force is the torque/force that is available to the motor to perform the requested motions. A small maxForce value may result in a servo being unable to move to the target position because of its weight or other external forces.

The control P field specifies the initial value of the P parameter, which is the proportional gain of the Webots servo controller. A high P results in a large response to a small error, and therefore a more sensitive system. However, by setting P too high, the system may become unstable. With a small P, more simulation steps are needed to reach the target position, but the system is more stable.

As an alternative to the embedded Webots P controller, the user can design a custom controller. This allows the user to directly specify the amount of torque/force that must be applied by a servo based on, e.g. a PID controller. Exactly in this way, we designed a 3 PD controllers for robot's links, as we'll see in the next section.The position field represents the current position of the servo. For a rotational servo, this field represents the current rotation angle in radians, while for a linear the magnitude of the current translation in meters.

The minPosition and maxPosition fields define the soft limits of the servo. When both of the fields are zero, the soft limits are deactivated.

The minStop and maxStop fields define the hard limits of the servo. When both of the fields are zero, the hard limits are deactivated.

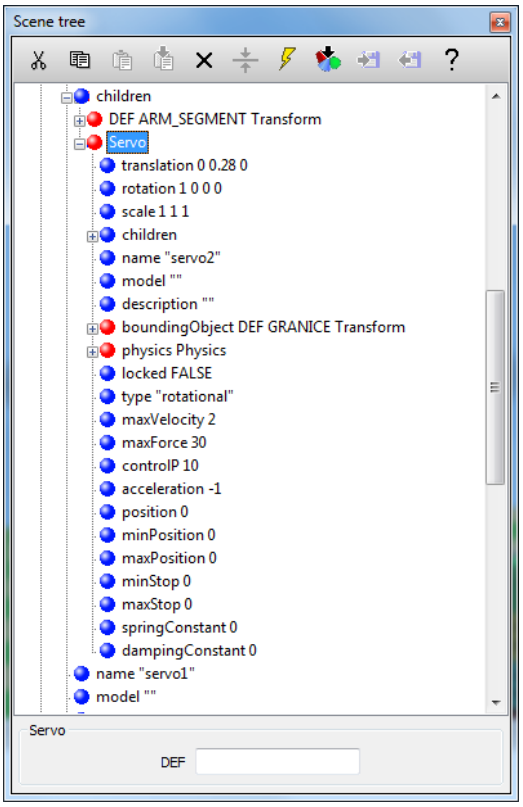


Figure 5. Scene Tree window

So, when we designed the second robot's link, we repeat the same procedure for the next object in the hierarchical structure, i.e. the third link: in the children field of the second link we create a new Servo node.

3. POSITION CONTROL OF ROBOTIC SYSTEM: WEBOTS VERSUS SIMULINK

In this section we will first derive a mathematical model of robotic system and then we will consider the problem of position control, followed by design of robot controller and selection of its parameters. Finally, simulation of robot manipulator in Webots is made, and its results are compared with those obtained from Matlab/Simulink.

3.1 Dynamical equations of robotic system

From a mechanical point of view a robotic system is considered as an open linkage consisting of $n+1$ rigid bodies $[V_j]$ interconnected by n one-degree-of–freedom joints forming kinematical pairs of the fifth class where the robotic system possesses n degrees of freedom. Joints can be essentially of two types: revolute and prismatic. The whole structure forms a kinematic open-chain structure. The configuration of the robot mechanical model can be defined by the vector of the joint (internal) generalized coordinates q of the dimension n, $(\boldsymbol{q})=(q^1,q^2,\ldots,q^n)^{\mathrm{T}}$, with the relative angles of rotation (in case of revolute joints) and relative displacements (in case of prismatic joints), [7].

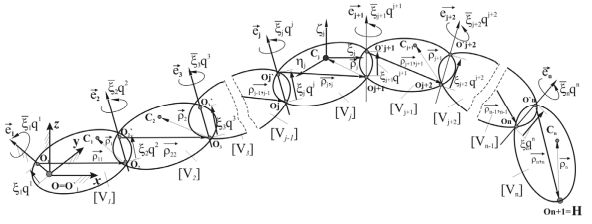


Figure 6. Open-chain structure of a robotic multi-body system

The geometry of the system has been defined by the unit vectors $\boldsymbol{e}_j$, $j=1,2,\ldots n$ where the unit vectors $\boldsymbol{e}_j$ describe the axis of rotation (translation) of the j-th segment with respect to the previous segment as well as the position vectors $\boldsymbol{\rho}_j$ and $\boldsymbol{\rho}_{jj}$ usually expressed in local coordinate systems $C_j\xi_j\eta_j\zeta_j$ connected with the bodies $\left(\boldsymbol{\rho}_j^{(j)}\right)$, $\left(\boldsymbol{\rho}_{jj}^{(j)}\right)$. With $\boldsymbol{\rho}_{jj}=\overrightarrow{O_j'O_{j+1}}$ is denoted a vector between two neighboring joints in robot structure, while a position of a center of mass of i-th link is expressed by a vector $\boldsymbol{\rho}_j=\overrightarrow{O_{j+1}C_j}$. The parameters $\xi_j,\overline{\xi}_j=1-\xi_j$ denote the parameters for recognizing the joints $\xi_j,\overline{\xi}_j=1-\xi_j$, $\xi_j=1$-prismatic, 0-revolute. For the entire determination of this mechanical system, it is necessary to specify the masses m_j and the tensors of inertia J_{Cj} expressed in local coordinate systems. In order that the kinematics of the robotic system may be described, the points O_j, O_j' are noticed somewhere at the axis of the corresponding joint (j) such that they coincide in the reference configuration. The point O_j is immobile with respect to the $(j-1)$-th segment and O_j' does so with respect to the $j-$th one; obviously, for a revolute joint (j), the points O_j and O_j' will coincide all the time during robotic motion. Here, the Rodriguez` method [8] is proposed for modeling the kinematics and dynamics of the robotic system in contrast to Denavit-Hartenberg's method. In our case, it is presented a robotic system with 3 revolute joints, i.e. with $n=3$ DOFs (see Figure 7) where an end-effector orientation will not be considered here, but only its position in space. As shown in figure below, let *Oxyz* be the orthonormal fixed frame.

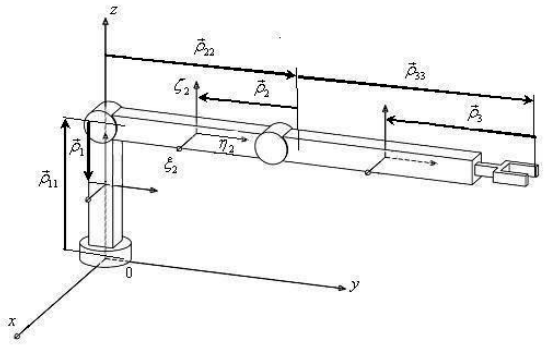


Figure 7. Robotic system in referent configuration with 3DOFs

The end-effector point H is of great importance for us because we want to control its position in space. Besides, the position vector of the end-effector $\boldsymbol{r}_H$ can be written as a multiplication of the matrices of transformation $[A_{i-1,i}]$, the vectors $\boldsymbol{\rho}_{jj}$ and $\xi_j q^j \boldsymbol{e}_j$ and it is expressed by [9]:(1)

$$\begin{aligned}\boldsymbol{r}_H(q)&=\sum_{j=1}^{n=3}\left(\boldsymbol{\rho}_{jj}+\xi_j q^j\boldsymbol{e}_j\right)=\\&=\sum_{j=1}^{n=3}\left(\prod_{i=1}^{j}[A_{i-1,i}]\right)\left(\left(\boldsymbol{\rho}_{jj}^{(j)}\right)+\xi_j q^j\left(\boldsymbol{e}_j^{(j)}\right)\right)\end{aligned}\tag{1}$$

where the appropriate Rodriguez' matrices of transformation are

$$[A_{i-1,i}]=[I]+\left[e_i^{d(i)}\right]^2\left(1-\cos q^i\right)+\left[e_i^{d(i)}\right]\sin\left(q^i\right)\tag{2}$$

as well as $\left[e_i^{d(i)}\right]$ denotes a skew symmetric matrix

$$\left(e_i^{(i)}\right)=\left(e_{\xi i},e_{\eta i},e_{\zeta i}\right)^T,\ \left[e_i^{d(i)}\right]=\begin{bmatrix}0&-e_{\zeta i}&e_{\eta i}\\e_{\zeta i}&0&-e_{\xi i}\\-e_{\eta i}&e_{\xi i}&0\end{bmatrix}\tag{3}$$

To derive the equations of motion of the robotic system, we use Lagrange's equations of second kind:

$$\frac{d}{dt}\left(\frac{\partial E_k}{\partial\dot{q}_\gamma}\right)-\frac{\partial E_k}{\partial q_\gamma}=Q_\gamma\quad\gamma=1,2,3\tag{4}$$

which can be expressed in the identical covariant form as follows [8]:

$$\sum_{\alpha=1}^{n} a_{\gamma\alpha}\ddot{q}_{\alpha} + \sum_{\alpha=1}^{n}\sum_{\beta=1}^{n} \Gamma_{\alpha\beta,\gamma}\dot{q}_{\alpha}\dot{q}_{\beta} = Q_{\gamma} \quad \gamma = 1,2,3 \quad (5)$$

where the coefficients $a_{\alpha\beta}$ are the covariant coordinates of the basic metric tensor $\left[a_{\gamma\alpha}\right] \in R^{n\times n}$ and $\Gamma_{\alpha\beta,\gamma}$ $\alpha,\beta,\gamma = 1,2,...,n$ presents Christoffel symbols of the first kind. The generalized forces Q_{γ} can be presented in the following expression (6) where $Q_{\gamma}^{c}, Q_{\gamma}^{g}, Q_{\gamma}^{\beta}, Q_{\gamma}^{w}, Q_{\gamma}^{a}$ denote the generalized spring forces, gravitational forces, viscous forces, semi-dry friction and generalized control forces, respectively.

$$Q_{\gamma} = Q_{\gamma}^{c} + Q_{\gamma}^{g} + Q_{\gamma}^{\beta} + Q_{\gamma}^{w} + Q_{\gamma}^{a}, \quad \gamma = 1,2,...,n \quad (6)$$

In summary, the equations of motion of our robotic system can be rewritten in compact matrix form:

$$A(\boldsymbol{q})\ddot{\boldsymbol{q}} + C(\boldsymbol{q},\dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{Q}^{a} \quad (7)$$

where

$A(\boldsymbol{q}) = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \in \mathbb{R}^{3\times 3}$ is basic metric tensor,

$C(\boldsymbol{q},\dot{\boldsymbol{q}}) \in \mathbb{R}^{3}$ - vector which takes care configuration of robotic system and velocity-dependent effects

$\boldsymbol{Q}^{g}(q) = -\boldsymbol{g}(\boldsymbol{q}) \in \mathbb{R}^{3}$ - vector of generalized gravitational forces

$\boldsymbol{Q}^{a} = \begin{pmatrix} M_1 & M_2 & M_3 \end{pmatrix}^{T} \in \mathbb{R}^{3}$ - vector of generalized control forces.

3.2 Position control system design

The task of position control of robotic system means to determine the three components of vector of external torques $\boldsymbol{Q}^{a}$ that allow execution of motion $\boldsymbol{q}(t)$ so that

$$\lim_{t\to\infty} \boldsymbol{q}(t) = \boldsymbol{q_d} \quad (8)$$

where $\boldsymbol{q_d}$ denotes the vector of desired joint trajectory variables. To solve the position control problem we will consider a class of robotic system called *computed torque controllers* [10]. As a result, a complicated nonlinear controls design problem will be converted into a simple design problem for a linear system consisting of decoupled subsystems. To do this, let's look at (7) and choose $\boldsymbol{Q}^{a}$ as follows:

$$\boldsymbol{Q}^{a} = A(\boldsymbol{q})\boldsymbol{Q}_{R} + C(\boldsymbol{q},\dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q}) \quad (9)$$

Substituting (9) into the (7) yields

$$\ddot{\boldsymbol{q}}(t) = \boldsymbol{Q}_{R} \quad (10)$$

where $\boldsymbol{Q}_{R}$ denotes output of PD controller. The resulting control scheme appears in Figure 8 (nonlinear term $N(\boldsymbol{q},\dot{\boldsymbol{q}})$ is equivalent to $C(\boldsymbol{q},\dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q})$).

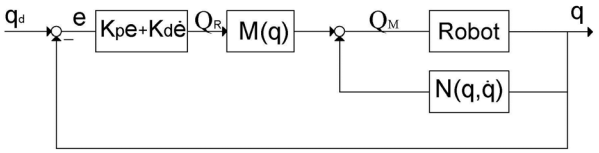


Figure 8. Block diagram scheme of proposed robot control

One way to select control signal $\boldsymbol{Q}_{R}$ is as the proportional plus derivative (PD) feedback

$$\boldsymbol{Q}_{R} = K_{P}\boldsymbol{e}(t) + K_{D}\dot{\boldsymbol{e}}(t) \quad (11)$$

where $\boldsymbol{e}(t) = \boldsymbol{q}_{d}(t) - \boldsymbol{q}(t)$ is output error. Now (10) becomes

$$\ddot{\boldsymbol{q}}(t) + K_{D}\dot{\boldsymbol{q}}(t) + K_{P}\boldsymbol{q}(t) = K_{P}\boldsymbol{q}_{d}(t) + K_{D}\dot{\boldsymbol{q}}_{d}(t) \quad (12)$$

The closed loop characteristic polynomial is

$$f(s) = \prod_{i=1}^{3}\left(s^{2} + K_{Di}s + K_{Pi}\right) \quad (13)$$

and the systems is asymptotically stable as long as the K_{Pi}, K_{Di} are all positive for $i = 1,2,3$. Also, it is undesirable for the robot to exhibit overshoot, since this could cause impact at the surface of a box. Therefore, gain matrices are selected for critical damping $\xi = 1$. In this case

$$\begin{aligned} K_{P} &= diag\{36,36,36\} \\ K_{D} &= diag\{12,12,12\} \end{aligned} \quad (14)$$

We created a model of robot manipulator in Matlab Simulink based on previous equations. System's response for $q_{di} = h(t)$, $i = 1,2,3$ is recorded, and results are compared with those obtained from Webots simulation environment and Figure 9 illustrates it.

We can see that system's responses obtained from Webots and Simulink overlay each other, which confirms that mathematical model of robotic system is well derived. Control torque values are shown on the right hand side of Figure 9, so we can get insight into their magnitudes. In that way, it could be helpful if we would to choose actuators.

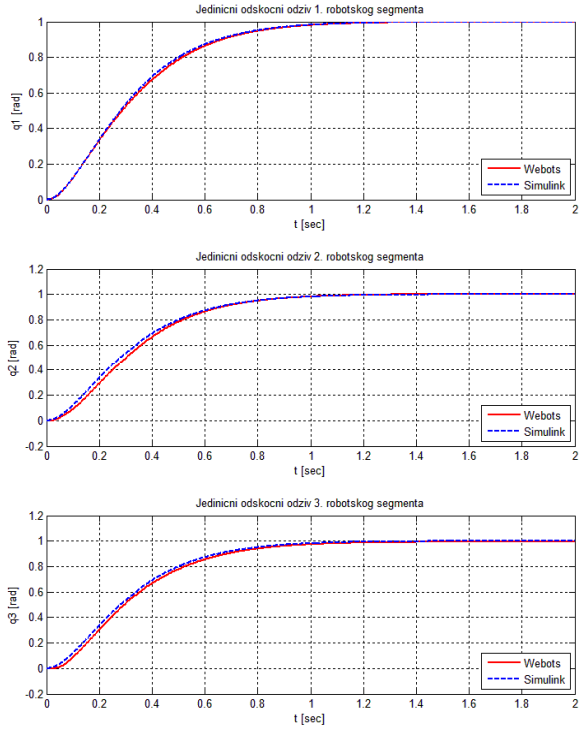
4. TOWER OF HANOI PROBLEM

Further, we proceed with executing more complex tasks in Webots environment. Firstly, it will be given a description of a robot task which needs to execute. Next, the task is divided into three independent subproblems which need to be solved. Particularly, we present and resolve the inverse kinematics problem in detail. At the end, a part of programming code needed for controlling the robot is given and explained.

4.1 Description of the problem

Robot needs to solve a so called Tower of Hanoi problem, by which is meant the following: it consists of three platforms (A, B and C) and an arbitrary number of boxes labelled with 1, 2, 3 etc.

Boxes can be moved on any of three platforms. At the beginning, all boxes are on platform A, in a neat



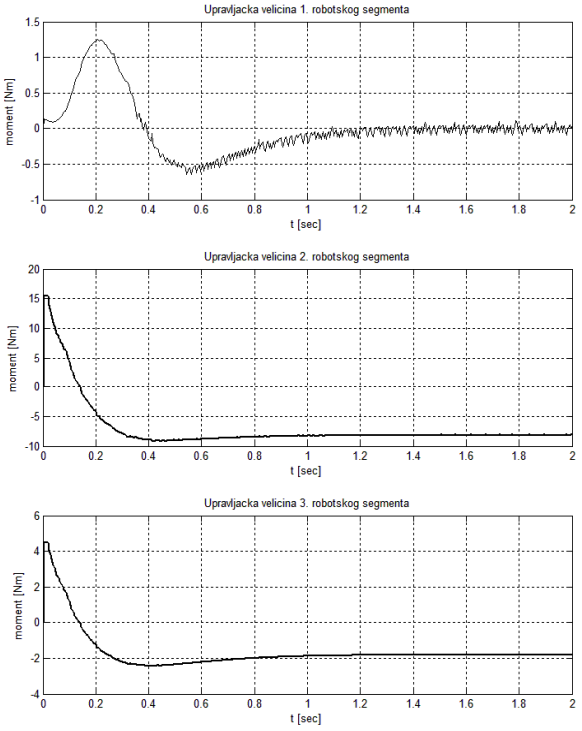


Figure 9. System's step response

stack in ascending order, i.e. box with the smallest number is on top, see Figure 10. The objective of the task is to move the entire stack to platform C, obeying the following rules:

- only one box can be moved at a time.
- each move consists of taking the upper box from one of the platforms and moving it onto another platform, on top of the other boxes that may already be present on that platform.
- no box may be placed on top of a box with smaller number.

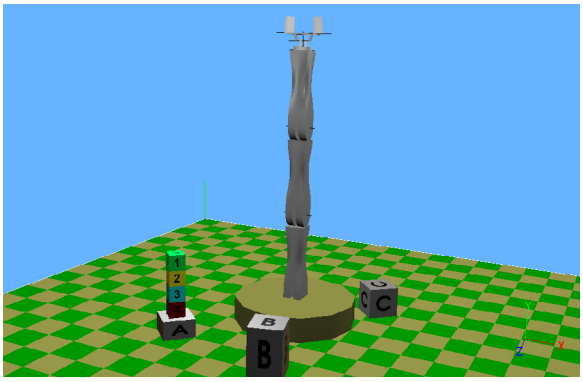


Figure 10. Tower of Hanoi problem setup

In order for robot to execute successfully Tower of Hanoi problem, three independent subproblems need to be solved:

- algorithm for moving the boxes,
- inverse kinematics problem, and
- control design problem.

A 3 DOFs robotic system presented in previous section will be used here. Additionally, we will need two more DOFs: one for the end-effector's orientation, and one for gripping the boxes. Embedded Webots controller will be used for controlling these additional joints.

4.2 Algorithm for moving the boxes

The Tower of Hanoi problem seems impossible to solve for larger number of boxes, yet is solvable with a simple algorithm. There are number of different algorithms that solve the problem, here will be presented a so called iterative solution because it is very suitable for programming. Whether the number of boxes is odd or even, next steps need to be followed:

a. for an even number of boxes
- make the legal move between platforms A and B
- make the legal move between platforms A and C
- make the legal move between platforms B and C

b. for an even number of boxes
- make the legal move between platforms A and C
- make the legal move between platforms A and B
- make the legal move between platforms B and C

The number of moves required to solve a Tower of Hanoi task is $2^N - 1$, where N is the number of boxes.

4.3 Inverse kinematics problem

The inverse kinematics problem consists of the determination of the joint variables corresponding to a given end-effector position and orientation. In our case, for a desired end-effector position, i.e. position of the box with respect to the inertial reference frame, we need to determine four joint variables, see Figure 11. Thus, in this subsection we present procedure for solving the given inverse kinematics problem which can be divided into three inverse kinematics subtasks due to its nature.

The position of a point O_2 with respect to the reference frame $Oxyz$ is expressed by components

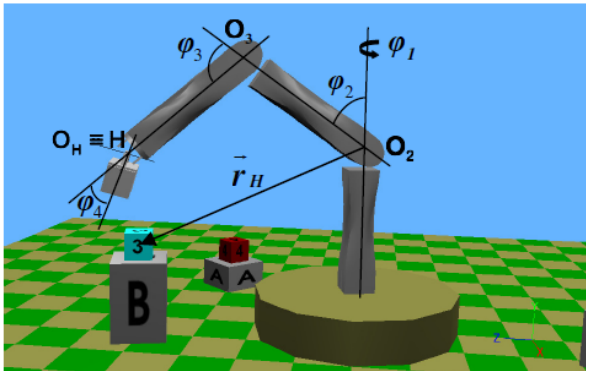


Figure 11. Inverse kinematics problem setup

x_R, y_R, z_R along the frame axes. In the same way, let x_T, y_T, z_T be the components that denote the box position with respect to the $Oxyz$ frame. Then, we can write:

$$\boldsymbol{r}_R + \boldsymbol{r}_H = \boldsymbol{r}_T \tag{15}$$

respectively,

$$\boldsymbol{r}_H = \boldsymbol{r}_T - \boldsymbol{r}_R \tag{16}$$

Above equation can be expressed by three scalar equations:

$$\begin{aligned} x_H &= x_T - x_R, \quad y_H = y_T - y_R, \\ z_H &= z_T - z_R \end{aligned} \tag{17}$$

First, joint variable φ_1 can now be determined as (see Figure 12):

$$\sin\varphi_1 = \frac{x_H}{\sqrt{x_H^2 + z_H^2}}, \cos\varphi_1 = \frac{z_H}{\sqrt{x_H^2 + z_H^2}} \tag{18}$$

$$\varphi_1 = \operatorname{atan2}\left(\sin\varphi_1, \cos\varphi_1\right) \tag{19}$$

Now we need to determine variables φ_2 and φ_3. Let's look at Figure 13, where robot is in position ready to grip the box where coordinate y_{fH} and $r_{xz\,fH}$ determine the

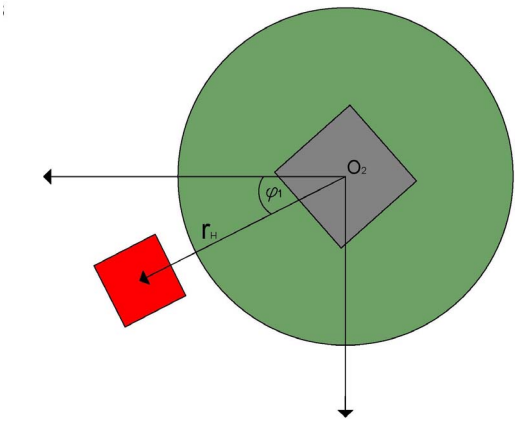


Figure 12. Determination of the joint variable φ_1

final position of end-effectors' point H:

$$\begin{aligned} r_{xz\,fH} &= \sqrt{x_H^2 + z_H^2} \\ y_f &= y_H + \frac{A}{2} + 0.022 \end{aligned} \tag{20}$$

where A represents a size of the box. Based on figure below we can write [11,12]:

$$\begin{aligned} r_{xzfH} &= D\sin\varphi_2 + \overline{O_3H}\sin\left(\varphi_2 + \varphi_3\right) \\ y_f &= D\cos\varphi_2 + \overline{O_3H}\cos\left(\varphi_2 + \varphi_3\right) \end{aligned} \tag{21}$$

where $\overline{O_2O_3} = D$. Squaring and summing above equations, it yields:

$$r_{xzfH}{}^2 + y_f^2 = D^2 + \overline{O_3H}^2 + 2D\overline{O_3H}\cos\varphi_3 \tag{22}$$

so that, it follows

$$\cos\varphi_3 = \frac{r_{xzfH}{}^2 + y_f^2 - D^2 - \overline{O_3H}^2}{2D\overline{O_3H}} \tag{23}$$

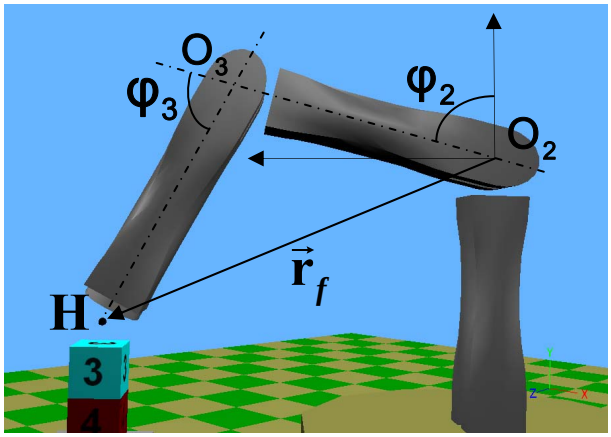


Figure 13. Determination of the joint variables φ_2 and φ_3

The existence of a solution obviously imposes that $|\cos\varphi_3| \le 1$, otherwise the given point would be outside the arm reachable workspace. Then, we have two solutions of the inverse kinematics problem, see Figure 14. We choose when $\varphi_3 > 0$, i.e.

$$\sin\varphi_3 = +\sqrt{1 - \cos^2\varphi_3} \tag{24}$$

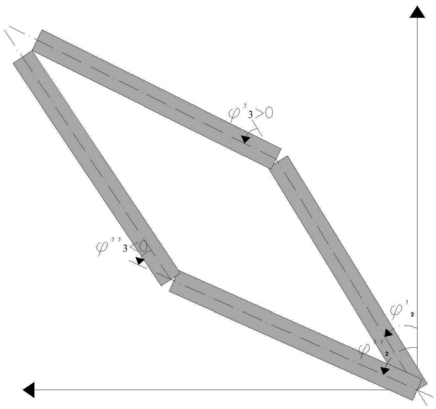


Figure 14. Two admissible postures for φ_2 and φ_3

Hence, the angle φ_3 can be computed as

$$\varphi_3 = \text{atan2}\left(\sin\varphi_3, \cos\varphi_3\right) \tag{25}$$

Having determined φ_3, the angle φ_2 can be calculated as follows. Substituting φ_3 into (21) yields an algebraic system of two equations in the two unknowns $\sin\varphi_2$ and $\cos\varphi_2$, whose solution is

$$\begin{aligned}\sin\varphi_2 &= \frac{\left(D + \overline{O_3H}\cos\varphi_3\right) r_{xzfH} - \overline{O_3H}\sin\varphi_3 y_f}{{r_{xzfH}}^2 + y_f^2} \\ \cos\varphi_2 &= \frac{\left(D + \overline{O_3H}\cos\varphi_3\right) y_f + \overline{O_3H}\sin\varphi_3 r_{xzfH}}{{r_{xzfH}}^2 + y_f^2}\end{aligned} \tag{26}$$

and $\varphi_2 = \text{atan2}\left(\sin\varphi_2, \cos\varphi_2\right)$. Finally, the angle φ_4 can be computed based on:

$$\varphi_2 + \varphi_3 + \varphi_4 = \pi \ rad \Rightarrow \varphi_4 = \pi - \varphi_2 - \varphi_3 \tag{27}$$

because the end-effector needs to form angle of 180° with vertical axis in a moment of grasping of given box.

4.4 Control design problem

Robot model described in previous section will be used here. For position control of end-effector we will use computed torque control given by (11) and gain matrices selected as (14).

4.5 Controller programming

As mentioned earlier, controller is a programming code used to control a given robot. This section introduces the basic concepts of C programming with Webots [13]. A part of controller used for solving Tower of Hanoi problem is explained and given at the end.

Every programming language will yield exactly the same simulation results as long as the sequence of function/method calls does not vary. Hence the concepts explained here with C example also apply to C++/Java/Python/Matlab.

Some of the basic functions necessary for Webots to work normally are:

- wb_robot_init(). A call to this initialization function is required before any other C API function call, because it initializes the communication between the controller and Webots.

- prior to using a device, it is necessary to get the corresponding device tag. This is done using the wb_robot_get_device() function. For example, wb_robot_get_device("servo2") adds the device tag to the second robot link.

- each sensor must be enabled before it can be used. If a sensor is not enabled it returns undefined values. Enabling e.g. a position sensor of the second robot link is achieved using the wb_servo_enable_position (servo2, TIME_STEP), where TIME_STEP specify the time interval (in milliseconds) between two updates of the sensor's data.

- a call to wb_servo_get_position() retrieves the latest value of the given sensor.

Usually the highest level control code is placed inside a for or a while loop. Within that loop there is a call to the wb_robot_step() function. This function synchronizes the controller's data with the simulator. The function wb_robot_step() needs to be present in every controller and it must be called at regular intervals, therefore it is usually placed in the main loop. Argument in this function specifies the duration of the control steps expressed in milliseconds. The control step is the duration of an iteration of the control loop. If control step is 4, Webots shall compute 4 milliseconds of simulation and then return. If a controller does not call wb_robot_step() the sensors and actuators won't be updated and the simulator will block.

To control a motion, it is generally useful to decompose that motion in discrete steps that correspond to the control step. At each iteration a new torque value is computed according to a control law described above. For example, if we want to control a motion of the second link, a call to wb_servo_set_force(servo2, moment_2) at each iteration is required, where second argument (moment_2) denotes control torque. The desired position will be reached only if the physics simulation allows it, that means, if the specified motor torque is sufficient and the motion is not blocked by obstacles, external forces or the servo's own spring force.

At Figure 15 it is shown a part of controller's code used for control of robot's motion. As we can see, function wb_robot_step() is called first at each iteration of a while loop. Then, a call to wb_servo_get_position() gives us current position of robot links. Torque values are computed based on computed torque control law. This control algorithm does not apply to servo4 (rotation of end- effector) because it uses Webots embedded controller to control its rotational motion. When robot links reach desired position or time reserved for that motion elapses, robot exits the while loop and goes onto the next step in simulation.

5. CONCLUSION

In this paper we introduced Webots robotics simulation software. Its key features, possibilities and advantages over real experiments are presented. The procedure how to construct a 3D model of robot manipulator in Webots is given. Then its mathematical model is derived, followed by the design of position control system. Results obtained in Webots and Matlab Simulink environments are compared. Finally, a simulation of robot executing Tower of Hanoi task is given in last section.

One of advantages of Webots is that user doesn't need to derive mathematical model of robot every time like in Matlab in order to run a simulation accurately (realistically). Namely, Webots relies on powerful ODE (Open Dynamics Engine) to perform an accurate physics simulation. Hence, it is only necessary to specify physical properties of objects like mass, inertia matrix, dimensions, friction coefficients etc., and Webots will take care of rest. This allows Webots to

simulate complex robotic setups with simulations that can run up to 300 times faster than the real robot.

Another important advantage of Webots is the possibility to transfer Webots programming code onto real robot. Webots User Guide [5] explains how to build your own Webots cross-compilation system for real robot.

For future research, solving more complex tasks with different control algorithms will be considered. Also, transfer from simulation to the NeuroArm real robot and comparison of those results will be a subject for future investigations.

ACKNOWLEDGMENT

Authors gratefully acknowledge the support of Ministry of Education, Science and Technological Development of the Republic of Serbia under the project TR 33047 as well as partially supported by projects TR 35006, 41006.

REFERENCES

[1] Lazarević, M., Mandić, P. and Vasić, V.: Some applications of NeuroArm interactive robot and Webots robot simulation tool, in: Proceedings on Accomplishments in Mechanical and Electrical Engineering and Information Technology, 26-29.05.2011, Banjaluka, pp. 923-928.

[2] Miljković, Z., Mitić, M., Lazarević, M. and Babić, B.: Neural Network Reinforcement Learning for Visual Control of Robot Manipulators, Expert Systems with Applications.

[3] Hohl, L., Tellez, R., Michel, O. and Ijspeert, A.J.: Aibo and Webots: Simulation, wireless remote control and controller transfer, Robotics and Autonomous Systems, Vol. 54, pp. 472 -485, 2006.

[4] Michel, O.: Webots™: Professional Mobile Robot Simulation, International Journal of Advanced Robotics Systems, Vol. 1, No. 1, pp. 39 -42, 2004.

[5] Webots User Guide, Cyberbotics Ltd, 2011.

[6] Webots Reference Manual, Cyberbotics Ltd, 2011.

[7] Zorić, N., Lazarević, M. and Simonović A.: Multi Body Kinematics and Dynamics in Terms of Quaternions: Lagrange Formulation in Covariant Form – Rodriguez Approach, FME Transactions, Vol. 38, No. 1, pp. 19 -28, 2010.

[8] Čović, V. and Lazarević, M.: *Robot Mechanics*, Faculty of Mechanical Engineering, Belgrade, 2009, (in Serbian).

[9] Lazarević, M.: Mechanics of Human Locomotor System, FME Transactions, Vol. 34, No. 2, pp. 105 -114, 2006.

[10] Lewis, F.L., Dawson, D.M. and Abdallah, C.T.: *Robot Manipulator Control*, Marcel Dekker, New York, 2004.

[11] Mandić, P.: Diplomski rad, Faculty of Mechanical Engineering, Belgrade, 2011, (in Serbian).

[12] Siciliano, B., Sciavicco, L., Villani, L. and Oriolo, G.: *Robotics*, Springer-Verlag, London, 2009.

[13] Kraus, L.: *C programming language*, Akademska misao, Belgrade, 2008, (in Serbian).

ЈЕДАН ПРИМЕР ПРИМЕНЕ WEBOTS-А У РЕШАВАЊУ ЗАДАТАКА УПРАВЉАЊА РОБОТСКОГ СИСТЕМА

Петар Мандић, Михаило Лазаревић

У овом раду презентован је софтверски пакет Webots за симулацију датог роботског система. Прво су разматране основне особине и компоненте Webots-а. Затим је описан начин како се конструише 3Д модел роботског система у Webots-овом окружењу. У циљу решавања проблема позиционирања роботске хватаљке, одређују се ПД параметри управљачког система, а на основу претходно добијеног математичког модела роботског система. Резултати позиционирања добијени у Webots окружењу су упоређени са резултатима симулације истог добијени у Симулинк/Матлаб окружењу, како би се показала физичка веродостојност симулације. Пример сложенијег задатка који роботски систем треба да реши дат је у наставку. То је тзв. Tower of Hanoi проблем где је посебно детаљно разматран и решен инверзни кинематски задатак. Део програмског кода који је коришћен за управљање симулацијом објашњен је и дат на крају рада.

```
while (1)
                                        {
 wb_robot_step(TIME_STEP);
 t+= TIME_STEP/1000.;
 pos_1= wb_servo_get_position(servo1);
 pos_2= wb_servo_get_position(servo2);
 pos_3= wb_servo_get_position(servo3);
 double eps_1= pos_des_1 - pos_1;
 double eps_2= pos_des_2 - pos_2;
 double eps_3= pos_des_3 - pos_3;

 if(pos_des_1==0 && pos_des_2==0 && pos_des_3==0) { greska/=1000; vreme*=1000;
}
 /* Ova naredba se izvrsava ukoliko je telo van domasaja robotske hvataljke */
 if((fabs(eps_1)<greska && fabs(eps_2)<greska && fabs(eps_3)<greska) ||
t>vreme) {
 printf("t=%.3f;\n", t);
 break;
 }
 double eps1_1;
 if(t!= TIME_STEP/1000.) eps1_1= (eps_1-eps_pr_1)*1000./TIME_STEP;
 else eps1_1=0;
 if (fabs(pos_des_2)<0.1 && pos_des_3==0) moment_1= Kp1*eps_1 + Kd1*eps1_1;
 else moment_1= 10*Kp1*eps_1 + 10*Kd1*eps1_1;
 wb_servo_set_force(servo1, moment_1);
 eps_pr_1=eps_1;
 wb_servo_set_position(servo4, pos_des_4);
 moment_4= wb_servo_get_motor_force_feedback(servo4);
 double eps1_3;
 if(t!= TIME_STEP/1000.) eps1_3= (eps_3-eps_pr_3)*1000./TIME_STEP;
 else eps1_3=0;
 moment_3= Kp3*eps_3 + Kd3*eps1_3;
 moment_3 -= ( M*9.81*L*sin(pos_2+pos_3) + Mh*9.81*0.268*sin(pos_2+pos_3) +
```

Figure 15. A part of controller's code written in C programming language