15

Virtual Reality Control Systems

Tomislav Reichenbach, Goran Vasiljević and Zdenko Kovačić
University of Zagreb - Faculty of Electrical Engineering and Computing
Croatia

1. Introduction

Control systems theory and application, as engineering in general, has greatly benefited from development and progress of the computer science and technology. An exponential growth in computational power in the last few decades gave rise to new methods, algorithms and theories. Gradually, the ability to deal with more and more complex problems was within reach. Unsolvable problems, or more precisely, formerly lengthy problems, became solvable in real-time.
At the same time, the controlled systems also became more advanced and complex. Better precision and control is required within each new generation or series, this naturally leading to increased computational power requirements; thus reducing the impact of the computational power growth.
Technological advance in both hardware and software, namely in hardware graphic capabilities and complementary software, allowed creating copies of real systems and environments in the computer, and thus a new technology emerged - so called Virtual Reality.
Since the term "Virtual Reality" has different connotations in today's culture, mostly due to popular science and entertainment industry (mis)interpretations, a precise meaning and the term used in this article - "Virtual Reality Systems (VRS)" - is equivalent to the meaning conveyed by "3D Graphic Simulator". Furthermore, unless clearly specified, no clear distinction is made between the virtual reality's classes - mixed reality, augmented reality or virtuality (see Fig. 1). In general, the umbrella term "virtual reality (VR)" is used to include anything forming part of the virtuality continuum.

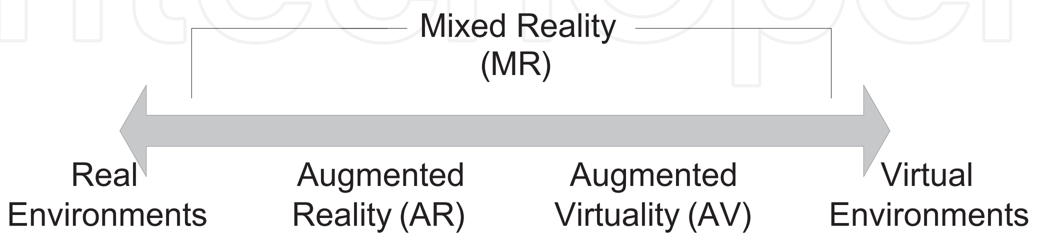


Fig. 1. Virtuality Continuum (Milgram & Colquhoun, 1999)

As the VRS turned more sophisticated due to the technological advance, new functionality became available. From the first plain 3D visualization system, subsequent major evolutionary steps are mainly: the ability to detect collisions, the introduction of physic based modeling and lately the addition of fluids, clothes and deformable bodies.

Entertainment industry embraced VRS almost from the start, but thenceforth a lot of different industries found VRS valuable, e.g. Bioinformatics (genome and protein folding research), Medicine (Virtual surgery and diagnostics, Therapeutics) and even Sociology, with virtual worlds-cultures like "Second Life" as a product and producer of culture simultaneously.

The VRS also are broadly used in control system prototyping, design and simulation. The benefits of using VRS are principally - reduced costs of development, mitigation of non-conformity and failure risks and greatly reduced prototyping and design time.

The authors became involved with VRS in control a decade ago, and continue to keep themselves engaged with VRS applications. During that time several software applications that provided the evaluation ground for new applications were developed. The aim of this chapter would be to enumerate, systematize and formalize different practices and techniques applicable to the VRS. New methodologies, showing how the VRS systems can be used complementary to the control systems, will be introduced - creating a new paradigm - "Virtual Reality Control Systems" (VRCS). Introduced methods have particularly suitable, but not exclusive application in robotics.

For the purpose of establishing the relations between virtual environments and control systems, the VRCS taxonomy is proposed, subject to the functionality available in the VRS and the way that functionality is used.

Hence, the following VRCS categories are established:

1. VRCS Visualization
2. VRCS Monitoring - Virtual sensors
3. VRCS Open loop control
4. VRCS Closed loop control

Each subsequent category adds more interdependency between the virtual and control system constituting that particular category, in respect to its antecedent.

An illustrative example for each category will be presented, with the application either in robotic arms, mobile robotics or humanoid robotics control. Data interchange between a real and a virtual system is described with an emphasis on how the data from the real system is incorporated into the VRCS.

2. Visualization

The first and the simplest category, in the defined taxonomy of the VRCS, is named "visualization". It is a baseline and principal prerequisite for other established VRCS categories, which are achieved simply by improving and adding features to the visualization core.

2.1 Introduction

Visualization itself, in general, is a technique of creating images, drawings or animations to convey information; to communicate both concrete and abstract ideas.

Visualization as a technique is not new, rather it is around 30,000 years old, dating back to the first cave wall drawings. Today, visualization has ever increasing applications in science, education, engineering, medicine, entertainment, etc. Further categorization according to the general objective in given applications is established as follows: Scientific visualization, Educational visualization, Information visualization, Knowledge visualization, Product Visualization, Visual Communication and Visual analytics ((Wikipedia, 2010)).

2.2 Definition

VRCS visualization, according to this categorization, falls into scientific visualization group. The precise definition of VRCS visualization is given with the following statement: *The use of computer-generated 3D visual representation of data from simulations or experiments to support analysis, exploration and comprehension of systems.*

In addition, for the general computer-generated 3D visualization to be VRCS visualization type, following conditions must be satisfied:

Perspective - Visualization is done in three dimensional (3D) space (with time as a fourth dimension)

Performance - Visualization execution performance is at least at the same level as the real system performance (real-time)

Model-based visualization - 3D models used are as accurate as possible or as circumstances require. (A digital construction of a real object is made directly from the available scientific data).

2.3 Introducing perspective

For three dimensional illusion of perception on the two dimensional screen, a proper technique for displaying depth and the corresponding mathematical method is required.

Historically, even though techniques for displaying depth in visualization were used before, it was not until Renaissance Italy that physically (optically) correct and mathematically valid technique was invented - the first major advance in scientific visualization. The first application of "true perspective" (commonly known today as geometric perspective[1] or central projection) is attributed to Filippo Brunelleschi when he painted the Baptistery of St.John in the Piazza Del Duomo in Florence. Remarkably enough, there is another reason for this application to be noteworthy in the context of this paper. As a person would stand at the exact spot where Brunelleschi painted the panel, he could observe the original scene through the tiny eyehole in the panel, or by holding the mirror in front of the panel look at the accurately drawn painting indistinguishable from the original. However, the sky was not drawn on the panel, rather coat of burnished silver was put there - creating a mirror. Consequently, a person looking would see the clouds drift across the upper part of the painting. "Here, in this calculated confusion of real world and artifice, the technological quest for virtual reality was launched."(Talbott, 1995)

The central perspective has one vanishing point where all the lines coming from the observer's viewpoint intersect. Two-point perspective exists when the picture plane is parallel to a Cartesian scene in one axis (usually the z-axis) but not to the other two axes. Therefore, there is one set of lines parallel to the picture plane and two sets of lines oblique to it, which converge to two vanishing points. Similarly, three-point perspective axes are not parallel to any of the Cartesian's three axes in the scene. In three-point perspective each of the three axes of the scene corresponds with one of the three vanishing points.

2.4 The graphics pipeline

The second prerequisite, for VRCS visualization class, is that the visualization performance has to be at least as fast as the performance of real system. However, it should be stressed

[1] The word "perspective" derives from the Latin perspectiva, a term adopted in the Middle Ages to render the Greek ὀπτική (optics)(Osborne, 2010).

that this only sets the minimum execution speed requirements, because in some cases we would like the virtual environment to execute the tasks as quickly as possible.
To accomplish this, rigorous mathematical method for perspective calculations is required. Subsequently, hardware capable of using that method to render virtual 3D environments is essential.
The essence of mathematical method used to describe the perspective is surprisingly simple - a 4x4 matrix. Consequently, visualization uses linear algebra and matrix multiplications for correct 3D perspective positioning, for rotations and translations in 3D space, for scaling and skewing, for projecting onto 2D surfaces, etc.[2]

2.5 Graphics pipeline

Basic building blocks in rendering process that creates a virtual environment are numbers. Numbers form coordinates that describe points in 3D space, which arranged create surfaces that shape a virtual world. Additionally, the points that create a virtual environment are further projected to a plane (computer screen) creating the viewpoint. The actual process and data flow is naturally more complex as the graphics pipeline takes advantage of the available hardware to process efficiently and to render scenes to a display (Walnum, 2003). Figure 2 conceptually illustrates the main components of the (DirectX) graphics pipeline.[3]

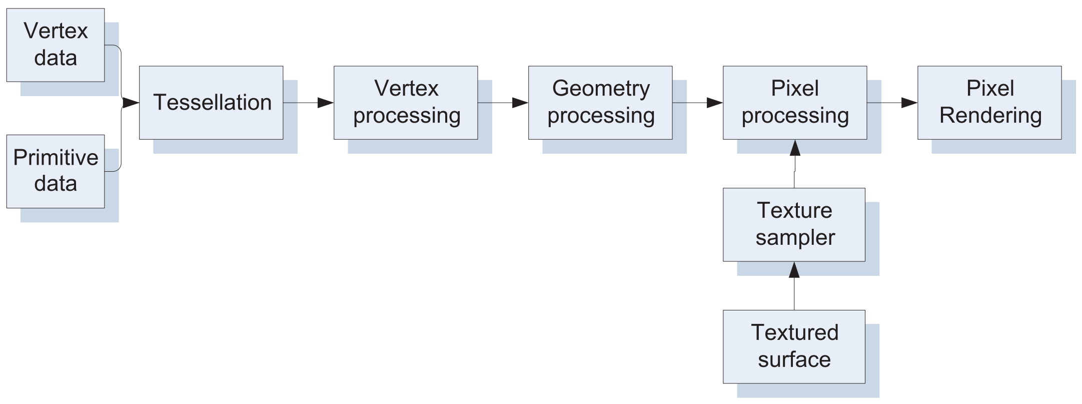


Fig. 2. Direct 3D graphics pipeline

2.6 Engine

Before the VRCS's (software) integral parts can be further described, concepts of engine, simulation engine and visualization engine need to be introduced.
An **engine** is general term for encapsulated block of software functionality, distinct from the user interface of that particular software. It can be either a library or collection of libraries, platform or SDK (software development kit). Engine has to provide a way to communicate with another software. Typically it is done through a dedicated API (Application Programming Interface).
Simulation engine, in general, is an engine with simulation functionality. Specifically, in the VRCS context, simulation engine is an engine capable of simulating dynamic systems.

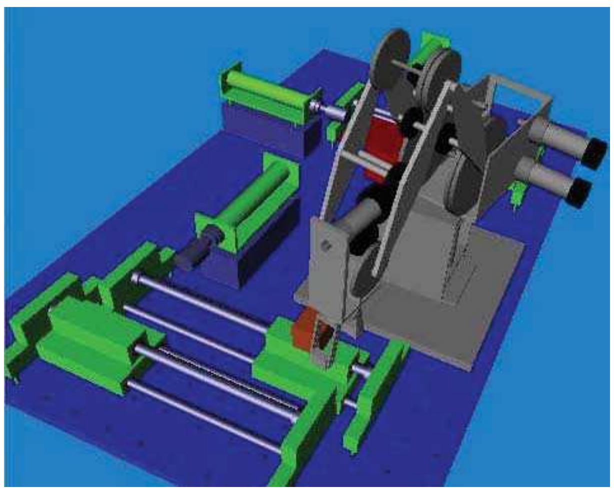
[2] For additional reading regarding 4x4 matrix transformations in 3D perspective display, 3D rotations, translations and scaling, please refer to (Goldman, 1990),(Möller & Haines, 1999) or (Verth & Bishop, 2004).
[3] A similar process sequence is taking place in the OpenGL graphics pipeline (OpenGL is second major 3D standard along the DirectX)

Visualization engine is analogously an engine with visualization capabilities. Moreover, in the VRCS context it has to correspond to the previously defined requirements for the VRCS visualization. Visualization engine in normally a part of CAD software, computer games or simulation package.

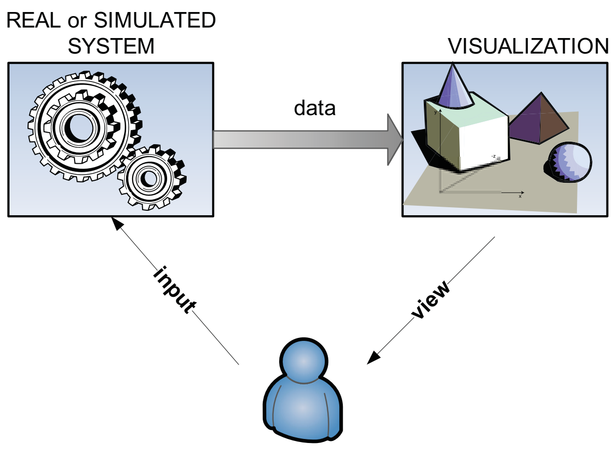
Finally, an engine that possesses both simulation and visualization functionality is named **VR simulator**. During the development of the VRCS systems, a new proprietary VR Simulator was designed and developed in C++ using open-source or closed-source components, SDKs (OGRE, NxOgre, PhysX) and standards (Collada, XML, DAE) (Reichenbach, 2009).

One of the most commonly used examples of visualization engine is Virtual Reality Modeling Language (VRML) viewer. It conforms to all the requirements for VRCS Visualization and is used in many commercial products as a visualization tool. The Virtual Reality Toolbox[4] that ships with Matlab/Simulink platform is one of the examples (MathWorks, 2010).

The introduction of the VRML viewer into the proprietary platform (see (Kovačić et al., 2001) and (Smolić-Ročak et al., 2002)) for testing Flexible Manufacturing Systems (FMS) was one of our group's first forays into the field of 3D computer graphics. This typical example of the VRCS visualization (see Fig.3(a)), served as a stepping stone for next VRCS projects.



(a) VRML visualization - FlexMan (Larics (FER) -2002.)



(b) Information flow - Visualization case

Fig. 3. The VRCS Visualization

The visualization concept is shown in Fig. 3(b). The system has two main components, real or simulated system and visualization engine. Data flow is unidirectional – from real system/simulation to the visualization engine – and the visualization results (actions) are presented to the user. If no real system is present and system is simulated (within the simulation engine), one must note that this concept places no restriction on the software architecture. Simulation and visualization engine can run either in the same process, or as different processes, either on one or various machine(s), etc.

The "data" term used in the figure has broader connotations than normally associated with this word. What is communicated can be raw data in predefined format; i.e. variables, states,

[4] Virtual Reality toolbox was recently renamed Simulink 3D Animation

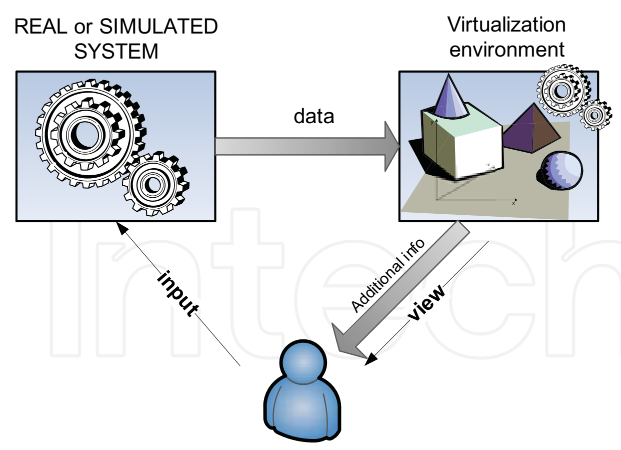
alerts; commands like direct function calls that manipulate objects (normally through some API) or events and messages that correspond to the particular situation occurring.
One important observation should be emphasized here. The VRCS visualization does not provide any new information, states, inputs or outputs to the particular system being subjected to the simulation or analysis. Nevertheless, there is an apparent cognitive value immediately available to the external observer performing or monitoring the undergoing simulation, through the act of observing the system behavior in a "visual" manner.

3. Monitoring - virtual sensor

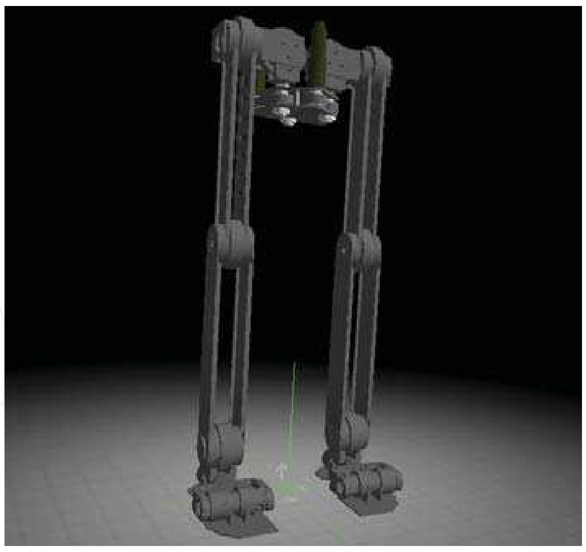
Even though the computers have been used almost from their beginning for scientific visualization, insufficient graphics power limited their usefulness. With the general introduction of 3D hardware acceleration in mid '90, and enhancements following with each new generation, it became possible to produce more and more realistic looking visualization on a common personal computer. Apart from major advances in graphics, which exceed the scope of this paper, important step was introduction of a GPGPU (General Purpose Graphics Processing Unit) concept. This concept allows the use of graphics accelerator's massive computational power for general-purpose, in particular for real-time dynamic system simulations ("physics simulation").
In contrast to the VRCS visualization, where there was no additional logic and every action was the product of received data or command, in monitoring at least some simulation engine parts are merged with visualization engine. This integration is done preferably within the corresponding hardware (GPGPU), or with dedicated software (and paying the performance penalty).
Consequently, in VRCS monitoring case, there is additional information available to the user as a product of the visualization and simulation integration (see Fig. 4(a)). The communication between system or simulation and visualization engine is still unidirectional, as it was in the visualization case.
This additional information, representing new state, variable or an indicator of the system behavior, is something that was not previously available in the system. This information can lead to new insights, a better understanding of the system or to improved quality in the control of the system.
To help better understanding, an illustrative example of the monitoring concept is provided. The humanoid robot presented in Fig. 4(b) is modeled in detail in 3D CAD (Computer-aided- design) software. Not only is the robot's appearance data described but also its physical characteristics are specified, i.e., its links, masses, inertias, motors, etc. This physical properties are essential for dynamic system simulation. There are several predetermined points on the soles of humanoid robot's feet that are sensitive to the force exerted upon them. The resultant force and position, in respect to each sole, therefore can be calculated. This new sensor is termed **virtual sensor**, and in this particular case virtual pressure sensor. Figure 4(c) shows the resultant forces (pressure) when the lateral force is applied to the humanoid robot's body.

4. Open loop control

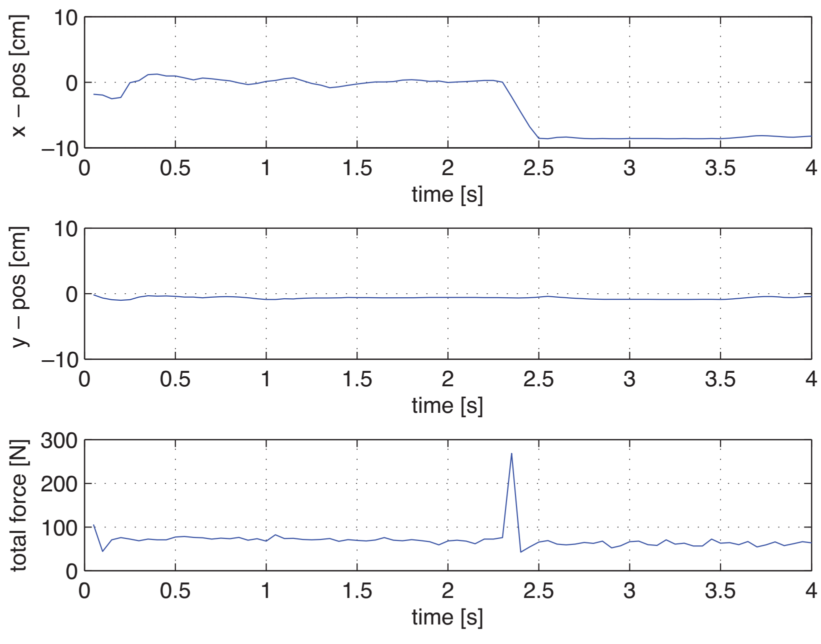
Further extension to the VRCS monitoring class, is the VRCS open-loop case (see Fig. 5(a)) where the distinction is the presence of a controller (and controller strategy in general). The prerequisite for the basic functioning of the controller is a presence of a virtual environment.



(a) Information flow - Monitoring - Visual sensor case



(b) Humanoid robot

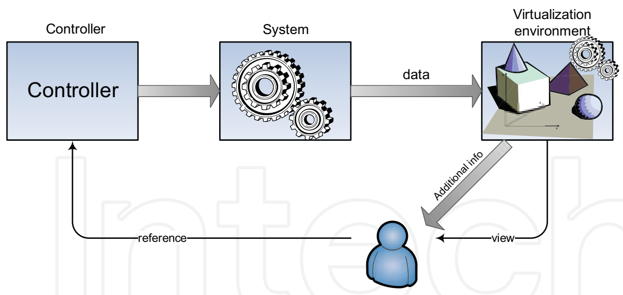


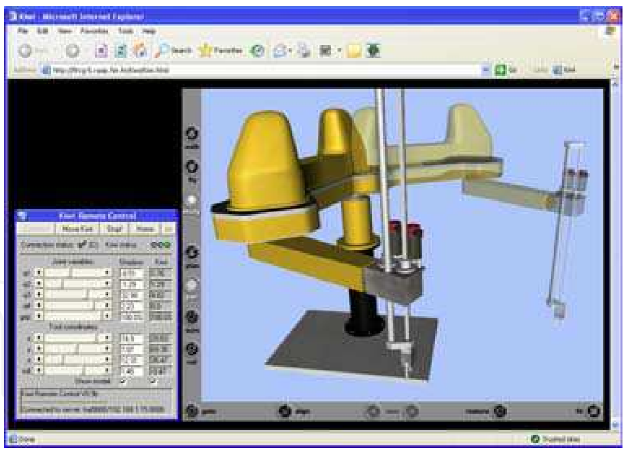
(c) Resultant virtual pressure sensor

Fig. 4. The VRCS Monitoring

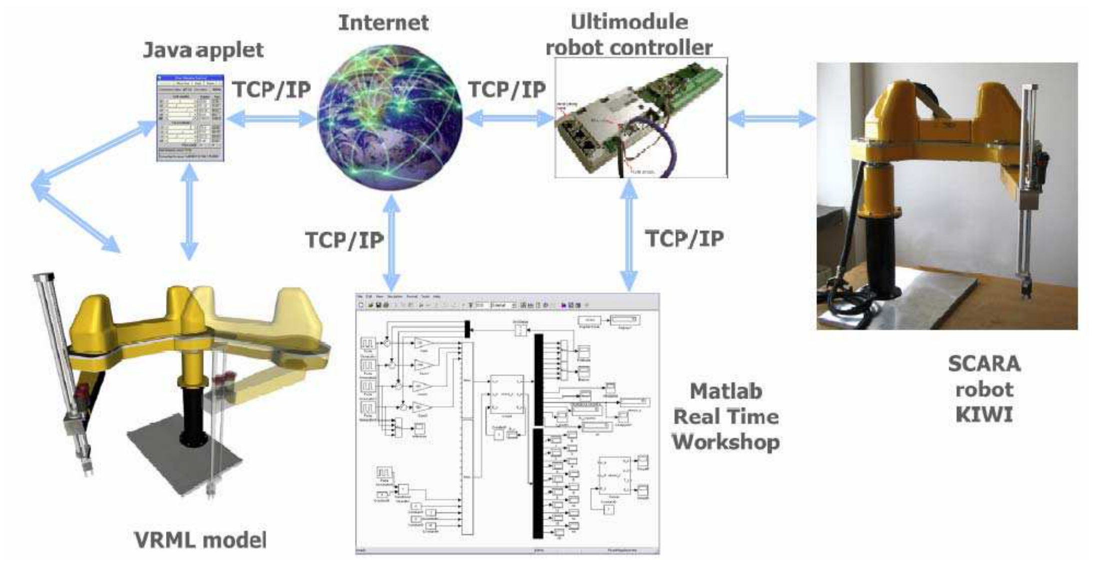
Systems that fall into this category generally are remote systems that provide virtual environment of the (probably not so easy to reach) system. No other form of the feedback is available, except the supervising operator who can act only as an observer or as a man-in-the-loop control feedback.

One such example is the implementation of the Internet accessible robot control laboratory (see (Kovacic et al., 2007)) based on the use of the Matlab Real Time Workshop (see Fig. 5(c)). The developed system contains a four degrees of freedom SCARA robot, Ultimodule robot controller that communicates via TCP/IP with the server application embedded into the Simulink function block and a client web application (see Fig. 5(b)) with VRML-based GUI and Java applet for setting joint positions or tool center positions.





(a) Information flow - VRCS open-loop case (b) The client application



(c) Internet accessible robot control laboratory - the concept

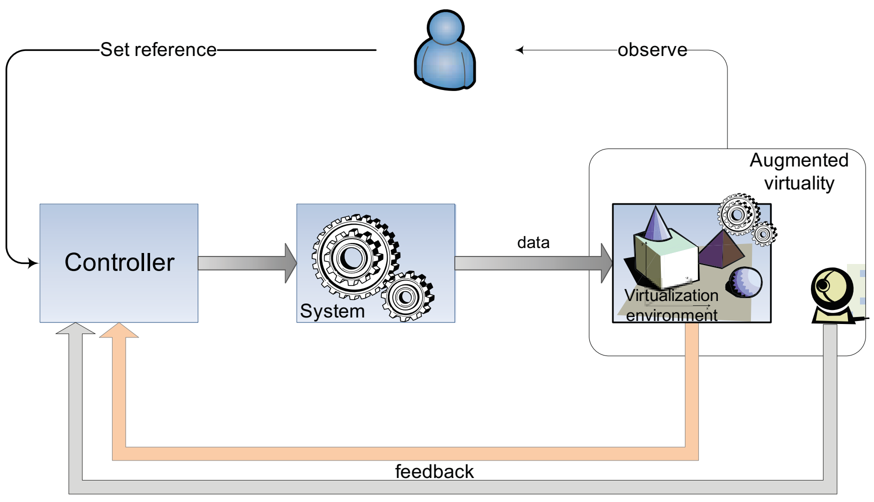
Fig. 5. The VRCS open loop control

5. Closed loop control

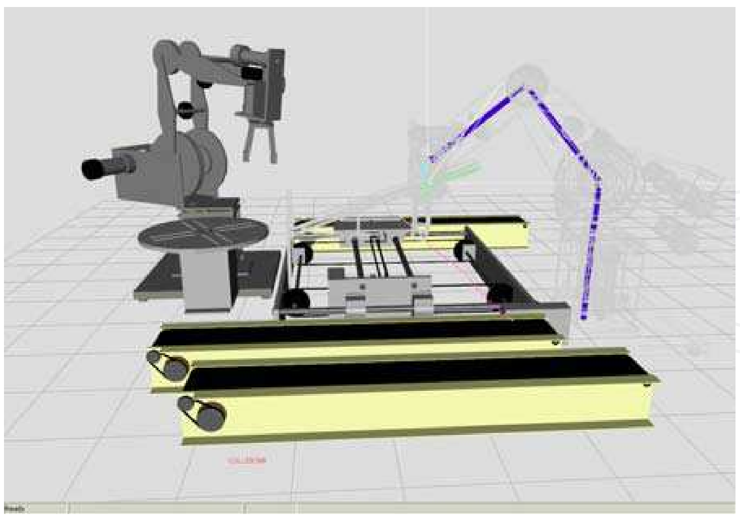
The VRCS closed loop system is presented in Fig. 6(a). At least part of the simulation engine is bundled with the visualization engine, as was the case in the VRCS monitoring and open-loop control. Identically as in those cases, this part of simulation engine, in conjunction with the visualization engine, provides additional valuable information for the control. The improvement over the previous category, VRCS open-loop control, is that here this extra information is used as a feedback, consequently improving better control quality and responsiveness.
The closed loop can be achieved using a virtual sensor or with a help of one or two cameras (stereovision). The former one is considered next, while the later case – the usage of cameras to provide feedback – is explained in great detail in the next section. By using cameras we are thus effectively creating the feedback through augmented virtuality (See Fig. 1).

5.1 Collision avoidance

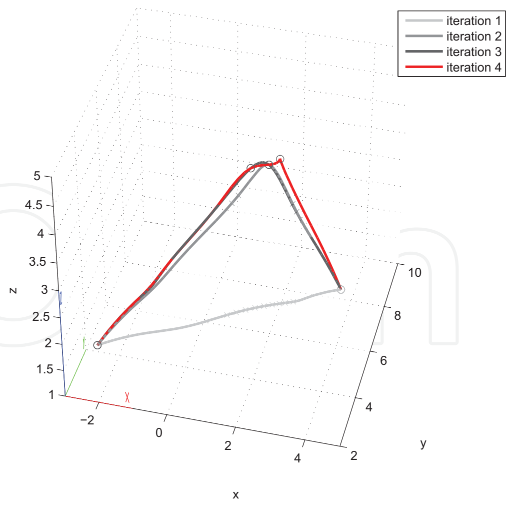
Another illustrative example, shown in Fig. 6(b) with two robotic arms sharing the manufacturing test bed, is the case of collision avoidance by the principle of minimal movement from the possible collision point detected using the virtual collision detection sensors. The sensors are placed along the entire robotic arm's surface thus allowing the exact collision point to be detected and the link, that is in collision, to be determined. Since that particular collision point can be evaded only by moving that link and/or links lower in the hierarchy from the colliding link, the collision evasion algorithm calculates the collision free position of the robotic arm following the principle of minimal movement of those links. The collision evasion continues iteratively until the complete trajectory is collision free (Reichenbach et al. (2006)). The resulting trajectory is shown in Fig 6(c).



(a) Information flow - VRCS closed-loop case



(b) FMS testbed with two robotic arm sharing the same space. One robotic arm is (will be) colliding with the gravitational buffer if the trajectory is not modified.

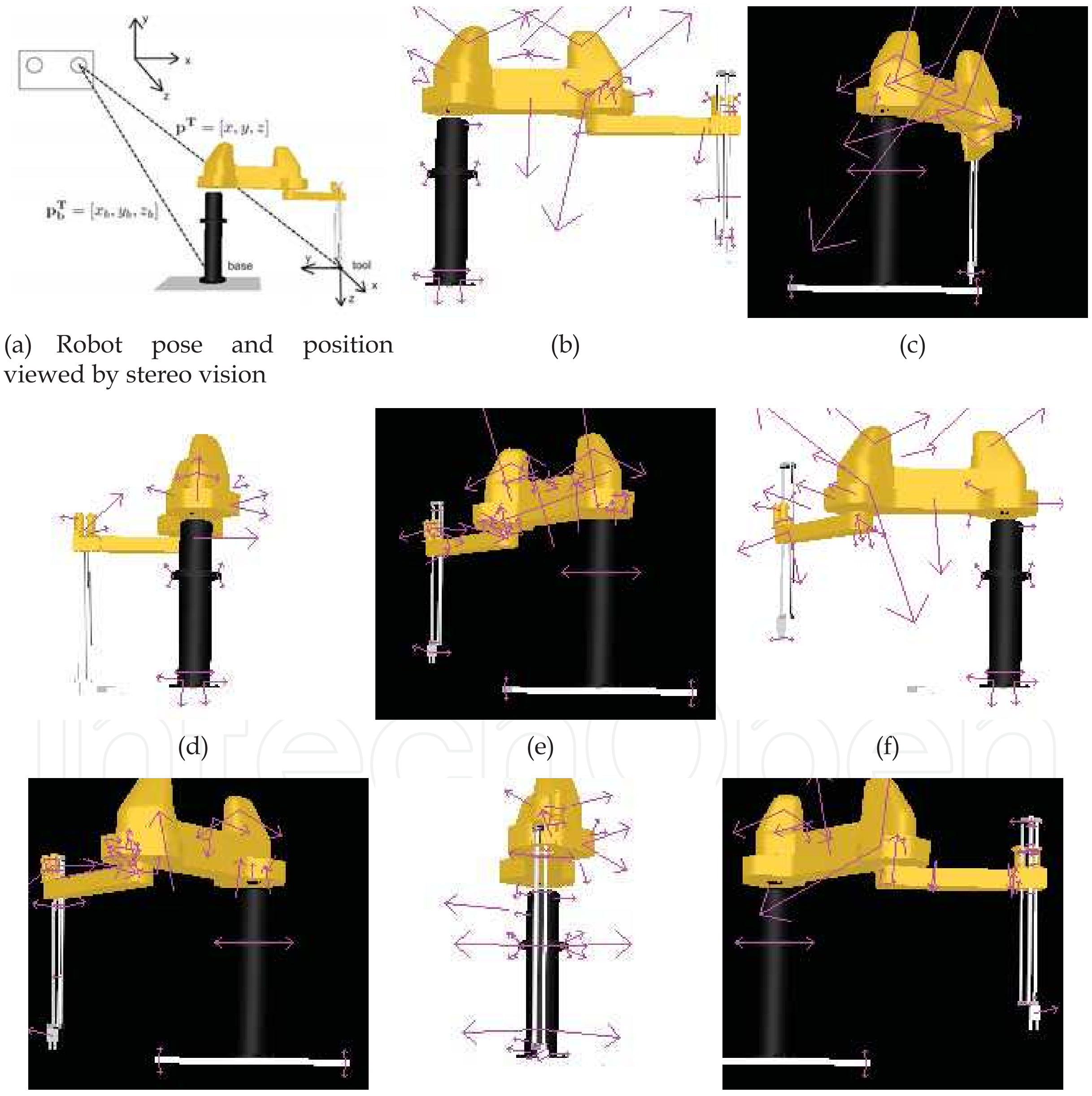


(c) Collision free trajectory and the previous iterations that led to that trajectory

Fig. 6. The VRCS closed loop control - virtual collision sensor

6. Virtual reality feedback

With visual sensors, providing a depth information, it is possible to identify position and pose of every actor in the virtual reality scene without the need for additional sensors. The vision-based approach for robot pose and position identification using virtual models and stereo vision in effort to match robot's representation in a virtual environment with a real one, is explained in detail.
One of the algorithms that extracts and describes local features (called keypoints) on the camera acquired image is a scale invariant feature transform (SIFT) algorithm (Lowe, 2004). Having a known 3D model of a robot in the scene and by using a SIFT algorithm, a database of robot keypoints associated with different images (views) of the robot has had to be created first (see Fig. 7). For finding the pose of the robot, keypoints of the left camera image


y
x
z
$\mathbf{p}^T = [x, y, z]$
$\mathbf{p}_b^T = [x_b, y_b, z_b]$
base
tool
y
x
z

(a) Robot pose and position viewed by stereo vision (b) (c)

(d) (e) (f)

(g) (h) (i)

Fig. 7. Created base of 3-dimensional images and corresponding keypoints

are first produced and then compared to the keypoints in the database. For the image pixels associated with the matched keypoints, the corresponding 3D points are calculated using a disparity map, found by dynamic programming matching algorithm (Cox, 1995), (Forstmann et al., 2004). This is possible only if a pinhole model of cameras is used, and if their intrinsic and extrinsic parameters are known (Davis, 1996). All bad points (positioned too far, too close, etc.) are discarded, while the remaining good ones are used for minimizing the distance to the robot represented with a chain of line segments. Minimization is executed by applying a Nelder-Mead simplex optimization algorithm (Mathews & Fink, 2004). Best optimization results are then compared to the image and the depth map of a 3D robot model, and the configuration with the best match is assumed to be a found pose of the robot.

6.1 Related work

Visual pose detection algorithms have been a subject of intensive research. From the practical point of view, few of them proved to be very effective. For example, in case of using only one camera, one can use the SIFT algorithm (Lowe, 2004) or its faster Speeded-Up Robust Features (SURF) version (Bay et al., 2006), which can extract and describe local features of image necessary to robustly find correct position and orientation of an arbitrary object, but these algorithms are not suitable for objects which change their configuration like robotic manipulators do. In most cases, the aim of using two cameras in robotic systems was to determine a position of robot's end effector, e.g. robot stereo-hand coordination for grasping curved parts (Dufournaud et al., 1998), stereo vision-based feedback (Toyama et al., 1996), aligning the end effector with object (Horaud et al., 1995), and visual servoing for precise manipulation (Jägersand et al., 1997). These approaches can give the exact position of manipulator's end effector, but they do not provide the information about positions of robot links. The method described in (Bischoff & Gerke, 2000) uses stereo vision for matching contours and critical points on both images and implements a fuzzy logic-based collision avoidance algorithm. In (Mulligan et al., 2000) the image obtained from a camera is being compared to the image from the scene model, and the difference is used for adjusting that model. In (Lawrence et al., 1989) joint angles are determined from a single image based on the positions of markers attached to robot's links. Similar problems are found in the human body pose identification. (Bernier, 2006) describes the statistical model for fast 3D articulated body tracking, (Shakhnarovich et al., 2003) shows the algorithm for the fast human body pose estimation using large base of example images. The work described in (Rehg & Knade, 1994) explains model-based hand tracking system, that can recover 27 DOF hand model from grayscale images.

6.2 Problem definition

Determining the pose and position of a robot by processing a stereo image requires a source of good quality video input. This is provided in the considered system by a pair of calibrated cameras located sideways to the robot and directed towards the robot. In this way, the installed stereo-vision system is able to generate fairly accurate 3D information about the elements in the scene being viewed (including a robot itself). As already mentioned, positions of elements can be obtained by producing a disparity map between the images acquired from two cameras. For this purpose, a dynamic programming stereo vision algorithm similar to one described in (Cox, 1995) and (Forstmann et al., 2004) is used.

In order to recognize the pose and position of a robot in the image, it is necessary to have its 3D model prepared and values of robot's kinematics parameters defined by a standard Denavit-Hartenberg method (Schilling, 1990).
As shown in Fig. 7(a), a position of a robot is defined as the location of a robot base $\mathbf{p_b}$ with respect to a left camera position:

$$\mathbf{p_b} = \begin{bmatrix} x_b & y_b & z_b \end{bmatrix}^T \tag{1}$$

where x_b is the horizontal axis, y_b is the vertical axis and z_b is the axis pointing from the camera toward the object. In case that the robot position vector $\mathbf{p_b}$ is known the problem is reduced only to recognition of robot's pose.
Position of robot tool is defined as its location with respect to a left camera position:

$$\mathbf{p} = \begin{bmatrix} x & y & z \end{bmatrix}^T \tag{2}$$

The image obtained from the left camera, together with the corresponding depth map, is used to find the appropriate pose of a robot expressed in the joint space by finding joint variables:

$$\mathbf{q} = \begin{bmatrix} q_1 & \cdots & q_n \end{bmatrix}^T \tag{3}$$

where q_i is the i-th joint variable, and n denotes a number of degrees of freedom.

6.3 Creating a database of keypoints

A virtual 3D model of a robot is used for creation of keypoints that are going to be stored in the database. The applied procedure is the following: a 3D robot model is rotated consecutively by the angle of 45°, while simultaneously joint variables are changed by one fourth of the maximal robot joint value, which results in 8 images. In the same time the color of the background is alternating from black to white and vice-versa. The same procedure is repeated four times, first for the images of 350x350 resolution, and then for images of respective height and width reduced by 20% (the reasons for varying the image size are elaborated in (Liebelt et al., 2008)). The database of keypoints is created and stored for further use for all 32 created images. Fig. 7 shows the keypoints corresponding to original 350x350 images[5].

6.4 Searching for robot keypoints

As mentioned above, keypoints extracted from the left camera image are compared to key-points stored in the image database. In order to make correct decision whether a given key-point T_i point is valid or not, its respective nearest and second nearest neighbors A and B should be found. Points A and B are those keypoints in the same image that have the shortest Euclidean distances of their descriptors. A point T_i is discarded if its nearest and second nearest neighbors satisfy the following relation (Lowe, 2004):

$$D(A,T_i) > \alpha \cdot D(B,T_i) \tag{4}$$

[5] It should be noted that a larger number of images being created will slow down the execution of the algorithm, while a smaller number of images will decrease the accuracy.

where $D(x,y)$ is the Euclidean distance of descriptors for two keypoints x and y and α is a distance ratio set to value 0.8 as recommended in (Lowe, 2004).
Regarding the remaining keypoints, the space coordinates are calculated with respect to the coordinate system of a left camera, using the pixel in the left camera image and the corresponding disparity map obtained by previously mentioned dynamic programming algorithm for stereo vision. In the studied system, the keypoints lying 10 m or more from the camera are automatically discarded, because for a robot that far away, the distance from the camera would be too long for reliable detection, or the number of detected keypoints would be too small for accurate pose identification.
A mean point is calculated taking into account all remaining points, and all points which are too far from the mean point are discarded. In our case, it is assumed that a point is too far from a mean point if it is located outside the area 60% larger than the maximum span L_{max} of a robot. In general, L_{max} is a distance from two outmost distinct points of the robot in any configuration.

6.5 Optimization of distance from remaining keypoints

In order to find the pose and position of a robot, a method for matching keypoints to the robot is developed. In this method robot is represented as a chain of line segments, whose points are calculated from a homogeneous transformation matrix. A homogeneous transformation matrix calculated using known robot kinematics parameters and joint variable values has the following form (Schilling, 1990):

$$\mathbf{T}_0^k = \begin{bmatrix} \mathbf{R} & \mathbf{p} \\ \boldsymbol{\eta}^T & \sigma \end{bmatrix} \tag{5}$$

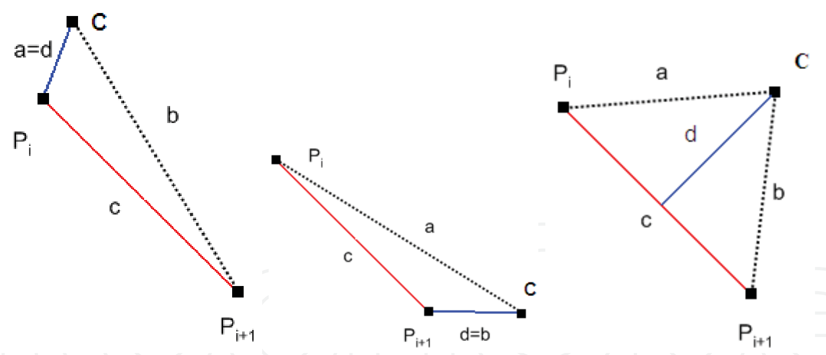
where $\mathbf{R}$ is a rotation matrix, $\mathbf{p}$ is a translation vector, $\boldsymbol{\eta}$ is a perspective vector (usually null vector) and σ is a scaling factor (usually 1).
Using (5), it is possible to determine the points of a chain that form line segments representing the robot:

$$\begin{aligned} P_{2k-1} &= P_{2k-2} + l_k \cdot \mathbf{z}_{k-1} \\ P_{2k} &= \mathbf{p}_k \end{aligned} \tag{6}$$

where P_i is the i-th point in the chain, l_k is the robot kinematics parameter representing a length of the k-th link, $\mathbf{p}_k$ is the translation vector, and $\mathbf{z}_{k-1}$ is the vector representing direction of ($k-1$)-st robot link, which is the 3-rd column of rotation matrix $\mathbf{R}_{k-1}$.
For a given keypoint, its distance to the nearest line segment of the robot can be calculated as follows (see Fig. 8):

1. if $b^2 > a^2 + c^2$ then $d = a$;
2. if $a^2 > b^2 + c^2$ then $d = b$;
3. else $d = \sqrt{b^2 - (\frac{b^2+c^2-a^2}{2c})^2}$

where a is the length from the beginning of the line segment (point P_i) to the point(C), b is the length from the end of the line segment (point P_{i+1}) to the point (C), c is the length of a line segment and d is a wanted distance.
The distance from a point to a chain of line segments is calculated by finding a minimal distance from that point to every line segment in the chain.



(a) $b^2 > a^2 + c^2$ (b) $a^2 > b^2 + c^2$ (c) $b^2 \leq a^2 + c^2$ and $a^2 \leq b^2 + c^2$

Fig. 8. Calculation of distance from line segment to point

It is necessary to find the pose and position of robot for which the sum of distances from points which are not discarded to a chain of line segments is minimal. This is done by using the Nelder-Mead simplex optimization method, which is performed for various initial conditions. Parameters involved in optimization are a position vector $\mathbf{p_b}$ and a vector of joint variables $\mathbf{q}$. Initial conditions for the position vector are:

1. (x_s, y_s, z)
2. (x_{max}, y_s, z)
3. (x_{min}, y_s, z)
4. (x_s, y_{max}, z)
5. (x_s, y_{min}, z)

where x_s, x_{max} and x_{min} are respective mean value, maximal value and minimal value of x coordinates of all points, y_s, y_{max} and y_{min} are respective mean value, maximal value and minimal value of y coordinates of all points, and z is a minimal value of z coordinates (it is assumed that the robot base is located in the bottom, which is the case for the studied robot). Initial conditions for joint variables are a combination of first three joints in which, working area of each joint is divided in 8 parts. In this way, there are 2560 initial conditions. For every initial condition 100 iterations of optimization are performed. First 30 iterations are used only to optimize the position vector of robot base, because first it is necessary to set an approximate position of the chain of line segments. Remaining 70 iterations of optimization are performed for optimizing the position vector of robot base together with the vector of joint variables.

6.6 Choosing the best pose and position

Twenty best results obtained by optimization from different initial conditions are then used to create the image of a 3D robot model. For each set of parameters, 3D model is set to correspond to an actual situation (the base of the robot is set to given coordinates with respect to the virtual camera and joint values are set to a given values), virtual camera is set to have identical parameters as the real one (which can be determined by calibration). The comparison is done only on the image segments where robot model is located. There are three criteria of comparison.

The first criterion compares RGB intensities of the images:

$$\begin{aligned}
\Delta R &= I_{R1}^{i} - I_{R2}^{i} - I_{comp} \\
\Delta G &= I_{G1}^{i} - I_{G2}^{i} - I_{comp} \\
\Delta B &= I_{B1}^{i} - I_{B2}^{i} - I_{comp} \\
K_1 &= \frac{\sum_{i=1}^{n} (\Delta R)^2 + (\Delta G)^2 + (\Delta B)^2}{n}
\end{aligned} \tag{7}$$

where K_1 is a value of the first criterion; n is a number of points being compared; I_{R1}^{i}, I_{G1}^{i} and I_{B1}^{i} are intensities of red, green and blue channels of the image from a virtual camera; I_{R2}^{i}, I_{G2}^{i} and I_{B2}^{i} are intensities of red, green and blue channel of the image from a real camera and I_{comp} is a mean value of intensity differences between virtual and real camera image segments where robot is located, used to compensate different intensities between images. The second criterion is related to the absolute sum of differences of z coordinates:

$$K_2 = \frac{\sum_{i=1}^{m} \left| z_1^{(i)} - z_2^{(i)} \right|}{m} \tag{8}$$

where K_2 is a value of the second criterion; $z_1^{(i)}$ is the i-th value of z coordinate obtained from the virtual image; $z_2^{(i)}$ is the i-th value of z coordinate obtained from a disparity map of real images and m is a number of points belonging to image segments where robot model is located, for which holds $| z_1^{(i)} - z_2^{(i)} | < L_{max}/2$ (it is necessary to discard points with distant z coordinates, because of possible errors in the disparity map).
The third criterion has the following form:

$$K_3 = \frac{n-m}{n} \tag{9}$$

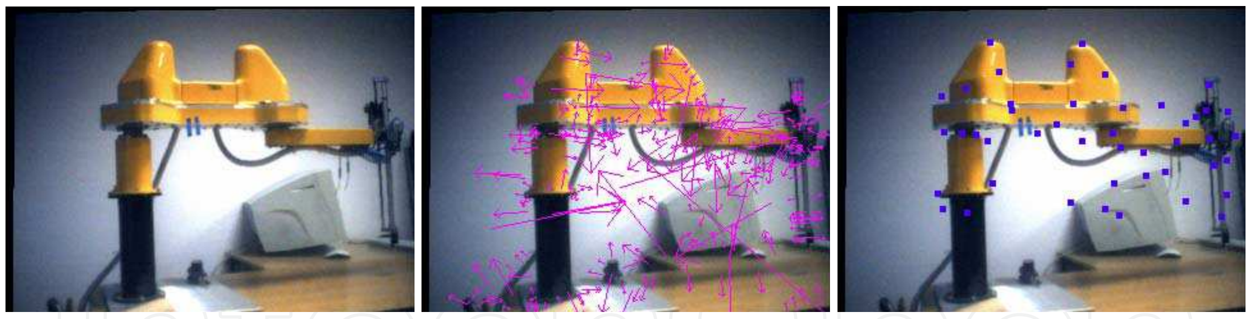
where K_3 is a value of the third criterion. This criterion shows the ratio between the number of discarded points and the overall number of points being compared.
An overall criterion for comparison is obtained by multiplication of criteria obtained from (7), (8) and (9):

$$K = K_1 \cdot K_2 \cdot K_3 \tag{10}$$

The entire procedure for finding a pose and position of the robot is illustrated in Fig. 9.

6.7 Results

The presented algorithm has been tested on a 4-DOF SCARA robot configuration. For this purpose two BTC PC380 USB web cameras anchored parallel at the distance of 10 cm were used. Prior to experiments, exact intrinsic and extrinsic parameters of cameras were obtained by calibration. During experiments, the original 640x480 image size was reduced to the 320x240 size and the images were rectified prior to starting a dynamic programming algorithm. Image size reduction is done to decrease execution time, but the cost is decreasing accuracy of an algorithm.
Experiments were conducted using two different camera positions with respect to the robot base, as shown in Table 1. Camera positions must be far enough to let the robot fit into the image frame, but still close enough for good quality of robot pose and position recognition.


(a) Image from camera (b) Found SIFT keypoints (c) Remaining keypoints

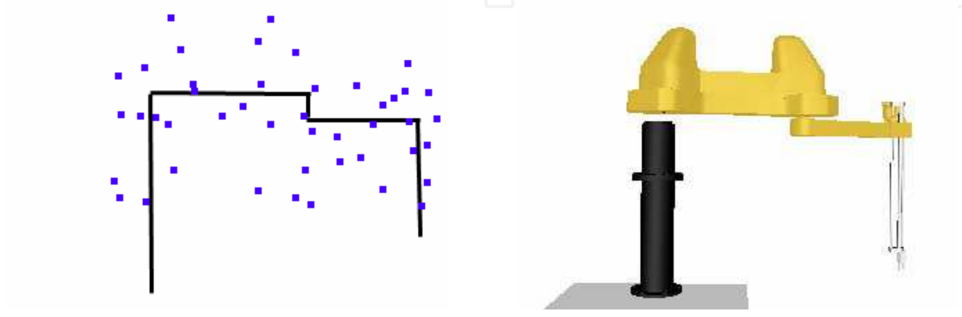

(d) Chain of line segments (e) Recognized robot pose

Fig. 9. Searching for robot pose and position

x_1 [cm]	y_1 [cm]	z_1 [cm]	x_2 [cm]	y_2 [cm]	z_2 [cm]
-22	209	-41	-41	166	-40

Table 1. Two positions of robot base w.r.t. the left camera

The proposed algorithm was executed five times for 40 different robot poses, which makes a total number of 400 algorithm runs. The execution times of the algorithm were in the interval of 3-5 seconds depending on a number of keypoints found and a number of points being discarded. Using the results of the algorithm (position of robot base and values of joint variables) and by calculating direct kinematics of the robot, the assumed position of a robot's end effector has been obtained and compared to the real position of robot's end effector. The Euclidean distance between them has been taken as a direct quality measure of the result. All results having that distance greater than 40 cm were treated as inaccurate and accordingly, discarded.

The experiments have shown that 27% of the results were discarded in the first camera position, and 33% in the second one. In other words, approximately seven out of ten solutions are satisfactory. This means that good solutions prevail. But this also means that dissatisfactory solutions should be further processed to ensure constantly reliable solutions. This could be done by using some filter (e.g. a median filter) and appropriate interpolation algorithm related to prevailing good solutions.

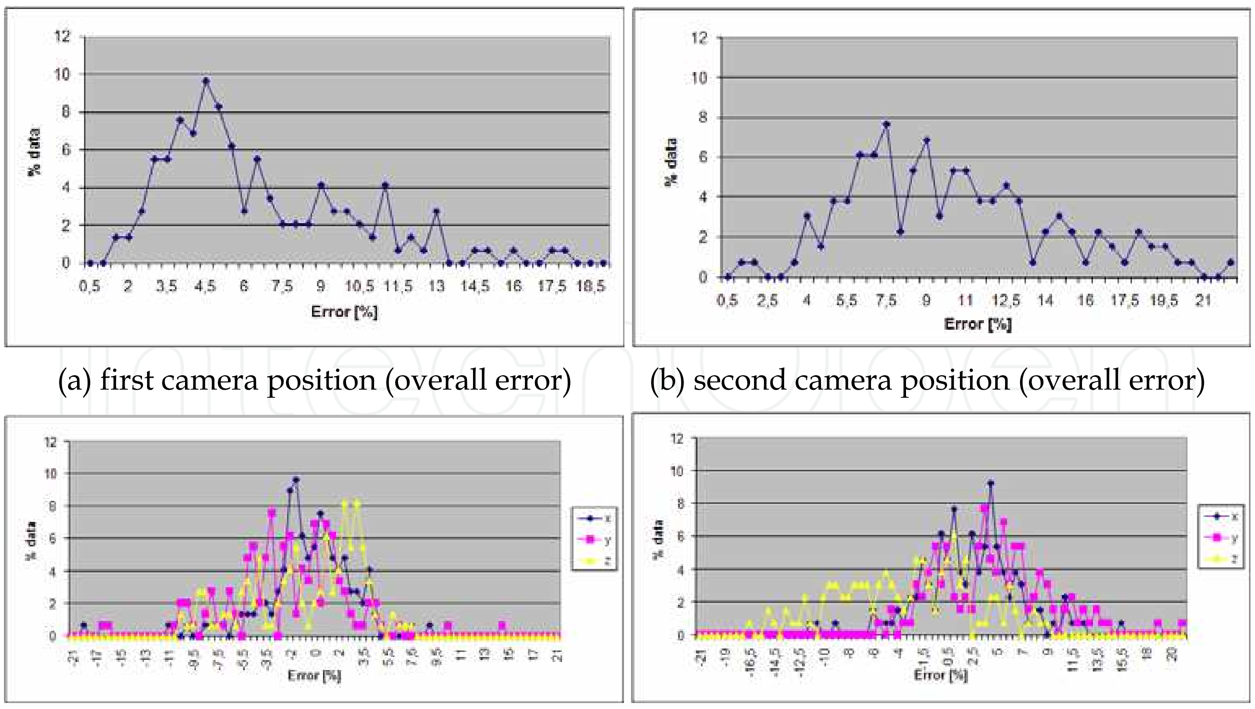
Only the satisfactory results have been used for a statistical analysis of experiments. Table 2 shows the mean value of the distance error and the standard deviation for two respective camera positions in direction of x, y, and z coordinates, as well as the overall Euclidean distance error mentioned above. Table 3 shows the mean value and the standard deviation of robot joint variables estimates error and the position of the robot base estimates error. It is

Parameter	Mean [cm]	Deviation [cm]
x_1	-1.3	7.3
y_1	-4.3	9.3
z_1	-1.29	9.13
$\sqrt{x_1^2 + y_1^2 + z_1^2}$	13.9	7.35
x_2	4.57	8.16
y_2	8.45	8.78
z_2	-5.41	10.78
$\sqrt{x_2^2 + y_2^2 + z_2^2}$	17.8	9.08

Table 2. Mean values and deviation of the robot tool position estimates error

	First camera pos.		Second camera pos.	
Parameter	Mean	Deviation	Mean	Deviation
q_1	−5.07°	22.1°	−2.62°	13.3°
q_2	−0.33°	21.8°	−9.3°	25.8°
q_3	0.98 cm	9.28 cm	-0.62 cm	10.4 cm
x_b	-2.03 cm	3.8 cm	-6.26 cm	3.9 cm
y_b	-8.69 cm	3.9 cm	4.32 cm	3.9 cm
z_b	0.07 cm	2.9 cm	-6.07 cm	2.4 cm

Table 3. Mean values and deviation of joint variables estimates error and the position of the robot base estimates error



(a) first camera position (overall error) (b) second camera position (overall error)

(c) first camera position (individual error) (d) second camera position (individual error)

Fig. 10. Percentage of data related to overall error and to the individual error in x, y and z direction

interesting to see the distribution of results with respect to the magnitudes of Euclidean distance errors expressed relative to $\|\mathbf{p_b}\|$. Figs. 10(a) and 10(b) show the distribution of percentage of acceptable solutions having the same distance error. In the same manner, Figs. 10(c) and 10(d) show the distribution of percentage of solutions having the same distance errors along x, y and z coordinates expressed relative to $\|\mathbf{p_b}\|$. From those figures one can see that the distribution assumes the form similar to gaussian one with a center close to zero.

7. Conclusion

Technological advances in computing power and engineering and consequently available functionality enabled the implementation of new methodologies in control systems. The integration of computer graphics and control systems, described in this chapter, allowed for the new control system paradigm named Virtual Reality Control Systems (VRCS). The classification of the VRCS into the following categories, the VRCS visualization, the VRCS monitoring, the VRCS open-loop control and the VRCS closed-loop control, has been established and explained. Furthermore, for each VRCS category an illustrative example and its practical usage has been elaborated. The Flexible manufacturing testbed application in VRML is presented as a typical example of VRCS visualization. Virtual sensor – a humanoid robot foot pressure sensor – was introduced as the VRCS monitoring usage case. Subsequently, the Open loop VRCS control system was explained and finally a detailed consideration was given to the Closed loop VRCS case and how the visual feedback from the virtual system is established.

Having a known 3D model of a robot in the scene and by using a scale invariant feature transform (SIFT) algorithm, a database of robot keypoints associated with different images (views) of the robot has been created. By comparison of keypoints of an actual robot with the keypoints from the database, the position and pose of the robot are determined by a Nelder-Mead simplex optimization. For this purpose, a three-elements comparison criterion has been defined and best suitable parameters were determined. The comparison of the so obtained pose of the considered 4 DOF SCARA robot with the pose being directly measured has indicated that the proposed method has a precision comparable to the precision human beings have. The transfer of the estimated robot pose into the virtual model of the robot system enables use of VR feedback.

It is important to note that the VRCS principles and the methods described here are indifferent to the quality and detail of 3D models and virtual environments. The principles can be applied to various sophistication levels of simulation and virtual environment. The control quality, however, is related to both simulation and virtual environment refinements. Viceversa, the refinements can be tailored according to the quality desired, or achievable performance limitations.

8. References

Bay, H., Tuytelaars, T. & Gool, L. V. (2006). Surf: Speeded up robust features, *ECCV*, pp. 404– 417.

Bernier, O. (2006). Real-time 3d articulated pose tracking using particle filters interacting through belief propagation, *18th International Conference on Pattern Recognition (ICPR'06)*, Vol. 1, pp. 90–93.

Bischoff, A. & Gerke, M. (2000). Fuzzy collision avoidance using stereo-vision, *SYROCO*, Vol. 1.

Cox, I. J. (1995). A maximum likelihood n-camera stereo algorithm, *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 733–739.

Davis, E. R. (1996). *Machine Vision: Theory, Algorithms, Practicalities 2nd Edition*, Academic Press.

Dufournaud, Y., Horaud, R. & Quan, L. (1998). Robot stereo-hand coordination for grasping curved parts, *Proceedings of the Brittish Machine Vision Conference, BMVC'98*, Springer- Verlag, pp. 760–769.

Forstmann, S., Kanou, Y., Ohya, J., Thuering, S. & Schmitt, A. (2004). Real-time stereo by using dynamic programming, *Computer Vision and Pattern Recognition Workshop*, pp. 29–29.

Goldman, R. (1990). Matrices and transformations, *in* A. S. Glassner (ed.), *Graphics Gems*, Academic Press, pp. 472–475.

Horaud, R., Dornaika, F. & Espiau, B. (1995). Visually guided object grasping, *IEEE Transactions on Robotics and Automation* 14: 525–532.

Jägersand, M., Fuentes, O. & Nelson, R. (1997). Experimental evaluation of uncalibrated visual servoing for precision manipulation, *IEEE International Conference on Robotics and Automation*.

Kovacic, Z., Smolic-Rocak, N., Dujmovic, S. & Munk, R. (2007). Matlab real-time workshop-based internet accessible robot control laboratory, *The Proceedings of the International Conference on Remote Engineering and Virtual Instrumentation REV07* .

Kovačić, Z., Bogdan, S., Reichenbach, T., Smolić-Ročak, N. & Punčec, M. (2001). Flexman - a computer-integrated tool for design and simulation of flexible manufacturing systems, *CD-ROM Proceedings of the 9th Mediterranean Conference on Control and Automation Control*, Vol. TM2-B.

Lawrence, P. D., Mackworth, A. K. & Mulligan, I. J. (1989). Patent us4826391 manipulator arm position sensing.

Liebelt, J., Schmid, C. & Schertler, K. (2008). Viewpoint-independent object class detection using 3d feature maps, *Computer Vision and Pattern Recognition*, pp. 1–8.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints, *International Journal of Computer Vision* 60: 91–110.

Mathews, J. H. & Fink, K. K. (2004). *Numerical Methods Using Matlab, 4th Edition*, Prentice Hall Inc.

MathWorks (2010). Simulink 3d animation, http://www.mathworks.com/products/3d animation/.

Milgram, P. & Colquhoun, H. (1999). A taxonomy of real and virtual world display integration.

Möller, T. & Haines, E. (1999). *Real-Time Rendering*, A. K. Peters. MÖL t 02:1 1.Ex.

Mulligan, I. J., Mackworth, A. K. & Lawrence, P. D. (2000). A model-based vision system for manipulator position sensing, *Workshop on Interpretation of 3-D Scenes*, pp. 186–193.

Osborne, H. (2010). The arts: Fine art, contemporary art and music - perspective, http://arts.jrank.org/pages/16368/perspective.html.

Rehg, J. & Knade, T. (1994). Visual tracking of high dof articulated structures: an application to human hand tracking, *European Conference on Computer Vision*, pp. 35–46.

Reichenbach, T. (2009). A dynamic simulator for humanoid robots, *Artificial Life and Robotics* 13(2): 561–565.

Reichenbach, T., Miklic, D. & Kovacic, Z. (2006). Supervisory control by using active virtual 3d models in-the-loop, *CCA '06. IEEE International Conference on Control Applications, Munich, Germany* pp. 1409–1413.

Schilling, R. J. (1990). *Fundamentals of robotics*, Prentice Hall, New Jersey.

Shakhnarovich, G., Viola, P. & Darrell, T. (2003). Fast pose estimation with parameter-sensitive hashing, *Ninth IEEE International Conference on Computer Vision (ICCV'03)*, Vol. 2, p. 750.

Smolić-Ročak, N., Bogdan, S., Kovačić, Z., Reichenbach, T. & Birgmajer, B. (2002). Modeling and simulation of fms dynamics by using vrml, *CD-ROM Proceedings of the b'02 IFAC World Congress*, Vol. WM2-C.

Talbott, S. (1995). *The Future Does Not Compute: Transcending the Machines in Our Midst,* 1 edn, O'Reilly & Associates, Inc.

Toyama, K., Hager, G. D. &Wang, J. (1996). Servomatic: A modular system for robust positioning using stereo visual servoing, *Proc. IEEE Int'l Conf. Robot. and Automat*, pp. 2636– 2643.

Verth, J. M. V. & Bishop, L. M. (2004). *Essential Mathematics for Games and Interactive Applications: A Programmer's Guide*, The Morgan Kaufmann Series in Interactive 3d Technology, Morgan Kaufmann.

Walnum, C. (2003). *Direct3D Programming*, SAMS.

Wikipedia (2010). Visualization (computer graphics), http://en.wikipedia.org/wiki/Visualizatio