MATLAB-based Robot Control Design Environment for Research and Education

L. Žlajpah, B. Nemec, D. Omrčen, "Jožef Stefan" Institute, Ljubljana, Slovenia

SNE Simulation Notes Europe SNE 20(3-4), 2010, 55-66, doi: 10.11128/sne.20.en.09995

Research in the field of robotics is tightly connected to simulation tools for many reasons. On one side, simulation supports the development of new advanced control algorithms and on the other side it is always not feasible to build a whole robot system to test some algorithms or it is not safe to perform tests on a real system (at least in the first design stages). In the paper we present an integrated environment for the design and testing of advanced robot control schemes, including visual tracking, force feedback on a single robot or in multi-robot applications. The kernel of our simulation environment is MATLAB/Simulink. The main capabilities are: the simulation of the kinematics and dynamics of manipulators, the integration of different sensor systems like vision and force sensors, scenarios for complex robot tasks, the visualization of robots and their environment and the integration of real robots in the simulation loop. The advantage of our system is the simplicity, which allows easy integration of different robots, sensors and other devices. Some of these can be easier simulated by using other tools. Hence, other simulation tools can be used for the simulation of different parts of the system and then these subsystems are integrated in out simulation environment. The other important feature is easy final testing of developed control algorithms. Namely, for final testing of the control algorithms the models in the simulation scheme are just replaced by interface blocks for real system and the user does not need to consider implementation details. Finally, to show the efficiency and usability of our control design environment we outline some typical experimental examples using our robots. We explain some typical control design procedures from the "pure" simulation to the testing of algorithms on real robots.

Introduction

The ways and methods in robotics research and development have always been influenced by the tools used. This is especially true when one considers the profound impact of recent technologies on robotics, especially the development of computers which become indispensable when designing the complex systems like robots. Not many years ago, computing cost was still a significant factor to consider when deriving algorithms and new modeling techniques [1, 2, 3]. Nowadays, distributed computing, network technology and the computing power developed by commercial equipment open new possibilities for doing systems design and implementation. However, in spite of all that the creativity of a human designer cannot be left out in the design process. The best solution seems to be to provide the designer with proper tools which significantly increase his efficiency. Among them, the simulation has been recognized as an important tool in designing the new products, investigating their performances and also in designing applications of these products. For complex systems as robots the simulation tools can certainly enhance the design, development, and even the operation of the robotic systems. Augmenting the simulation with visualization tools and interfaces, one can simulate the operation of the robotic systems in a very realistic way.

Currently, many different simulation tools for robotic systems are available. They differ from each other depending on which aspect of the robot research they support, how open they are or on which platforms they work. However, many tools are not always fulfilling all the requirements of the research and teaching activities in robotic laboratories like reconfigurability, openness and ease of use, etc.

Reconfigurability and openness are features already recognized by many as essential in the development of advanced robot control algorithms [4, 5, 6]. Not only is it important to have easy access to the system at all levels (e.g. from high-level supervisory control all the way down to fast servo loops at the lowest level), but it is a necessity to have open control architectures where software modules can be modified and exteroceptive sensors like force/torque sensors and vision systems can be easily integrated. Reconfigurability should also be reflected when more fundamental changes to the controller architecture are required, in the necessity of quickly being able to make modifications in the original design and verify the effect of these modifications on the system. In other words, the user should be able to quickly modify the structure of the control without having to alter the simulation system itself.

In the last decade the software has become more and more easy to use. This is still one of the main major issues when selecting a software tool. First of all, the tools are used by many users in a laboratory and not all of them have the same expertise. To boost the knowledge exchange, it is of benefit that they work with the same tools. Next, testing of different control algorithms on real robotic systems is in general not very user friendly: the algorithms usually have to be rewritten for the real-time execution and the different implementation details have to be considered [4, 7]. This forces the user to devote a large part of the design time to topics not connected with the main issues of the control design, especially when he is not interested in software implementation issues. The ease of use becomes even more important when students are working with robots. In most cases they work in a laboratory for a shorter period, they are focused on their projects and they could become frustrated if they have to learn a lot of things not directly connected to their tasks. Finally, in research laboratories different robot systems are used equipped with more or less open proprietary hardware and software architecture. Therefore, it is much desired that the control design environment is unified, i.e. the same tools can be used for all robot systems.

The simulation tools for robotic systems can be divided into two major groups: tools based on general simulation systems and special tools for robot systems [8]. Tools based on general simulation systems are usually represented as special modules, libraries or user interfaces which simplify the building of robot systems and environments within these general simulation systems (e.g. SolidWorks [9]). On the other hand, special simulation tools for robots cover one or more tasks in robotics like off-line programming and design of robot work cells (e.g. Robcad [10]) or kinematic and dynamic analysis [11, 12]. They can be specialized for special types of robots like mobile robots, underwater robots, parallel mechanisms, or they are assigned to predefined robot family. Depending on the particular application different structural attributes and functional parameters have to be modelled.

For the use in research and teaching laboratories, robot simulation tools focused on the motion of the robotic manipulator in different environments are important, especially those for the design of robot control systems [13, 11, 12, 14]. Recently, Microsoft Robotics Studio (MSRS) [13] has been launched with a general aim to unify robot programming for hobbyist, academic and commercial developers and to create robot applications for a variety of hardware platforms. The system enables both remotely connected and robot-based scenarios using .NET and XML protocols. Simulation engine enables real-time physics simulation and interaction between simulated entities. Each part of control loop can be substituted with the real or simulated hardware. Although the system is still under the development, it is not easy to add new entity, for example a new robot or a new sensor. One of the major drawbacks seems to be the low data throughput rate, which does not allow the realization of complex control laws at high sampling frequency. Therefore, it is not clear yet if MSRS is appropriate for research robotics, especially for complex systems. Real time requirements are better solved in another programming/simulation framework, MCA2 [15]. MCA is a modular, network transparent and realtime capable C/C++ framework for controlling robots and other hardware. The main platform is Linux/RTLinux, but support for Win32 and MCA OS/X also exists. However, it is still a complex system and therefore less appropriate for education and students projects.

MATLAB is definitely one of the most used platforms for the modelling and simulation of various kind of systems and it is not surprising that it has been used intensively for the simulation of robot systems. “The Robotics Toolbox” [11] provides many functions that are required in robotics and addresses areas such as kinematics, dynamics, and trajectory generation. The Toolbox is useful for simulation as well as for analyzing the results from experiments with real robots, and can be a powerful tool for education. However, it is not very good for the simulation in Simulink and for the hardware-in-the-loop simulation “SimMechanics Toolbox” [12] extends Simulink with the tools for modelling and simulating mechanical systems. With SimMechanics, one can model and simulate mechanical systems with a suite of tools to specify bodies and their mass properties, their possible motions, kinematic constraints, and coordinate systems and to initiate and measure body motions.

In the following, we present our approach to the integrated environment for the design and testing of robot control systems. Our framework is not intended as an alternative to the MSRS or MCA2. It is not as complex as MSRS and it does not possess physic simulation capabilities. On the other hand, real time capabilities cannot be compared to the MCA2.

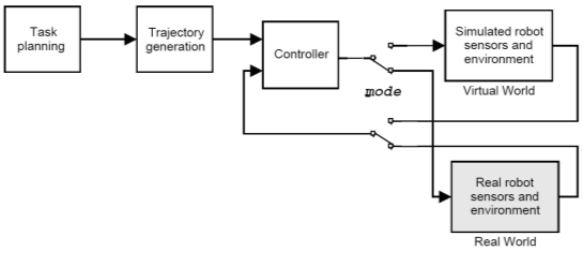

Figure 1. A block diagram of the integrated environment.

The advantage of our system is the simplicity, which allows easy integration of new entities and is also very appropriate for the education and research robotics. First we present our concept of the use of simulation tools in the control design for research and education. Next, the experimental setup in our laboratory is described. Finally, to show the efficiency and usability of our control design environment we outline some typical experimental examples using our service robot.

1 The concept

The importance of simulation tools in the development of robot control systems has been recognized by our team very early. We have been using different simulation tools for over 20 years and many of them have been developed in our laboratory. In the last decade we have been using for the control design MATLAB/Simulink based integrated environment based on Planar Manipulators Toolbox for dynamic simulation of redundant planar manipulators [7]. It enables the use of different sensors in the control loop and also the real-time implementation of the controller and hardware-in-the-loop simulation. Figure 1 shows the general simulation scheme in this environment.

A crucial feature inherited in this scheme is indicated by the mode switches. Namely, the user can easy switch between using model or a real system in the simulation loop. This is one of the main features which we need for development of robot control systems. The Planar Manipulators Toolbox has proved to be a very useful and effective tool for many purposes, but is has been primary designed for kinematic and dynamic simulation of planar manipulators and to develop and test control algorithms on the lower control level, especially for redundant manipulators. In the last years, the scope of our research is oriented more in the development of control systems for humanoid and service robots.

These robots have in general a more complex mechanical structure with many degrees-of-freedom. So, complex kinematic and dynamic models are necessary to simulate them.

Furthermore, the control methods and algorithms we are developing are now usually a part of the higher robot control levels and the low level close-loop control algorithms are assumed to be a solved issue. These high level control algorithms can become very complex and may even require parallel computation distributed over more computers.

Considering all new requirements, which are:

- to simulate the kinematics and dynamics of arbitrary chosen kinematic chain describing different manipulators,
- to enable integration of different sensor systems like vision and force sensors,
- to enable simulation of scenarios for complex robot tasks,
- to include the model the robots' environments,
- to visualize the robots and their environment and
- to enable integration of real robots in the simulation loop,

we had to reconsider the concept of the control design environment we will use in future. Based on our good experience with MATLAB/Simulink we have decided that this environment will be the kernel of our simulation tools. However, some of the above requirements can be easier fulfilled by using other tools. For example, the visualization of the robot and the environment can be easily done by dedicated graphics tools. Furthermore, advanced robot control strategies rely intensively on feedback sensor information. The most complex sensor system is the vision system, which can have several configurations and can be implemented on a single computer or on a computer cluster composed of many computers running different operating systems.

To integrate such a diversity of hardware components in unique framework we have decided to use the Ethernet communication and the UDP protocol. In this way, we have maximal possible "degree-of-openness" of the system. Figure 2 shows a typical scheme of our robot integrated environment.

In this scheme, each block can represent a real system or a model of that system.

Note that because we are using ethernet communication between the blocks, different software tools on different platforms can be used to simulate specific parts of the system.

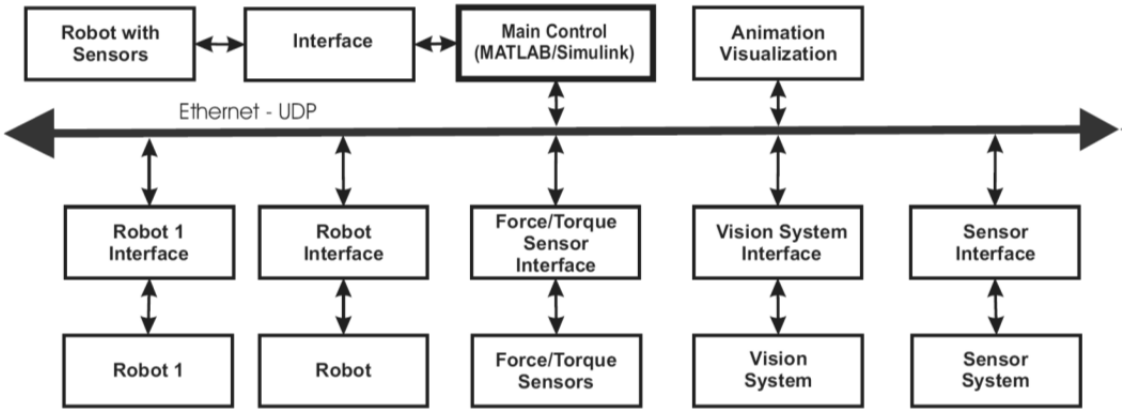

Figure 2. A functional block diagram of the robot integrated environment in Robotics Laboratory including the robot PA10, mobile platform Nomad XR 4000 and sensor systems

Consequently, the simulation environment can consist of several interacting applications, each representing a part of the system.

2 The experimental setup

The experimental setup in our Robotics Laboratory consists of several robots: the Mitsubishi PA10 robot which can be mounted on the Nomad XR 4000 mobile platform, humanoid robot Fujitsu HOAP3 and a 7 DOF humanoid head. They serve to research new approaches in the service and humanoid robotics.

In the following examples we use the Mitsubishi PA10 robot (see Figure 3). The PA10 robot is a general purpose seven degrees of freedom robot arm with brushless AC Motors and harmonic drive transmission in each joint.

The robot has an open architecture as well as in the hardware as in the software, and this provides the possibility to control and modify any aspect of the robot's behavior and to include new sensor information to the control system. The MHI controller is a four layer controller based on ARCNET which allows to control the robot in velocity mode at 100Hz. In same applications, it turned out that the MHI robot controller is not appropriate due to the limited sampling frequency, speed and acceleration limits and redundancy resolution algorithms used for the robot control. Therefore, we have developed an interface which communicates with the robot power system via ARCNET, which enables direct access to the velocity and torque motor inputs with sampling rates up to 700 Hz.

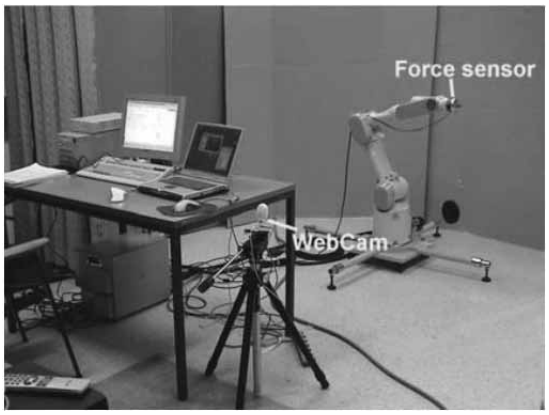

Figure 3. Experimental setup (Mitsubishi PA10, vision system and force sensor)

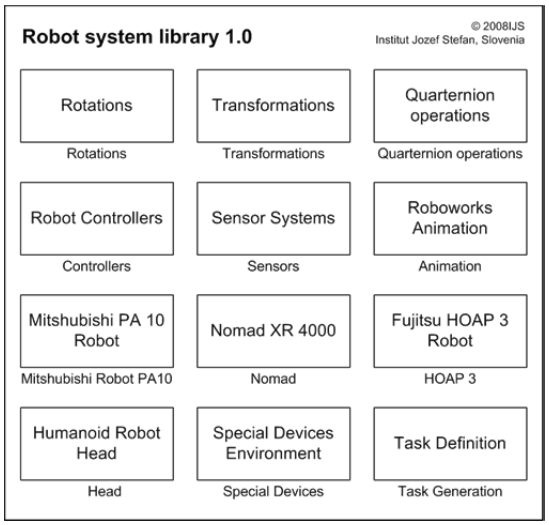

Figure 4. Simulink Robot systems library

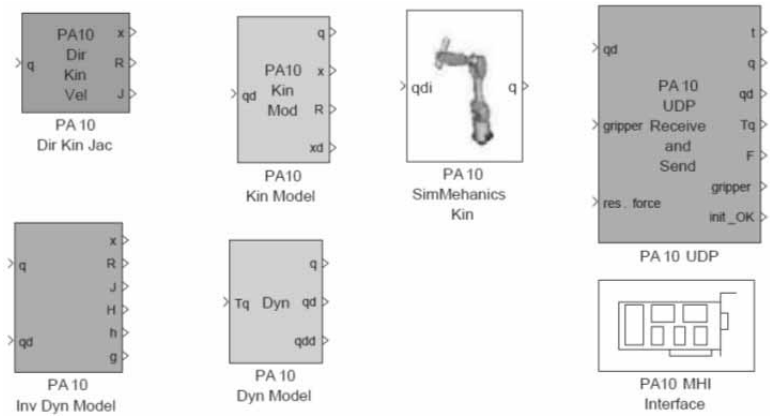

Figure 5. PA10 robot blocks

3 Simulink block library

In Simulink, a system is modelled by combining input-output blocks. To gain the transparency we try to represent a system by the block structure with several hierarchical levels, i.e. by combining different basic blocks subsystems are built which become a single block at the higher level.

In Figure 1 typical robot subsystems can be seen: the trajectory generation, the controller, the model of the manipulator and the environment and the animation of manipulator motion.

Figure 4 shows the Robot systems block library. The goal of the library is to provide blocks which are needed to simulate robotic systems and cannot be modelled with standard blocks.

First of all, this are the blocks for robot kinematic and dynamic models, the blocks for sensors systems, the typical transformations present in robot systems and the special interface blocks for robots, sensors and all other communications.

Additionally, the library includes some blocks with standard subsystems like task space controllers, trajectory generation modules, etc.

3.1 Robot models

Let the configuration of the manipulator be represented by the vector $\boldsymbol{q}$ of n joint positions, and the end-effector position (and orientation) by m-dimensional vector $\boldsymbol{x}$ of task positions. The joint and task coordinates are related by the following expressions

$$\boldsymbol{x} = \boldsymbol{p}(\boldsymbol{q}), \quad \dot{\boldsymbol{x}} = \boldsymbol{J}(\boldsymbol{q})\,\dot{\boldsymbol{q}},$$

$$\dot{\boldsymbol{x}} = \boldsymbol{J}(\boldsymbol{q})\,\dot{\boldsymbol{q}}, \quad \ddot{\boldsymbol{x}} = \boldsymbol{J}\ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}\dot{\boldsymbol{q}} \qquad (1)$$

where $\boldsymbol{J}$ is the Jacobian matrix, and the overall dynamic behaviour of the manipulator is described by the following equation

$$\boldsymbol{\tau} = \boldsymbol{H}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{h}(\dot{\boldsymbol{q}}, \boldsymbol{q}) + \boldsymbol{g}(\boldsymbol{q}) - \boldsymbol{\tau}_{\mathrm{F}} \qquad (2)$$

where $\boldsymbol{\tau}$ is the vector of control torques, $\boldsymbol{H}$ is the symmetric positive-definite inertia matrix, $\boldsymbol{h}$ is the vector of Coriolis and centrifugal forces, $\boldsymbol{g}$ is the vector of gravity forces, and vector $\boldsymbol{\tau}_{\mathrm{F}}$ represents the torques due to the external forces acting on the manipulator.

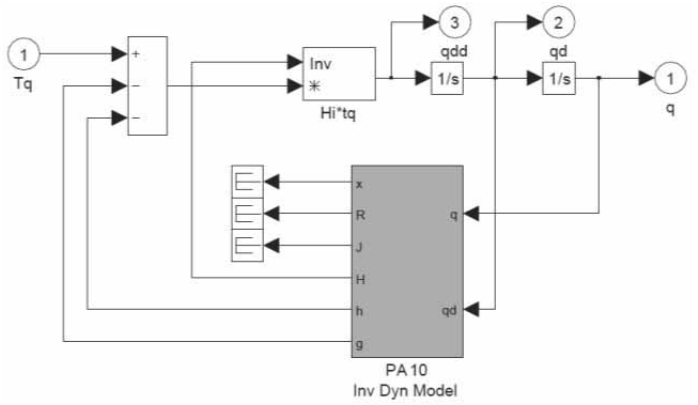

Figure 6. The dynamic model of the PA10 robot

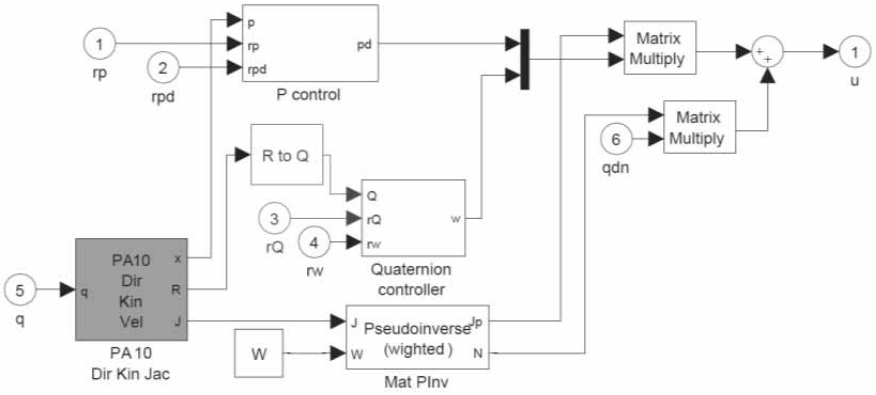

Figure 7. A block diagram representing the task space controller.

The robot model blocks in the library (see Figure 5) represent the basic terms of the system as given in the above equations. Hence, the modelling of the robot is actually only the transformation of the model equations into block diagrams. In the library there are model blocks for all robots we are using. For example, the dynamic model described by the Eq. (2) for the PA10 robot, i.e. the block `PA10 Dyn Mod` in the library, is built using the basic block `PA10 Inv Dyn Mod` as shown on Figure 6, where the model matrices $\boldsymbol{H}$, hand $\boldsymbol{g}$ of the robot mechanism are calculated. The same principle is used for other robots and model types. Therefore, if one wants to use a dynamic model for another robot, he has only to substitute the block `PA10 Inv Dyn Mod` with an adequate block for the desired robot. In the same way, the other common subsystem which includes models is built. Figure 7 shows the task space controller for PA10 robot used later in examples. Here, a kinematic model of a robot is used to obtain direct kinematic transformation (end-effector position vector and rotational matrix) and the Jacobian matrix of the robot, which are needed in the control algorithm.

For hardware-in-the-loop simulation, it is necessary to use hardware interfaces with corresponding software drivers to include a real robots into the control loop. Usually, in case of robotic manipulators interfaces for actuators and sensors are needed. In the past, it was common to use D/A convertors for controlling the actuators and joint positions were measured via incremental encoders. However, contemporary robot controllers and sensor systems enable the communication via different networks protocols like ethernet, Profibus, CAN, etc. In the Robot systems library, each robot has special interface blocks which allow simple integration of them into the simulation loop. For example, for the PA10 robot we have prepared drivers to control the robot using the MHI controller and using the Ethernet and UDP protocol.

Additionally, we have developed external applications for the simulation of our robots, which have the same interface as real robots. Using this applications, the control system realized in Matlab/Simulink is the same for the model or the real robot, i.e. the same interface blocks are used when a model or a real robot is included in the simulation loop. This enables easy and safe testing of control algorithms and the tests can be made even if the real robot is not available. When animation and visualization are also included, the simulation is even more realistic.

3.2 Integration of sensors

Advanced robotics is characterized by the variety of complex sensory system, e.g. vision sensors, force sensors, acoustic sensors, laser scanners, proximity sensor, etc. Therefore it is extremely important to apply as accurate as possible sensor models into the simulation environment. Models of sensors are completely transparent to the design environment, i.e. real sensor can be substituted with the simulated one and vice versa in the control loop.

The integration of sensors depends on their characteristics. Complex sensor systems like vision and acoustic sensors, or more advanced laser proximity sensors require relatively high computational power for signal processing. In many cases, it is difficult to accomplish all required data processing on the local computer. Often we have to apply a remote computer or even a remote computer cluster in order to obtain required computational power.

In such a case, the subsystems are connected through ethernet with UDP protocol. We have developed a special protocol classes for different sensors, actuators and other subsystems. However, the performance is also affected by the communication delays. Therefore, it is favourable to process signals of high framerate sensor, such as joint encoders, tachometers, force sensors, etc. on the local computer.

3.3 MATLAB robot language interpreter

When designing and testing complex robot tasks, it has turned out that standard Simulink blocks which can generate arbitrary trajectory cannot provide all the flexibility needed for complex robot tasks, especially for experimental work in service and humanoid robotics where the desired motion depends on the system/environment states. Commonly used solution for the definition of robot tasks are robot languages. Therefore, we have developed a MATLAB/Simulink block which can interpret the robot language. Included in the simulation it serves as the robot motion generator and supervisor. The developed interpreter module for Matlab Robot Language (MatRoL) is BASIC-like programming language extended with special commands for the robot control and supports all MATLAB interpreter commands.

In this way we have the advantage of a simple robot task definition and access to comprehensive MATLAB computation capabilities. The usage of the robot language is also favourable for the education.

Students can learn and accomplish their laboratory exercises much faster using robot language and the integrated environment allows safe testing of their algorithms on models and final tests on the real robots.

MatRoL is entirely written in Matlab. It has common instructions for the program flow (`IF THEN ELSE`, `FOR NEXT`, `REPEAT UNTIL`, `GOTO <label>`, `GOSUB <procedure_name> RETURN`) and special commands for the robot control (`FRAME`, `MOVE`, `APPROACH`, `DEPART`, `SPEED`, `ACCELERATION`, `FORCE`, `NULLMOVE`, `TRAJECTORY`). Additionally, all Matlab commands can be executed within a MatRoL program as an instruction. In this way we can use powerful Matlab matrix computation capability for controlling robot pose and for complex computation generally needed when vision and force sensors are applied. MatRoL supports various interpolation modes in Cartesian and task space, and supports also redundant robot systems, e.g. a special command `NULLMOVE` is used to define self-movement when kinematically redundant mechanisms are used.

Each robot in the simulation environment has its own MatRoL block, i.e. a special program. Program synchronization is done by assigning global variables, which can be signals, vectors, frames, or other. The MatRoL supports frame the orientation definition in roll-pitch-yaw angles, Euler angles and quaternions, while the interpolation is accomplished using the quaternions. The script in Listing 1 has been used for the vision based manipulation as explained later in Section 4.2.

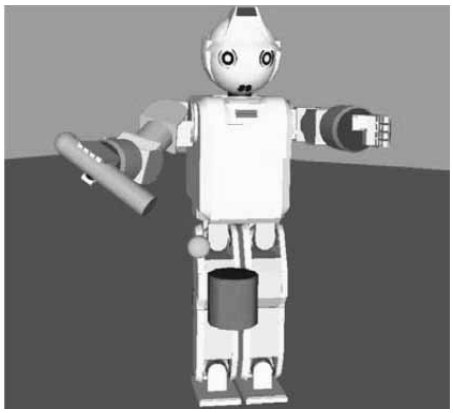

Figure 9. Animation of the HOAP 3 humanoid robot using RoboWorks.

```
1  % Case: Visual servo
2  TRACE 1
3  ! points(1).d=[0.0,0.0,1.1,0,-pi/2,0]';
4  ! points(10).d=[0,0,0,0,0,0]';
5  ! mat(10).d=eye(3);
6  SPEED 0.5
7  ACC 0.5
8  MOVE 1
9  ! ATK_Send('LetterA');
10 GRIP 1
11 GOSUB vservo
12 GOSUB grip
13 STOP
14 % ------------------------
15 LABEL vservo
16 ! disp('VISUAL SERVO');
17 LABEL loop
18   ![xc,var(1)]=Vis_Ser(points(10).d,mat(10.d);
19   IF (var(1) < 0.005)
20      GOTO exit
21   ENDIF
22 GOTO loop
23 LABEL exit
24 ! disp('DONE');
25 RETURN
26 % ------------------------
27 LABEL grip
28 SPEED 0.1
29 DELAY 1
30 SPEED 0.1
31 TDEPART 0 0.03 0
32 DELAY 0.5
33 TDEPART 0 0 0.15
34 DELAY 0.5
35 GRIP 0
36 DELAY 0.5
37 TDEPART 0 0 -0.01
38 SPEED 1
39 move 1
40 TDEPART 0 0.04 0
41 GRIP 1
42 RETURN
43 END
```

Listing 1. MatRoL script for the vision-based manipulation

3.4 Visualization and animation

It is very important to visualize the simulation results. Especially in robotics it is necessary to "see" the motion of the robot and objects in the working environment. In our system we rely on external software for the visualization and animation of robots. In general, joint angles of robotic manipulators as well as the position and orientation of the other simulated objects in the scene are passed to the visualization tools using TCP/IP or UDP protocol.

Figure 10. Animation of the PA10 robot in Blender.

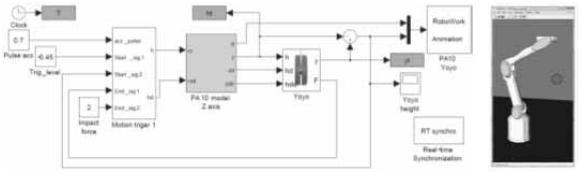

Figure 11. Yoyo simulation: top level block scheme in Simulink and animation of the PA10 robot and yo-yo.

Currently, we have integrated into our simulation environment two visualization software packages - RoboWorks [16] and Blender[17].

Roboworks incorporates simple, but efficient modeler. Because of its simplicity RoboWorks package is the favourable tool for the visualization of simpler systems, i.e. one or two robots in non-complex environment. Figure 9 shows the animation of our HOAP 3 humanoid robot and also in the following examples the RoboWorks environment has been used for the visualization.

For more complex scenes we use Blender, an open source multi-platform 3D computer animation program, which has a lot of features that are potentially interesting for engineering purposes, such as the simulation and programming of robots, machine tools, humans and animals, and the visualization and post-processing of all sorts of data that come out of such biological or artificial “devices”.

Blender supports also scripts (via Python interfaces to the core C/C++ code), hence it can be extended in many different ways. Among others, Blender has the capability of placing moving cameras at any link of the kinematic chain, it supports the real time photo realistic rendering for the virtual reality simulation and has also a physics engine for the simulation of the interactions between entities.

3.5 Real-time simulation

The real-time performance of the control algorithm is very important when dealing with low-level control. However, when developing higher level control algorithms real-time may be also important especially when high sample frequency improves the performance of the system. Therefore, when manipulator-in-the loop simulation is performed, the simulation system which controls the robot system has to provide real-time capabilities and enable high sample frequencies. There are many real-time operating systems as Real Time Linux, QNX, EYRX, SMX, etc. Disadvantages of these operational systems are time-consuming software development and incompatibility with other systems. The algorithms are usually written in C or some other low-level programming language, where more sophisticated control algorithms requires more time and increase the chance of error. Due to the above mentioned disadvantages of some real-time operation systems, we use the MATLAB/Simulink and the xPC Target operation system whenever possible [18]. xPC Target enables real-time simulation and hardware-in-the-loop simulation using corresponding interfaces. It is a good prototyping tool that enables to connect MATLAB/Simulink models to physical systems and to execute simulation in real-time on PC-compatible hardware. As xPC Target supports also UDP communication, this was also one of the reasons to select the UDP for the communication between different applications in the simulation environment. Nevertheless, using MATLAB/Simulink and xPC Target environment brings some disadvantages. Most of the hardware used for a robot control, which is available on the market, does not provide drivers for xPC Target. Therefore, we had to develop drivers for our robots and sensors.

4 Case studies

To show the efficiency, flexibility and usability of our control design environment we outline some typical experimental examples using the Mitsubishi PA robot and the mobile platform. We explain the complete design of the control system different simulation schemes used in this procedure from the "pure" Simulink simulation schemes, where the complete system is simulated in MATLAB/Simulink, to the hardware-in-the-loop schemes, where a real robot and sensor systems are part of the simulation loop and only the controller is realized in MATLAB/Simulink.

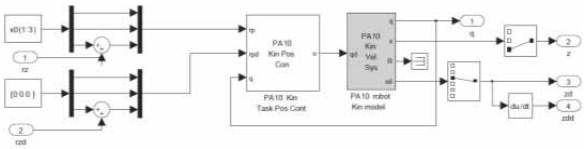

Figure 12. PA10 model with kinematic task space position controller

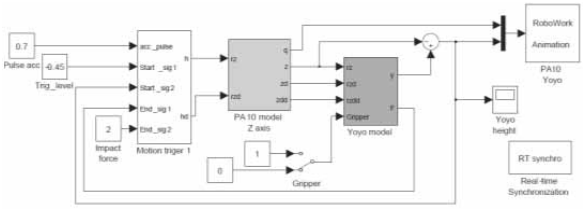

Figure 13. The case with kinematic PA10 robot model and external yo-yo simulator.

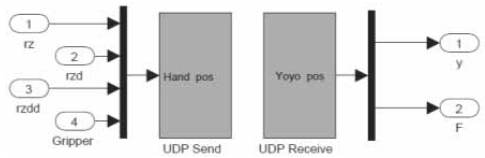

Figure 14. Interface for external yo-yo simulator (`Yoyo model` block)

4.1 Playing yo-yo

In the first example we use the Mitsubishi PA10 robot arm to play yo-yo. The objective of playing the yo-yo is to keep the amplitude of the yo-yo at a desired level. The yo-yo is tied to the tip of the robot. To be able to play the yo-yo it is necessary to know the position of the yo-yo and the force in the string or the velocity of the yo-yo (depending on the control algorithm). A webcam has been used to measure the position of the yo-yo.

To measure the string force a JR3 force/torque sensor mounted on the end-effector of the robot was used. The experimental setup is shown in Fig. 3.

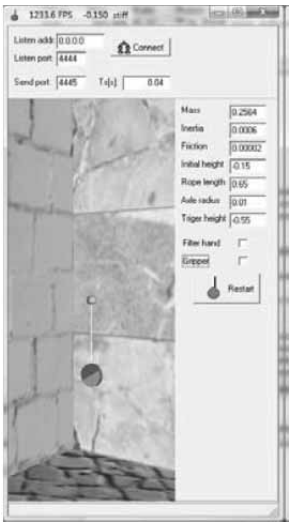

Figure 15. External yo-yo simulator.

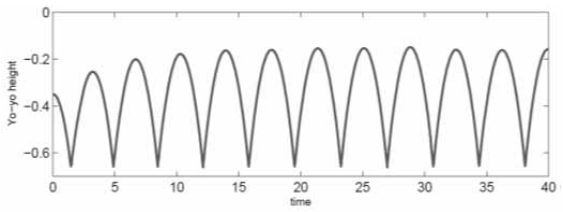

Figure 16. Swinging yo-yo motion - Simulation results

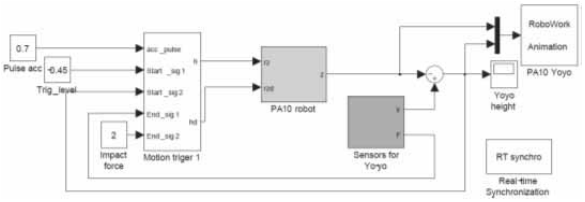

Figure 17. Hardware-in-the-loop simulation (real PA10 robot, force sensor and vision systems are in the simulation loop)

The control should be implemented on PC's in MATLAB/Simulink environment and we wanted to use the PA10 motion control board which allows to control the end-effector positions of the robot. In the first step of the control design when different control strategies have to be tested, we simulated the whole system in Simulink. We used the PA10 kinematic model and we had to develop a Simulink model of the yo-yo. The top level simulation scheme is shown in Figure 11. The main three blocks are the controller, the robot model and a special model of the yo-yo [19]. As we want to move the robot end-effector only in the vertical direction the z-axis motion (x and y positions are fixed to the initial values), we have to use a kinematic task space controller. This subsystem can be easily composed by combining blocks in our Simulink library as it is shown in Figure 12.

After the best control strategy has been verified using this simulation scheme, the next step is to test the control when the sensor systems information is obtained via Ethernet connection. Therefore, we have developed a special yo-yo simulator, which receives the hand position and sends the position of the yo-yo and the string force via Ethernet connection using UDP protocol (see Figure 15). The simulation scheme is the same except that instead of yo-yo Simulink model the corresponding UDP interface blocks are used (see Figures 13 and 14).

As the external yo-yo simulator is a real time simulator, also in Simulink real-time simulation should be used.

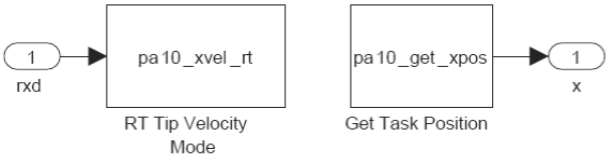

Figure 18. The interface block for PA10 task space position control

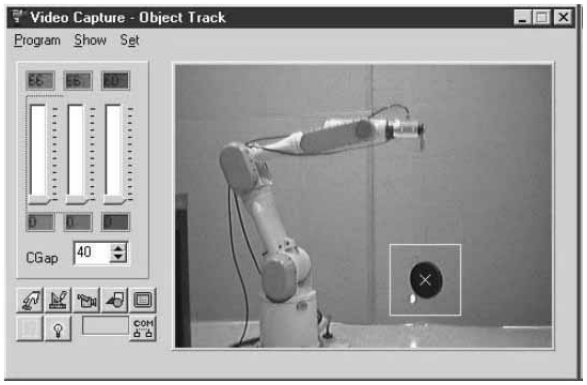

Figure 19. Capturing and identification of the yo-yo position with the webcam.

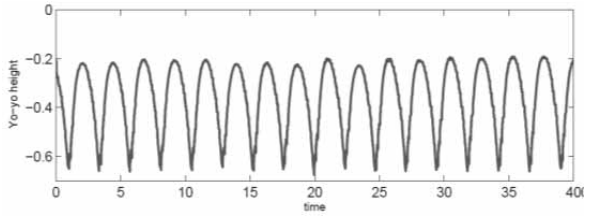

Figure 20. Swinging yo-yo motion - Experimental results.

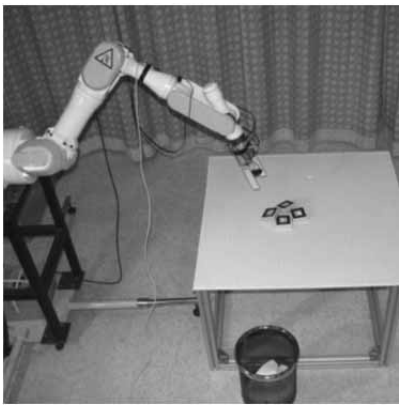

Figure 21. Experimental setup for vision based manipulation of objects

As the sampling frequency in this case is rather low (100 Hz for robot control and 25 Hz for vision system) and the computation time of the Simulink model is small enough, we can use a special block for real-time synchronization. After tuning the controller parameters the simulation results for the yo-yo swinging height as shown on Figure 16 have been obtained.

Finally, when the designed control algorithms give satisfactory simulation results, we can test the control strategy on a real system. In manipulator-in-the-loop simulation, the model of the PA10 robot is replaced by the corresponding interface blocks. The position of the yo-yo is now obtained from the vision system and the force sensor via the same interface blocks as when the yo-yo simulator has been used. Figure 19 shows the user interface of the webcam based vision system.

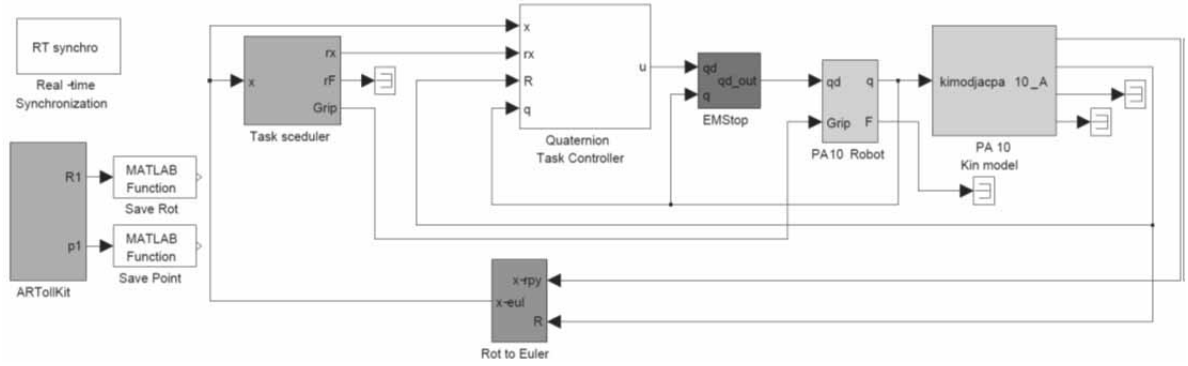

Figure 22. Vision based manipulation: Simulink block scheme

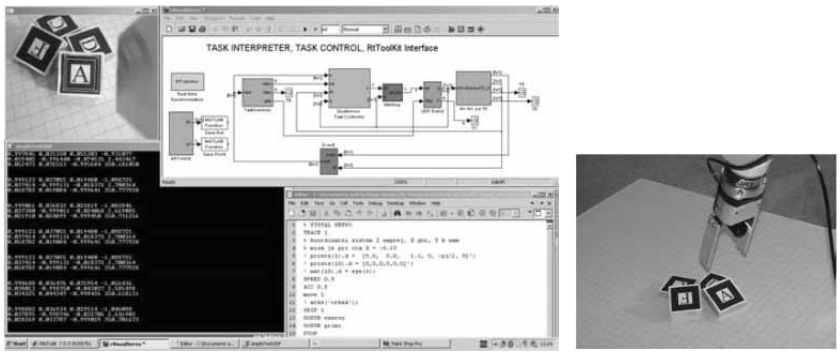

Figure 23. Vision based manipulation experiment: Robot is picking cube "A"

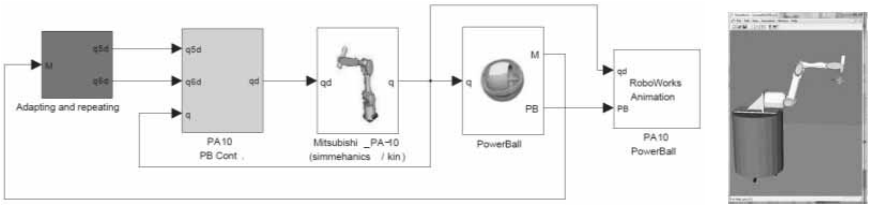

Figure 24. Power®Ball simulation: top level block scheme in Simulink and animation of the PA10 robot on Nomad platform and the Power®Ball

As explained before, special Simulink drivers for interfacing the PA10 robot control board, the JR3 force sensor and the webcam based vision system are already part of our Simulink library. Therefore, the user just replaces the model blocks. The corresponding scheme is shown in Figures 17 and 18.

From the top level scheme it can easily be seen that the controller part of the system has not been changed and is the same as in the previous simulation schemes. The final experimental results are shown on Figure 20. By comparing them with the simulation results on Figure 16 one can see that they are very similar. This confirms that simulation tools can be an important tool when designing control system.

4.2 Vision-based manipulation

In the second example we show the visual tracking experiment. The task for the robot has been to compose a text using cubes marked with letters.

The cubes have been randomly dispersed on the table. The robot has to identify a cube with the desired letter using vision, to grasp this cube and to place it on the table in order to compose the desired text. Note that cubes were arbitrary rotated in all three axes. Therefore, the visual tracking algorithm has to track not only the position of a cube but also the object orientation. Figure 23 shows the experimental setup.

To detect the object position and orientation we have used a USB webcam and the "ArToolKit" - an open source software library for building Augmented Reality (AR) applications [20]. These are applications that involve the overlay of virtual imagery on the real world.

Although, augmented reality is generally not needed in robotics, ArToolKot was chosen because of its object recognition capabilities. ArToolKit is capable of calculating 3D object position and orientation using single camera. The pose estimation is based on exact knowledge of the observed object geometry and its projection in the camera.

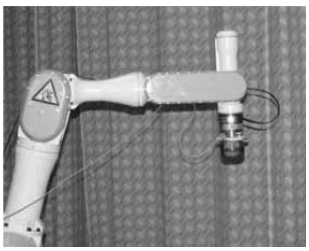

Figure 25. Experimental setup for spinning up the Power®Ball with PA10 robot

4.3 Playing "Power®Ball"

In the third example the robot should perform the spinning of a Power®Ball – a hand held gyroscopic toy or exerciser. To accelerate the rotor of the device with a robot, we first measured the way a human does it. Using the results from the motion capture, we transferred the movement of the wrist to the end-effector movement of the robot. For a successful spin-up a synchronization of the exerted torque with the control velocity of the circular motion is necessary. Figure 25 shows the experimental setup. Different control approaches using feedback information from the velocity counter and force/torque sensor were applied.

First, they have been tested using SimMechanics model of the PA10 and the model of the Power®Ball. Figure 24 shows the block scheme and the animation of the system in RoboWorks. Finally, the experiment with a real robot in the loop has been done

The model of the robot and the Power®Ball has been replaced with the interface blocks as explained before. Figure 25 shows the experimental setup.

5 Conclusions

The concept of the presented control design environment is a result of our experience in the use of robots inresearch and education. It has proved to be a very useful and effective tool for fast and safe development and testing of advanced control schemes and task planning algorithms, including force control and visual feedback. The main part is implemented in MATLAB/Simulink and we have developed models for the robots and sensors used in our laboratory. To integrate the variety of components in an unique framework we have decided to allow the use of different tools for their simulation. So, the simulation environment can be composed of more than one application and the Ethernet is used for the communication between them. In this way, our environment is very open and can be very easily extended and adapted to different requirements and applied to any types of robotic manipulators. We have augmented the simulation with the animation and we show the importance of the possibilities offered by the simulation in the "virtual" world. One of the most important features of our simulation environment is that the testing on real robots is made very easy — the model real systems is simply replaced in the simulation loop by proper interface blocks. For that purpose, we have developed interfaces for the robots and sensors. Additionally, we have developed external applications which simulate certain robot subsystem and use the same interface as a real system. In this way, the user can test algorithms using the final control system but on a system on models which is very easy. Last but not least, it is an efficient tool for educational purposes. Thus, it should be of interest to the researchers involved in the development of advanced robot systems, and for teaching laboratories.

References

[1] Yehong Zh., R.P. Paul. *Robot Manipulator Control and Computational Cost*. Technical Report MS-CIS-88-10, University of Pennsylvania Department of Computer and Information Science, 1988. http://repository.upenn.edu/cis reports/621.

[2] R.G. Fenton, F. Xi. *Computational analysis of robot kinematics, dynamics, and controlusing the algebra of rotations*. IEEE Trans. on Systems, Man, Cybernetics, (6):936 – 942, June 1994.

[3] J.-C. Latombe. *Controllability, recognizability, and complexity issues in robot motion planning*. In Proc. 36th Ann. Symp. on Foundations of Computer Science, pp. 484 – 500, Los Alamitos, CA, USA, 1995.

[4] J.M. Lambert, B. Moore, M. Ahmadi. *Essential Real-Time and Modeling tools for Robot Rapid Prototyping*. In Proc. 6th Int. Symp. on Artificial Intelligence and Robotics & Automation in Space i-SAIRAS 2001, Quebec, Canada, 2001.

[5] G. Alotto, B. Bona, T. Calvelli. *Prototyping Advanced Real-Time Robotic Controllers on Linux RTAI Systems with Automatic Code Generation*. In Proc. Int. Conf. Mechatronics and Robotics 2004, Aachen, Germany, 2004.

[6] V. Lippiello, L. Villani, B. Siciliano. *An open architecture for sensory feedback control of a dual-arm industrial robotic cell*. An International Journal Industrial Robot, 34(1):46–53, 2007.

[7] L. Žlajpah. *Integrated environment for modelling, simulation and control design for robotic manipulators*. Journal of Intelligent and Robotic Systems, 32(2):219 – 234, 2001.

[8] L: Žlajpah. *Simulation in robotics*. Math. comput. simul., doi:10.1016/j.matcom.2008.02.017, 2008.

[9] *RobotWorks – A Robotics Interface and Trajectory generator for SolidWorks*, www.robotworkseu.com.

[10] Tecnomatix. *ROBCAD/Workcell*, User's manual, 1988.

[11] P. I. Corke. *A Robotics Toolbox for MATLAB*. IEEE Robotics & Automation Magazine, 3(1):24–32, 1996.

[12] The Mathworks. *SimMechanics, User's Guide*, 2005.

[13] Microsoft Robotics Studio. MSDN. http://msdn2.microsoft.com/en-us/robotics/default.aspx.

[14] Cyberbotics Ltd. *Webots User Guide*, 2005.

[15] Modular Controller Architecture 2 - MCA2: http://mca2.org/

[16] Roboworks: http://www.newtonium.com/public_html/Products/RoboWorks/RoboWorks.htm.

[17] Blender: http://www.blender.org/.

[18] D. Omrcen. *Developing Matlab/Simulink and XPC target real-time control environment for humanoid jumping robot*. In 16th Int. Workshop on Robotics in Alpe-Adria-Danube Region - RAAD 2007, pp. 18–23, Ljubljana, Slovenia, 2007.

[19] L. Žlajpah. *Robotic yo-yo: modelling and control strategies*. Robotica, 24(2):211 – 220, 2006.

[20] ARToolKit: http://www.hitl.washington.edu/artoolkit.

Corresponding author: L. Žlajpah,
"Jožef Stefan" Institute
Jamova cesta 39, 1000 Ljubljana, Slovenia
leon.zlajpah@ijs.si

Received & Accepted: MATHMOD 2009: -
Revised: January 2010 -
Accepted:may 22, 2010