18

Towards Simulation of Custom Industrial Robots

Cosmin Marcu and Radu Robotin
Technical University of Cluj-Napoca, Department of Automation
Romania

1. Introduction

The simulation tools used in industry require good knowledge of specific programming languages and modeling tools. Most of the industrial robots manufacturers have their own simulation applications and offline programming tools. Even if these applications are very complex and provide many features, they are manufacturer-specific and cannot be used for modeling or simulating custom industrial robots. In some cases the custom industrial robots can be designed using modeling software applications and then simulated using a programming language linked with the virtual model. This requires the use of specific programming languages, a good knowledge of the modeling software and experience in designing the mechanical part of the robot.

Researches within the robots simulation field have been made by various research groups, especially in the field of mobile robots simulation. The results of their researches were complex simulation tools like: *SimRobot* (Laue *et al.*, 2005) capable to simulate arbitrary defined robots in the 3D space together with the sensorial and the actuation systems; *USARSim* (Wang *et al.*, 2003) or the *Urban Search and Rescue Simulator* using as main kernel the *Unreal* game which is very efficient and capable of solving rendering, animation and modelling problems; *UCHILSIM* (Zagal & del Solar, 2004) is a simulator which reproduces the dynamics *AIBO* robots and also simulates the interaction of these robots with objects in the working space; Rohrmeier's industrial robot simulator (Rohrmeier, 2000) simulates serial robots using VRML in a web graphical interface.

Our primary objective was to design and build a simulation system containing a custom industrial robot and an open architecture robot controller in the first part and a simulation software package using well known and open source programming languages in the last part.

The custom industrial robot is a RPPR robot having cylindrical coordinates. In the first part of the project we designed and modeled the robotic structure and we obtained the forward kinematics, inverse kinematics and dynamic equations. From the mechanical point of view, the first joint contains a harmonic drive unit actuated by a DC motor. The two prismatic joints (vertical and horizontal) contain ball-screw mechanisms, both actuated by DC motors. The fourth joint is represented by a DC motor directly linked with the robot gripper.

Another objective was to build a reprogrammable open architecture controller in order to be able to test different types of sensors and communication routines. The robot controller

contains a miniature modular computer, a PIC microcontroller interface board, 4 DC motor driver boards, 4 rotary encoders and 9 digital sensors. For reading the encoders and sensors data we built a multiplexer board capable of reading up to 16 digital or analog inputs in the same time.
From the software point of view the simulation system contains the software applications running on the miniature computer, the program running on the PIC microcontroller board and a remote visual application running on a desktop PC which contains the 3D graphical simulation module.
This chapter presents the technical information and the techniques we used in creating the simulation system together with the experimental results we obtained.

2. The Robot Modelling

2.1 Forward kinematics

In order to build the simulation system we considered a 4 degrees-of-freedom robot structure having two rotation joints and two translation joints. The kinematic scheme of the robot is presented in Fig. 1.

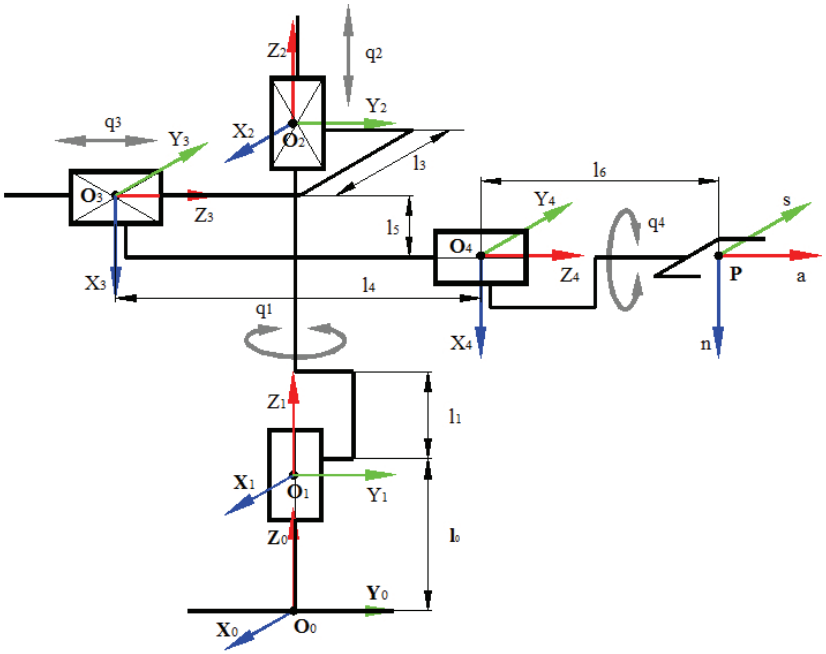

Figure 1. The kinematic scheme of the robot

To obtain the equations which determine the position and the orientation of the gripper relative to the robot base we applied the modified Denavit-Hartenberg method or Craig's method. Therefore, the motion axis for each joint will be Z_n, where n is the index of the corresponding joint. The displacement of each joint is denoted with q and it's measured in millimetres for the translation joints and radians for the rotation joints. The table of D-H parameters will be as follows:

Joint	a_{i-1}	α_{i-1}	d_i	θ_i
1	0	0	l_0	q_1
2	0	0	$l_1 + q_2$	0
3	l_3	$-\pi/2$	$-q_3$	$\pi/2$
4	l_5	0	l_4	q_4
P	0	0	l_6	0

Table 1. Denavit-Hartenberg parameters of the robot

To obtain the position and orientation matrix for each joint relative to the previous joint we applied equation (1):

$$ {}^{i-1}_{i}[T] = \begin{pmatrix} \cos(\theta_i) & \sin(\theta_i)\cdot\cos(\alpha_{i-1}) & \sin(\theta_i)\cdot\sin(\alpha_{i-1}) & -a_{i-1}\cdot\cos(\theta_i) \\ -\sin(\theta_i) & \cos(\theta_i)\cdot\cos(\alpha_{i-1}) & \cos(\theta_i)\cdot\sin(\alpha_{i-1}) & a_{i-1}\cdot\sin(\theta_i) \\ 0 & -\sin(\alpha_{i-1}) & \cos(\alpha_{i-1}) & -d_i \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{1} $$

The position and orientation matrix of each joint relative to the robot base or the fixed coordinate system *{0}* is calculated using (2):

$$ {}^{0}_{n}[T] = \prod_{i=1}^{n} {}^{i-1}_{i}[T] \tag{2} $$

Using the D-H parameters from Table 1 and applying equations (1) and (2) we obtained the position and orientation matrix of the gripper relative to the fixed coordinate system *{0}* or robot base:

$$ {}^{0}_{P}[T] = \begin{pmatrix} -cq_1\cdot sq_4 & -cq_1\cdot cq_4 & -sq_1 & (q_3 - l_4 - l_6)\cdot sq_1 + l_3\cdot cq_1 \\ -sq_1\cdot sq_4 & -sq_1\cdot cq_4 & cq_1 & -(q_3 - l_4 - l_6)\cdot cq_1 + l_3\cdot sq_1 \\ -cq_4 & sq_4 & 0 & l_0 + l_1 - l_5 + q_2 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3} $$

In equation (3) $\sin(q_n)$ and $\cos(q_n)$ are simplified with sq_n and cq_n.

Next step in modelling the robot structure is the determination of velocities and accelerations equations for each joint relative to the robot base.

In order to determine the velocity and acceleration of the gripper we set the angular and linear velocity and acceleration of the fixed coordinate system as being zero.

$$ \begin{aligned} {}^{0}\overline{\omega}_0 &= [0 \quad 0 \quad 0]^T; \quad {}^{0}\dot{\overline{\omega}}_0 = [0 \quad 0 \quad 0]^T; \\ {}^{0}\overline{v}_0 &= [0 \quad 0 \quad 0]^T; \quad {}^{0}\dot{\overline{v}}_0 = [0 \quad 0 \quad g]^T; \end{aligned} \tag{4} $$

To determine the velocity and acceleration of each joint we applied Negrean's iterative method (Negrean *et al.*, 1997), where ${}^{i}\overline{\omega}_i$, ${}^{i}\dot{\overline{\omega}}_i$, ${}^{i}\overline{v}_i$ and ${}^{i}\dot{\overline{v}}_i$ are calculated as follows:

$$ {}^{i}\overline{\omega}_i = {}^{i}_{i-1}[R]\,{}^{i-1}\overline{\omega}_{i-1} + \begin{cases} \dot{q}_i \cdot {}^{i}\overline{k}_i & \text{, if i = Rot.} \\ 0 & \text{, if i = Trans.} \end{cases} \tag{4} $$

$$ {}^{i}\overline{v}_i = {}^{i}_{0}[R]\,{}^{0}\overline{v}_i = {}^{i}_{i-1}[R]\left\{{}^{i-1}\overline{v}_{i-1} + {}^{i-1}\overline{\omega}_{i-1} \times {}^{i-1}\overline{r}_i\right\} + \begin{cases} 0 & \text{,if i = R} \\ \dot{q}_i \cdot {}^{i}\overline{k}_i & \text{,if i = T} \end{cases} \tag{5} $$

$$ {}^{i}\dot{\overline{\omega}}_i = {}^{i}_{i-1}[R]\,{}^{i-1}\dot{\overline{\omega}}_{i-1} + \begin{cases} {}^{i}_{i-1}[R]\,{}^{i-1}\overline{\omega}_{i-1} \times \dot{q}_i \cdot {}^{i}\overline{k}_i + \ddot{q}_i\,{}^{i}\overline{k}_i & \text{,if i = R} \\ 0 & \text{,if i = T} \end{cases} \tag{6} $$

$$ \begin{aligned} {}^{i}\dot{\overline{v}}_i = {}^{i}_{i-1}[R]\left\{{}^{i-1}\dot{\overline{v}}_{i-1} + {}^{i-1}\dot{\overline{\omega}}_{i-1} \times {}^{i-1}\overline{r}_i + {}^{i-1}\overline{\omega}_{i-1} \times {}^{i-1}\overline{\omega}_{i-1} \times {}^{i-1}\overline{r}_i\right\} + \\ + \begin{cases} 0 & \text{,if i = R} \\ 2 \cdot {}^{i}\overline{\omega}_i \times \dot{q}_i \cdot {}^{i}\overline{k}_i + \ddot{q}_i\,{}^{i}\overline{k}_i & \text{,if i = T} \end{cases} \end{aligned} \tag{7} $$

Where:

$$ {}^{i}\overline{k}_i = \begin{cases} [1 \;\; 0 \;\; 0]^T & \text{if motion axis = X;} \\ [0 \;\; 1 \;\; 0]^T & \text{if motion axis = Y;} \\ [0 \;\; 0 \;\; 1]^T & \text{if motion axis = Z.} \end{cases} \tag{8} $$

In equations (4)-(7) the parameters $\dot{q}_i$ and $\ddot{q}_i$ represent the 1st and 2nd time derivative of the *i* joint displacement, ${}^{i}_{i-1}[R]$ the transposed rotation matrix of joint *i* relative to joint *i-1* and ${}^{i-1}\overline{r}_i$ the position column matrix of joint *i*.

Using the D-H equations and applying equations (4)-(8) we obtained the gripper velocities and accelerations relative to the last joint coordinate system:

$$ {}^{P}\overline{\omega}_P = \begin{pmatrix} -\dot{q}_1 \cdot cq_4 \\ \dot{q}_1 \cdot sq_4 \\ \dot{q}_4 \end{pmatrix}; \; {}^{P}\overline{v}_P = \begin{pmatrix} -\dot{q}_2 \cdot cq_4 + (l_4 + l_6 - q_3) \cdot \dot{q}_1 \cdot sq_4 \\ \dot{q}_2 \cdot sq_4 + (l_4 + l_6 - q_3) \cdot \dot{q}_1 \cdot cq_4 \\ l_3 \cdot \dot{q}_1 + \dot{q}_3 \end{pmatrix} \tag{9,10} $$

$$ {}^{P}\dot{\overline{\omega}}_P = \begin{pmatrix} \dot{q}_1 \cdot \dot{q}_4 \cdot sq_4 - \ddot{q}_1 \cdot cq_4 \\ \dot{q}_1 \cdot \dot{q}_4 \cdot cq_4 + \ddot{q}_1 \cdot sq_4 \\ \ddot{q}_4 \end{pmatrix} \tag{11} $$

$$ {}^{P}\dot{\overline{v}}_P = \begin{pmatrix} (l_4 + l_6 - q_3) \cdot \ddot{q}_1 \cdot sq_4 - \ddot{q}_2 \cdot cq_4 + l_3 \cdot \dot{q}_1{}^2 \cdot sq_4 + 2 \cdot \dot{q}_1 \cdot \dot{q}_3 \cdot sq_4 - g \cdot cq_4 \\ (l_4 + l_6 - q_3) \cdot \ddot{q}_1 \cdot cq_4 + \ddot{q}_2 \cdot sq_4 + l_3 \cdot \dot{q}_1{}^2 \cdot cq_4 + 2 \cdot \dot{q}_1 \cdot \dot{q}_3 \cdot cq_4 + g \cdot sq_4 \\ l_3 \cdot \ddot{q}_1 + \ddot{q}_3 - (l_4 + l_6 - q_3) \cdot q_1{}^2 \end{pmatrix} \tag{12} $$

The linear velocity and acceleration of the gripper relative to the fixed coordinate system is obtained by multiplying the rotation matrix of the gripper with the ${}^{P}\overline{v}_P$ and ${}^{P}\dot{\overline{v}}_P$ matrices. Therefore, ${}^{0}\overline{v}_P$ and ${}^{0}\dot{\overline{v}}_P$ matrices have the following form:

$$ {}^{0}\overline{v}_P = \begin{pmatrix} -\dot{q}_1 \cdot (A \cdot cq_1 + l_3 \cdot sq_1) - \dot{q}_3 \cdot sq_1 \\ -\dot{q}_1 \cdot (A \cdot sq_1 + l_3 \cdot cq_1) + \dot{q}_3 \cdot cq_1 \\ \dot{q}_2 \end{pmatrix} \tag{13} $$

$$ {}^{0}\dot{\bar{v}}_P = \begin{pmatrix} -\ddot{q}_1 \cdot (A \cdot cq_1 + l_3 \cdot sq_1) - \ddot{q}_3 \cdot sq_1 + \dot{q}_1^{\,2} \cdot (A \cdot sq_1 - l_3 \cdot cq_1) - 2 \cdot \dot{q}_1 \cdot \dot{q}_3 \cdot cq_1 \\ -\ddot{q}_1 \cdot (A \cdot sq_1 - l_3 \cdot cq_1) + \ddot{q}_3 \cdot cq_1 - \dot{q}_1^{\,2} \cdot (A \cdot cq_1 + l_3 \cdot sq_1) - 2 \cdot \dot{q}_1 \cdot \dot{q}_3 \cdot sq_1 \\ \ddot{q}_2 + g \end{pmatrix} \quad (14) $$

Because the last joint is rotating around its Z axis, parameter q_4 is not influencing the velocity and the acceleration of the gripper relative to the fixed coordinate system.

2.2 Inverse kinematics

For a desired position, velocity and acceleration in order to find the joints angular or linear displacements and their 1st and 2nd time derivative we determined the inverse kinematics equations of the robot. The equations look as follow:

$$ sq_1 = \frac{x}{(q_3 - l_6 - l_4)} - \frac{x \cdot l_3^{\,2} - y \cdot l_3 \cdot (q_3 - l_6 - l_4)}{(q_3 - l_6 - l_4) \cdot \left((q_3 - l_6 - l_4)^2 + l_3^{\,2}\right)} $$

$$ cq_1 = \frac{x \cdot l_3 - y \cdot (q_3 - l_6 - l_4)}{(q_3 - l_6 - l_4)^2 + l_3^{\,2}} $$

$$ q_1 = a\tan 2(sq_1, cq_1) \quad (15) $$

$$ q_2 = z + l_5 - l_0 - l_1 \quad (16) $$

$$ q_3 = -\sqrt{x^2 + y^2 - l_3^{\,2}} + l_6 + l_4 \quad (17) $$

Fig. 2 presents the variation of q_1, q_2 and q_3 displacements function of a desired trajectory for a fixed *y* coordinate. The *x* and *z* coordinates are increased simultaneously from 50 [mm] to 250 [mm].

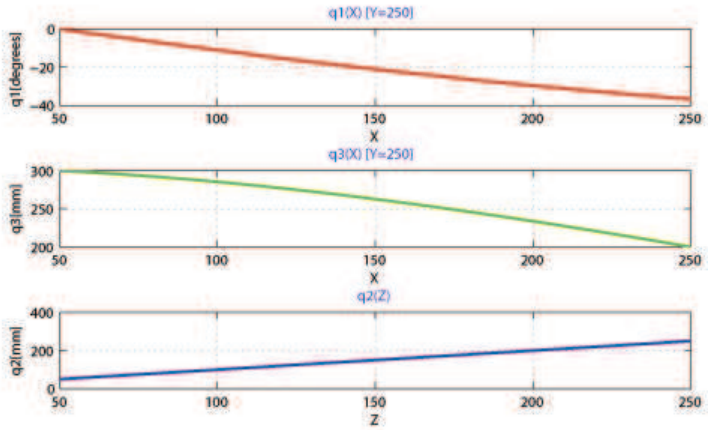

Figure 2. Inverse kinematics – displacements variation function of *{x,y,z}*

Equation (13) was used to obtain the 1st derivative of the joints displacements considering a desired linear velocity for the gripper. The inverse kinematics equations for velocities will be as follow:

$$\dot{q}_1 = \frac{v_x \cdot cq_1 + v_y \cdot sq_1}{l_4 + l_6 - q_3} \tag{18}$$

$$\dot{q}_2 = v_z \tag{19}$$

$$\dot{q}_3 = \left(v_y \cdot cq_1 - v_x \cdot sq_1\right) + \frac{l_3}{l_4 + l_6 - q_3} \cdot \left(v_y \cdot sq_1 + v_x \cdot cq_1\right) \tag{20}$$

To obtain the 2nd derivative of the joins displacements or the joints accelerations for a desired gripper acceleration we used equations (14). The resulting equations are:

$$\ddot{q}_1 = -\frac{l_3 \cdot \dot{q}_1^{\,2} + 2 \cdot \dot{q}_1 \cdot \dot{q}_3 + a_x \cdot cq_1 + a_y \cdot sq_1}{l_4 + l_6 - q_3} \tag{21}$$

$$\ddot{q}_2 = a_z - g \tag{22}$$

$$\ddot{q}_3 = \left(a_y \cdot cq_1 - a_x \cdot sq_1\right) + \left(l_4 + l_6 - q_3\right) \cdot \dot{q}_1^{\,2} + \frac{l_3 \cdot \left(a_y \cdot sq_1 + a_x \cdot cq_1 + \dot{q}_1^{\,2} + 2 \cdot \dot{q}_1 \cdot \dot{q}_3\right)}{l_4 + l_6 - q_3} \tag{23}$$

2.3 Robot mechanics

In order to build the mechanical structure of the robot we initially designed and 3D modelled the robot components. The mechanical structure chosen for this project is a 4 degrees-of-freedom (DOF) industrial robot having a rotation joint in the robot base represented by a gear unit, two translation joints using ball screw-nut mechanisms and a rotation joint represented by a DC geared motor, directly linked with a parallel gripper.

The mechanical structure was designed taking into consideration one of the most important factors which are influencing the robot dynamics: the friction. In order to reduce the friction and increase the robot performance we used very accurate mechanisms like: a harmonic drive for the robot base, linear ball bearings and ball screw-nut mechanisms for the translation joints. The most important physical properties of these mechanisms are:

- zero or very low backlash;
- high accuracy;
- high toque capacities;
- high efficiency.

The harmonic drive has a reduction ratio of 100:1 and the ball screw mechanisms have the lead of 5 [mm]. The 4th joint uses a DC motor and a gripper from a refurbished robot.

Table 2 presents some mechanical properties that we established for the robot joints, based on the virtual model initially designed and the mechanical properties of the actuators, gear units and translation mechanisms.

Joint	Type	Stroke [degrees], [mm]	Max. Speed [0/s],[mm/s]
1	Rotation	180^0	~18 [0/s]
2	Translation	310 [mm]	20 [mm/s]
3	Translation	350 [mm]	20 [mm/s]
4	Rotation	360^0	180 [0/s]

Table 2. The robot joints mechanical properties

Fig. 3 presents the 3D model of the robot arm and the real robot arm (first three joints).

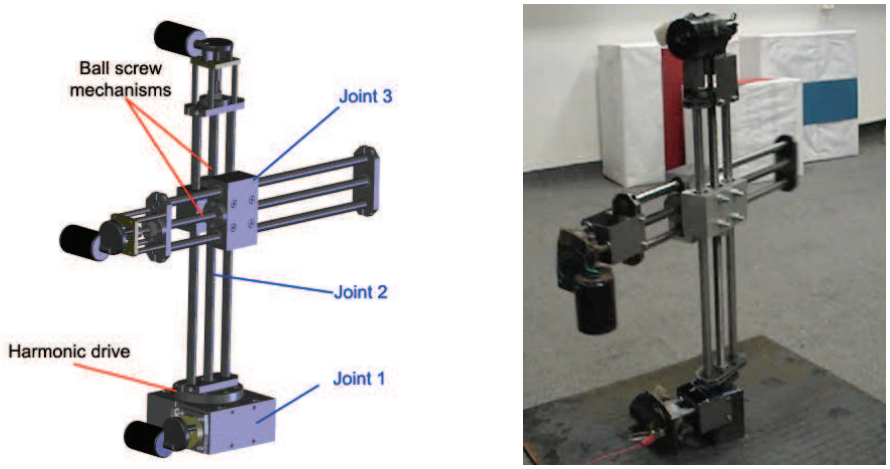

Figure 3. The virtual and the real robot arm

3. The Robot Controller

By definition, an industrial robot must be a re-programmable multi-functional manipulator designed to perform a variety of tasks (Dai Gil, 2005). The component which makes a robot reprogrammable and multi-functional is the robot controller. The robot controller, also known as the "robot brain", represents the component which gives functionality and autonomy to a certain robotic system. Basically, the robot controller is formed of two main parts: controller hardware and controller software. The complexity and the configuration of a controller differ from robot to robot. For simple tasks and robot structures the controllers may have simple configurations.

Nowadays, the production of industrial robot controllers is taken over by the robots manufacturers mostly because the industrial robots are used in mass production applications where high-level controllers are needed. For simple robotic systems, where simple mechanical structures are used, the robot controller can be made using cheap and small electronic devices. When designing controllers for research purposes it is recommended to focus on open architecture and/or modular design. Using open architecture and/or modular design lead to multi-purpose controllers.

In order to develop robot control algorithms, the robot controller must ensure a high degree of efficiency, modularity and scalability (Rusu et al., 2006; Lazea et al., 2006).

In our project the robot controller has an open architecture design both from the software and hardware point of view. The controller hardware consists of: a re-programmable processing unit board, a re-programmable microcontroller board, a 16-to-1 multiplexer board and four programmable motor controller boards. The controller software consists of:

- TCP/IP client-server application for processing unit;
- Serial communication application for processing unit;
- CCSC (*Custom Computer Services C*) application for PIC microcontroller.

3.1 The controller architecture

The robot controller was designed taking into consideration our previous researches made within the open architecture robot design area, especially the results that we obtained in the *"ZeeRO"* mobile robot project (Rusu et al., 2006). Therefore, the robot controller should be:

- open and modular in order to provide sufficient programmable Input/Output ports;
- customizable in order to permit both software and hardware upgrades;
- cheap and small in order to be affordable for any research applications.

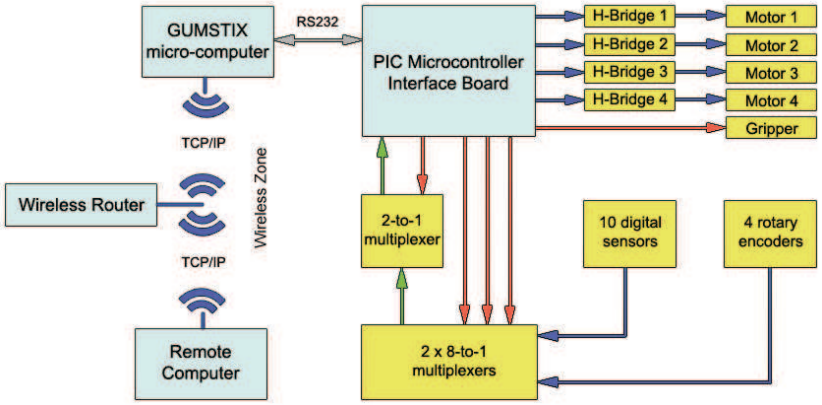

Figure 4. The robot controller architecture

The processing unit of the controller has two processing layers:

- high level processing layer, represented by the Gumstix micro-computer;
- low level processing layer, represented by the PIC microcontroller and the multiplexer board.

The communication between the two processing layers is handled by a RS232 connection. In order to be able to read as many input ports as possible we used a multiplexer board. The multiplexer board contains two 8-to-1 interconnected digital multiplexers and one 2-to-1 master multiplexer. The board is able to handle 16 input ports and is controlled by the PIC microcontroller. In our application, the multiplexer board handles 10 digital sensors (stroke end limit switches) and 4 rotary encoders.

The power circuit (the motors) is controlled by the PIC microcontroller via four H-Bridges. The H-Bridges are *Devantech MD03* type and they are able to control up to 100W (current – 20, voltage 50V) DC motors. The connection between the H-Bridges and the PIC microcontroller is made using two I/O lines, communication being handled by a I2C protocol (built in PIC microcontroller). The PIC microcontroller is able to control up to 12 MD03 H-Bridges.

The high level processing layer is represented by a Gumstix miniature computer (Fig. 5). The Gumstix micro-computer is a small device powered by a PXA255 Intel XScale processor, running at 400 MHz. The most important characteristics of this micro-computer are its modularity and size.

The Gumstix micro-computer is powered by Linux operating system which can be considered as an advantage, especially in developing open source applications.

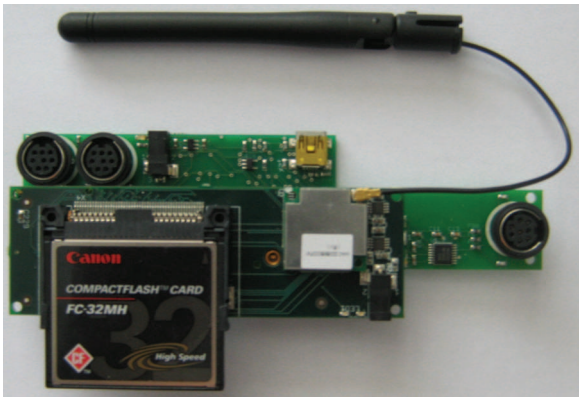

Figure 4. The GUMSTIX micro-computer

One of the serial ports is used to make the connection with the PIC microcontroller. The other two serial ports can be used to connect other devices. In our case, we reserved the two serial ports to connect two serial cameras (*CmuCam3*) for future researches.
The wireless expansion board is used to communicate with remote computers using the TCP-IP communication protocol.
The low level processing layer is represented by the PIC microcontroller board together with the multiplexer board and the H-bridges. From the hardware point of view, the low level processing layer is the interface between the controller "brain" and the robot sensors and motors.
The PIC microcontroller board (Fig. 5) contains a PIC16F876 microcontroller running at 20MHz, 15 I/O configurable pins and one serial port connected to the Gumstix micro-computer. One of the output pins of the microcontroller is reserved to control the gripper state (on/off state).

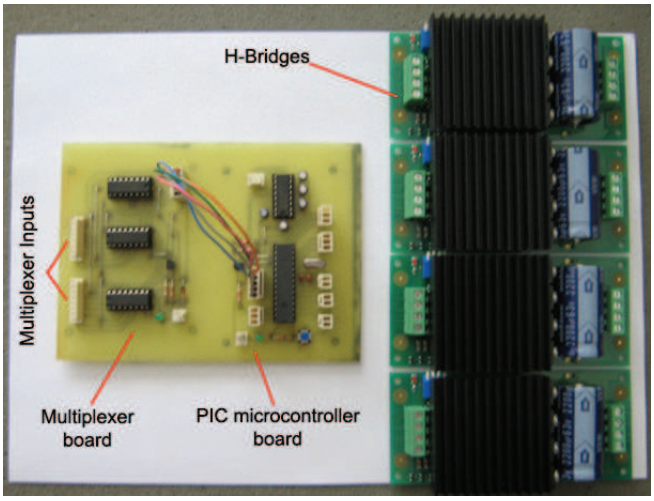

Figure 5. The low-level processing layer hardware

3.2 The controller software

The robot controller contains two main software packages and one additional application. The main packages are the Gumstix micro-computer programs and the PIC microcontroller program. The additional application is the remote client software which can be used by any remote computer from the same local area network with the Gumstix micro-computer.

The Gumstix micro-computer software contains two main routines:

- the TCP-IP routine, for communication with remote clients via wireless TCP-IP protocol;
- the RS232 routine, for communication with the PIC microcontroller via a serial port.

The application is written in ANSI C under Linux using standard *POSIX pthread* functions to provide support for multithreading, as in [5]. Because the Gumstix microprocessor has 32-bit *RISC* architecture, the programs need to be compiled using Advanced RISC Machine (*ARM*) GNU C Compiler (*ARM-GCC*).

Because the Gumstix memory is limited, the Linux libraries are not installed in the Gumstix flash memory. In order to compile the programs, the ARM Linux kernel was compiled on a stand-alone computer then used for C programs compilations. Also, the programs include all the libraries needed by the functions used. The disadvantage of this method is the executable size which is much bigger that an executable which uses precompiled libraries.

The RS232 routine receives the data sent by the PIC microcontroller through the serial port. The data received is in string format (array of characters) and contains information about the current sensors and motors state. For example, if the PIC microcontroller turns off the robot base motor, the RS232 routine receives "M1{0} – off". The number between braces represents the rotation sense of the motor, "0" for clockwise and "1" for counter clockwise. The RS232 routine processes the message received and sends a command back to the PIC microcontroller if needed. In some cases the PIC microcontroller acts autonomous, even if the message is sent to the Gumstix micro-computer. For example, if one of the stoke limit switches from the same axis with a running motor turns "*On*", the PIC microcontroller turns off the motor in order to avoid collisions between the mechanical parts of that axis. After that, the microcontroller sends the proper message to the Gumstix RS232 routine.

The wireless TCP-IP routine handles the connections from remote computers. If a connection is open, the routine will automatically send data to the remote computer.

The TCP-IP connection is made through a wireless access point which allows connections from multiple remote computers, both from local area network and external network.

We used the TCP-IP connection for monitoring purposes and for reading the data used in the simulator. In some cases, the program code can be easily modified and recompiled in order to give full control of the robot to the remote client.

The PIC microcontroller program is written in CCS C programming language which provides the same syntax as ANSI C. The program contains two main routines:

- the sensors reading routine;
- the motors control routine.

The sensors reading routine controls the multiplexer board through 4 output pins and 1 input pin. One of the output pins controls the 2-to-1 multiplexer (the master multiplexer). Function of the pin value (logic 1 or 0), the master multiplexer reads the output pin of the proper 8-to-1 multiplexer (slave multiplexer). The other 3 output pins of the PIC microcontroller are used to control the inputs of the slave multiplexers. The state of the

input pins is read continuously. Function of the input pins state, the routine either sends a message to the Gumstix micro-computer or stops the motors if needed.
The motors control routine controls the four Devantech H-Bridges using the I2C protocol. Each H-Bridge can open a serial communication line using a data register. The data register can be configured using the 4-mode switches placed on each H-Bridge. The PIC microcontroller selects which H-Bridge to control with the help of the built-in *i2c_start*, *i2c_write* and *i2c_read* functions.

4. The Simulation Software Application

According to Zaratti, a simulation tool for industrial robots must accomplish the following requirements (Zaratti et al., 2006):

- Flexibility: the tool should allow the simulation of various types of robots, sensors and actuators. The working space of the robot(s) should allow easy modeling;
- Physical realism: to obtain good results, the interaction between the robot and the simulated working space should be modeled by using physical law and rigid body dynamics laws;
- Visual realism: the entire system should be as realistic as possible in order to obtain a correct representation of the data collected from the sensors;
- Efficiency: the simulation must be made in the most efficient way, preferable at a maximum refresh rate;
- Modularity: the simulator should allow the modification of the working space and the robot components;
- Effective control: the simulator should be able to interact with the robot controller software packages.

The requirements above can be accomplished when talking about well-known robot types from the market. Even in this case the effective control is very hard to be made and the programmer(s) should be familiar with a large number of robots, robot controllers, sensors and actuators. More than that, in case of a custom robot the simulator must be flexible enough in order to simulate custom components. Therefore, we proposed that in case of a custom industrial robot the simulator should be designed and built from scratch. The idea of building a simulator from scratch may be discouraging but analyzing the variety of methods for creating a 3D simulator it becomes an encouraging idea. Since our simulation system contains a single industrial robot, the first requirement would be only partially accomplished.
In order to be able to accomplish all the requirements we decided to create the simulation software by modeling with open-source programming languages. Therefore, we used Linux operating system, Qt4 programming language to design the interface and OpenGL to create the 3D simulation scene.

4.1 The simulator main window

The main window of the simulator (Fig. 6) was designed and built in Qt4 programming language (Blanchette & Summerfield, 2006). Qt4 is a cross-platform application framework used desktop or embedded development. We choose this platform because of the advantages provided:

- free and open source for Linux operating system;

- rich set of application building blocks;
- easy to use and learn;
- implemented in C++ and provides support for C++ development;
- has implemented support for OpenGL.

Figure 6. The main window

The main window program consists of four primary classes, five secondary classes and two experimental classes (Fig. 6).

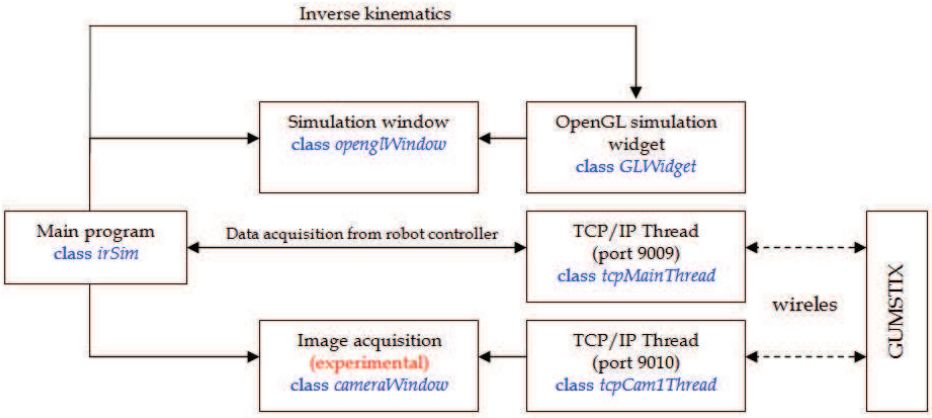

Figure 7. Main program classes and data flow

The data exchange between the program classes is made with the help of Qt *signals* and *slots*. The *signals* and the *slots* are predefined or user defined functions interconnected by the Qt *connect* function. When a *signal* is emitted by a function or a class, the corresponding *slot*

function loaded. From the programming point of view this represents an advantage and leads to an easy exchange of data between the classes.
The right side of the main window consists of information read by the *tcpMainThread* class from the Gumstix micro-computer. The *tcpMainThread* acts like a TCP/IP client class. The TCP/IP server resides on the robot controller and sends the sensors and motors status on connection start and every time a sensor or a motor changes its state (if a client is connected).
Based on the joints position read by the multiplexer and PIC microcontroller, the "Current Position" group indicates the current gripper position relative to the robot base. The gripper position is calculated with the forward kinematics equations implemented as variables in the Gumstix program and sent to the client main window through TCP/IP.
The "Sensor Status" group contains five pairs of edit boxes indicating the stroke limit sensors status for each of the four joints and for the robot gripper (*open* and *closed* state).
Function of the TCP/IP connection status, the "WiFi Connection" group can indicate three types of status messages: "Connected", "Not connected" or a connection error message (Fig. 8).

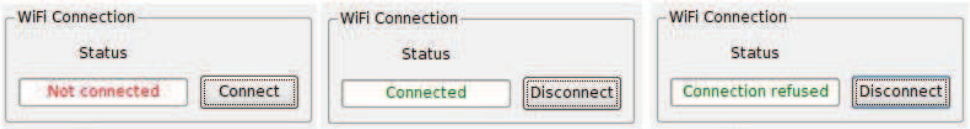

Figure 8. WiFi TCP/IP connection status

The simulation program is able to run in two modes: real mode and simulation mode. In real mode, the program read the information from the robot controller via TCP/IP and any change of data from the main window is changing the position of the virtual model from the simulation window. In simulation mode, the "*Set Position*" and "*Motion Axis*" groups are disabled (Fig. 9), the simulation window is automatically opened and the virtual model of the robot moves from initial position to the position given by the "*Set Position*" parameters.

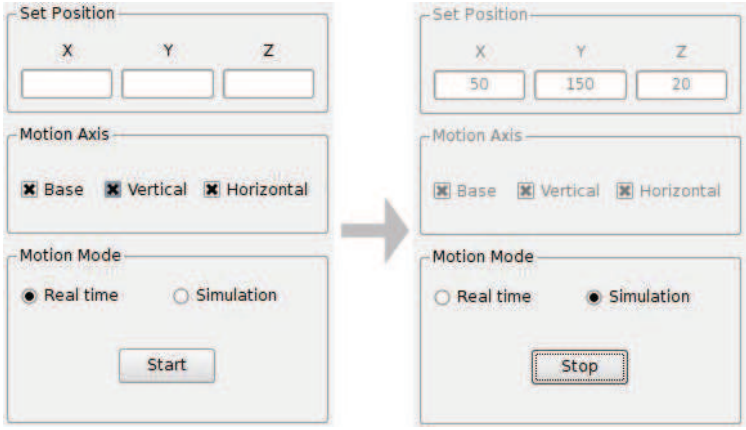

Figure 9. The simulation modes

4.2 The simulation window

The simulation window is *OpenGL* based and contains a primary class named *openglWindow* which is a child class of the main program class *irSim*. Therefore, the elements from this window can be easily controlled by the main window. The *openglWindow* is linked with a secondary class (child class) named *glWidget* which handles the graphical simulation process with the help of the *OpenGL* functions.

OpenGL is the premier environment for developing portable, interactive 2D and 3D graphics applications. Since its introduction in 1992, OpenGL has become the industry's most widely used and supported 2D and 3D graphics application programming interface (API), bringing thousands of applications to a wide variety of computer platforms. OpenGL fosters innovation and speeds application development by incorporating a broad set of rendering, texture mapping, special effects, and other powerful visualization functions. Developers can leverage the power of OpenGL across all popular desktop and workstation platforms, ensuring wide application deployment.

OpenGL routines simplify the development of graphics software—from rendering a simple geometric point, line, or filled polygon to the creation of the most complex lighted and texture-mapped *NURBS* curved surface. OpenGL gives software developers access to geometric and image primitives, display lists, modeling transformations, lighting and texturing, anti-aliasing, blending, and many other features.

Our simulation window reproduces the robot structure at a small scale (Fig. 10).

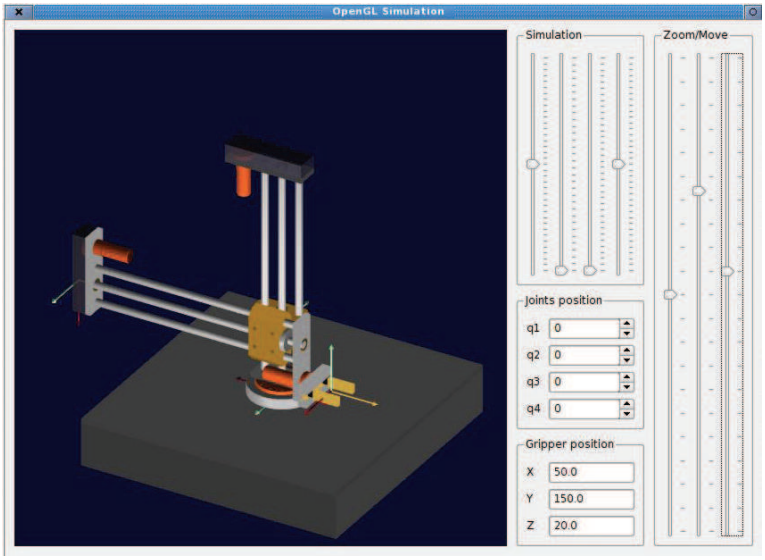

Figure 10. The simulation window

The simulation window is split into two main parts: the graphical simulation widget and the simulation control elements. Every component of the virtual model is represented by an OpenGL list and the position and orientation of the components is based on then values set by the simulation control elements. By using OpenGL lists the objects are precompiled and the simulation process is performed faster. Each list contains sets of functions which generate the objects from OpenGL primitives: points, lines, quads, disks, etc. The process of

building the lists is difficult and requires the exact knowledge of the object dimensions. Anyway, once created, the objects can be easily modified by changing their dimensions and visual properties.

Our robot simulation process can be performed automatically or manually. The simulation is performed automatically when the main program is set on real motion mode and is connected to the robot controller. Also, the simulation performs automatically when the OpenGL window is opened by the main window simulation mode. To simulate the robot manually the main program should not be connected to the robot controller. In this case, the virtual model of the robot can be moved in any possible position by changing the values of the joints positions. The joints positions can be changed both from the "*Simulation*" group slide bars and from the "*Joints position*" group spin boxes. At any change of the joints position the gripper coordinates are displayed in the "*Gripper position*" group edit boxes. The gripper position is calculated automatically using the forward kinematics equations, no matter if the simulation process is performed manually or automatically.

When the gripper coordinates are received from the main window the joints positions are calculated using the inverse kinematics equations and the virtual model of the robot is positioned accordingly (Fig. 11).

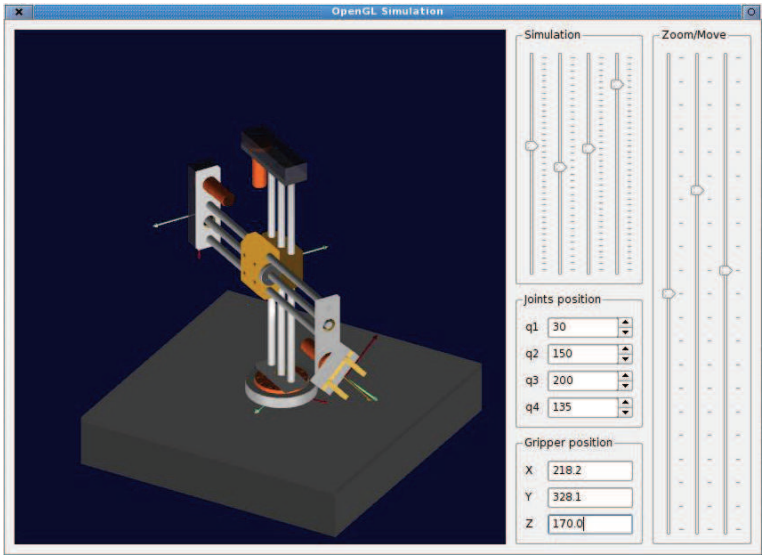

Figure 11. Robot virtual model in a particular position

5. Conclusions

In order to create a simulator for custom industrial robots, it is very important to know the forward and inverse kinematics equations of the robot structure, the controller output data and the limitations of the robot mechanical components. In this paper we presented the steps for building a simulation program for a custom industrial robot. The first step was the robot modeling where we obtained the forward and inverse kinematics equations used as motion laws both for the simulated and for the real robot. The second step was the design of

an open architecture robot controller able to communicate with TCP/IP clients. The last step was the design of an open source flexible simulation program, able to reproduce the real robot as realistic as possible. By designing and building this simulation system we implemented a free and easy to use simulation method which can be easily implemented in other research areas. The use of OpenGL features lead to a flexible simulation program which can be easily modified function of the system needs. With regards to the information presented in this chapter we conclude that the simulation system of the robot is a great step forward for us in our research within the field of industrial robot simulation.

6. References

Blanchette, J. & Summerfield, M. (2006). *C++ GUI Programming with QT4*, Prentice Hall Publishing, ISBN 0-13-187249-4.

Dai Gil, L., (2005). *Axiomatic Design and Fabrication of Composite Structures*, Oxford University Press, ISBN 0195278777, pp. 513.

Gerkey, B.P., Vaughan, R.T. & Howard, A. (2003). The Player/Stage Project: Tools for Multi-robot and Distributed Sensors Systems, *Proceedings of the International Conference on Advanced Robotics (ICAR)*, pp. 317-323, ISBN 972-96889-9-0, June 30 - July 3, 2003, Coimbra, Portugal

Laue, T., Spiess, K. & Rofer, T. (2005). SimRobot - A General Physical Robot Simulator and its Application in RoboCup, *Proceedings of RoboCup Symposium*, Universitat Bremen

Lazea, Gh., Rusu, R.B., Robotin, R., Sime, R. (2006). The ZeeRO mobile robot - A modular architecture, *RAAD 2006 International Workshop on Robotics*, ISBN 963-7154-48-5, Balaton, Hungary

Negran, I., Vuscan, I. & Haiduc, N. (1997). *Robotics – Kinematics and Dynamic Modelling*, Didactic and Pedagogic Publishing, ISBN 973-30-5309-8, Bucharest, Romania

Rohrmeier, M. (2000). Web Based Robot Simulation using VRML, *Simulation Conference Proceedings*, ISBN: 0-7803-6579-8, vol. 2, pp. 1525-1528, Orlando, U.S.A

Rusu, R.B., Lazea, Gh., Robotin R. & Marcu C. (2006). Towards Open Architectures for Mobile Robots: ZeeRO, *IEEE-TTTC International Conference on Automation, Quality and Testing, Robotics AQTR 2006 (THETA 15)*, pp. 260-265, 25-28 May, 2006, Cluj-Napoca, Romania

Wang, J., Lewis, M., Gennari, J. (2003). A game engine based simulation of the NIST Urban Search & Rescue arenas, *Proceedings of the 2003 Winter Simulation Conference*

Zagal, J. C. & del Solar, J. R. (2004). UCHILSIM: A Dynamically and Visually Realistic Simulator for the RoboCup Four Legged League. *In RoboCup Symposium 2004*, Robot Soccer World Cup VIII.

Zaratti, M., Fratarcangeli, M. & Iocchi, L. (2006). A 3D Simulator of Multiple Legged Robots based on USARSim, *Proceedings of RoboCup Syposium*, Bremen, Germany

Robot Manipulators

Edited by Marco Ceccarelli

ISBN 978-953-7619-06-0
Hard cover, 546 pages
Publisher InTech
Published online 01, September, 2008
Published in print edition September, 2008

In this book we have grouped contributions in 28 chapters from several authors all around the world on the several aspects and challenges of research and applications of robots with the aim to show the recent advances and problems that still need to be considered for future improvements of robot success in worldwide frames. Each chapter addresses a specific area of modeling, design, and application of robots but with an eye to give an integrated view of what make a robot a unique modern system for many different uses and future potential applications. Main attention has been focused on design issues as thought challenging for improving capabilities and further possibilities of robots for new and old applications, as seen from today technologies and research programs. Thus, great attention has been addressed to control aspects that are strongly evolving also as function of the improvements in robot modeling, sensors, servo-power systems, and informatics. But even other aspects are considered as of fundamental challenge both in design and use of robots with improved performance and capabilities, like for example kinematic design, dynamics, vision integration.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821