Robotics and Computer-Integrated Manufacturing 28 (2012) 132–146

Contents lists available at SciVerse ScienceDirect

Robotics and Computer-Integrated Manufacturing

journal homepage: www.elsevier.com/locate/rcim

Obstacle avoidance of redundant manipulators using neural networks based reinforcement learning

Mihai Duguleana [a,*], Florin Grigore Barbuceanu [a], Ahmed Teirelbar [b], Gheorghe Mogan [a]

[a] Department of Product Design and Robotics, University Transilvania of Brasov, Brasov 500036, Romania
[b] Faculty of Engineering, University of Alexandria, Alexandria 21544, Egypt

ARTICLE INFO

Article history:
Received 17 November 2010
Received in revised form
29 June 2011
Accepted 15 July 2011
Available online 7 September 2011

Keywords:
Obstacle avoidance
Redundant manipulators
Neural networks
Q-learning
Virtual reality
CAVE

ABSTRACT

This paper proposes a new approach for solving the problem of obstacle avoidance during manipulation tasks performed by redundant manipulators. The developed solution is based on a double neural network that uses Q-learning reinforcement technique. Q-learning has been applied in robotics for attaining obstacle free navigation or computing path planning problems. Most studies solve inverse kinematics and obstacle avoidance problems using variations of the classical Jacobian matrix approach, or by minimizing redundancy resolution of manipulators operating in known environments. Researchers who tried to use neural networks for solving inverse kinematics often dealt with only one obstacle present in the working field. This paper focuses on calculating inverse kinematics and obstacle avoidance for complex unknown environments, with multiple obstacles in the working field. Q-learning is used together with neural networks in order to plan and execute arm movements at each time instant. The algorithm developed for general redundant kinematic link chains has been tested on the particular case of PowerCube manipulator. Before implementing the solution on the real robot, the simulation was integrated in an immersive virtual environment for better movement analysis and safer testing. The study results show that the proposed approach has a good average speed and a satisfying target reaching success rate.

1. Introduction

Currently, one of the most important goals of researchers in robotics is achieving a natural interaction between humans and robots, a satisfying Human–Robot Interaction (HRI). In order to attain this, a lot of effort is put into designing and developing "intelligent machines". A fundamental interaction between humans and robot is the transfer of objects, most of the times seen as part of a more useful activity [1]. Directly handing objects to robots is a basic yet, highly complex scenario, as several issues need to be solved: determining goal position within a previously chosen Cartesian system, solving the kinematics problem, solving the path planning problem, avoiding environmental obstacles (whether they are static or dynamic), considering safety prerequisites, handling hardware/software problems and many others.

Kinematics problem is one of the most discussed issues in fields like robotics and computer animation. There are 2 different types of kinematics: Forward Kinematics (FK) and Inverse Kinematics (IK). FK explicitly solves the position and orientation of each segment in Cartesian coordinates at a specific time, using joint angles as input

$$F_k(\phi) = (X;Y;Z)_i \tag{1}$$

In Eq. (1), given the angle vector ϕ and a kinematic chain with n joints, F_k function provides the Cartesian coordinates $(X;Y;Z)$ for each i joint, where $i \in [1,n]$. For most robotic structures, the main goal of FK is to determine the position and orientation of the last link (the end-effector—EE). Opposite, IK provides direct control over the position of the EE by computing each joint angle from the kinematic chain

$$I_k(X;Y;Z)_i = F_k{}^{-1}(X;Y;Z)_i = \phi \tag{2}$$

Although solving IK is an important issue in robot manipulation, the problem of path planning also needs to be handled. The path planning problem contains the following key issues: a space of states (initial, intermediate and final states), the time, the operators/actions, a planner (algorithm) and a plan. Several path planning studies for manipulators have been conducted over the years, each proposing different manipulation strategies [2,3].

As robots are basically metal structures equipped with sensors from which they get environmental data, what matters is to

* Corresponding author at: Str. Universității nr. 1, Brașov 500068, Romania. Tel.: +40 726385701; fax: +40 268 418967.
E-mail addresses: mihai.duguleana@unitbv.ro (M. Duguleana), florin.barbuceanu@unitbv.ro (F.G. Barbuceanu), shuja.consulting@gmail.com (A. Teirelbar), mogan@unitbv.ro (G. Mogan).

0736-5845/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.rcim.2011.07.004

compute this data fast enough, in order to give the ability to handle tasks, most of the times, in unstructured environments. This study focuses on designing a manipulation experiment in such environments, where obstacle avoidance is needed. Obstacle avoidance in manipulation tasks is a much disputed field of robotics. Classical ways of dealing with obstacle avoidance include straight-forward mathematical apparatus. Obstacle avoidance is in many cases handled using the pseudo-inverse Jacobian [4,5]. Given that a multi-layer neural network is able form any continuous nonlinear mapping from one domain to another [6], this relatively new technology has been combined with *Q*-learning for solving the problem of obstacle avoidance in manipulation tasks [7,8].

Q-Learning is a specific algorithm which basically implements reinforcement learning technique [9]. For the case of a robotic manipulator, the quality of a state-action combination is defined as a cost function

$$Q : S \times A \rightarrow \Re \tag{3}$$

where Q is the solution set, S is the set of states and A is the set of actions.

The reward is given if the current trajectory is collision free and if the EE approaches the goal. The strong point of this approach is that the algorithm deals with the lack of knowledge regarding the working environment by using each sequence of state, action and resulting state, as a sample of the unknown underlying probability distribution. This information is used to form a reward parameter that helps in achieving the correct values. Due the fact that a large number of tests is required to converge, this algorithm may be wrapped on a feed-forward neural network. Only a few studies address this type of solution for solving IK problem [10,11].

Using real physical structures within experiments implies harder and more time consuming testing activities, mostly due safety considerations. Thus, this study proposes the usage of immersive virtual reality (VR) environments for easier testing and analysis of the developed IK algorithm. After satisfying results are achieved within the simulated environment, the algorithm is further tested on real robots.

2. Literature overview

This paper integrates aspects from several research fields. Thus, it is proposed below a resume of current achievements in studies related to the problem.

2.1. Obstacle avoidance during robot manipulation

Obstacle avoidance is an issue usually addressed to redundant manipulators. If a link structure can reach the same EE position by several different intermediate positions of the kinematic bodies from the chain, it is called redundant. Redundancy can be used to meet secondary tasks such as obstacle avoidance, singularity avoidance [12] or variable optimization (i.e. torque optimization [13], dexterity optimization [14], energy saving optimization and others).

One of the most used approaches to solving IK and obstacle avoidance is based on the use of inverse of Jacobian and its relatives [4,5,15,16]. The main idea that stands behind this approach is that a Jacobian matrix contains all the information necessary to relate a change in any component of EE position vector to a change in any other component of the configuration angle vector. The approximation which is usually made for the angle vector variation $\Delta\phi$ is

$$\Delta\phi = J^{-1}\Delta EE \tag{4}$$

where ΔEE is the desired incremental change of the EE position and J is the Jacobian matrix.

After imposing Eq. (4), current algorithms pick a step and iterate the equation on a loop until the EE reaches the goal or a position which is fairly satisfying near the goal. Variations on this technique rely on constructing J^{-1}; most used are the pseudo-inverse $(J^{+})^{-1}$ and the transposed $(J^{T})^{-1}$. Other studies propose a solution that requires only standard inversion instead of pseudo-inversion, by using the extended Jacobian method [17]. After the minimum norm and the homogenous solution are defined, a secondary goal can be translated mathematically [16]. However, this approach loses substance when dealing with local minima and highly irregularly shaped obstacles [18]. To remedy this, some studies propose using task priority strategies, but these are less suitable for manipulators with a high number of degrees of freedom (DOF) [12].

Other methods related to the objective of this study, also used in robot navigation, are the artificial potential field algorithm [19] and fuzzy logic [20]. Obstacles are viewed as reflecting magnetic poles while targets are viewed as attractive points. Artificial potential field algorithm is also suffering from local minima inconsistency, and because of the issues imposed by arbitrarily shaped obstacles and manipulators with multiple links, only a few studies address this approach [21,22].

One of the best approaches to the problem of manipulation in environments with obstacles is the one that uses the quadratic problem to minimize the redundancy resolution of the kinematic chain, with subject to bound constrains [18,23–26].

2.2. Solving IK and obstacle avoidance with artificial neural networks

Connectionist methods such as recurrent neural networks provide an alternative solution to IK problem. Solving the problem of obstacle avoidance using neural networks has been addressed in literature by several scientists.

One of the most basic approaches to robot arm control with neural networks uses the base to EE transformation matrix elements as neuron inputs, a hidden layer with 20 neurons and an output layer with 6 variables, which are the angles between the kinematic links [27]. However, this approach reveals high variable errors, unacceptable for real world use. Another study addresses the problem of finding an obstacle free position with the help of a neural network, given the desired EE position [28]. However, this study does not focus on achieving a trajectory free of obstacles. More elaborated studies propose using a penalty function corroborated with the ball covering technique for solving the problem of collisions [29]. In order to solve the task of obstacle avoidance, Tang et al. propose the usage of a recurrent neural network called the Lagrangian network. However, their study assumes that the entire working environment (together with EE movement trajectories) is known [10]. In some studies, neural networks are used to regulate the impedance parameters of manipulators' EE and avoid obstacles by "sensing" the contact with the working environment [30].

One of the most interesting cases of usage of neural networks is presented in the studies of Zhang et. all, focusing on solving obstacle avoidance with the help of the quadratic problem. Their latest research is based on a dual neural network design (called primal-dual neural network—PDNN) which is able to converge to optimal solutions based on linear variational inequalities (LVI) approach, basically expressing the minimum-energy redundancy resolution of the kinematic system [23,24]. However, the obstacles are modelled as points and the working environment is assumed to be known at any moment of computation.

Genetic algorithms are also used in collaboration with neural networks for evolving obstacle avoidance behaviour for robotic arms [31]. However, genetic algorithms are rather slow and have a "close to optimal" behaviour; there is no guarantee of their convergence and on the quality of the behaviour found [32].

Hasan et. al. propose an adaptive learning strategy using an artificial neural network for controlling the motion of a 6 DOF manipulator, without explicitly specifying the kinematic configuration [33] nor the configuration of the working space. In a recent study [11], Kohonen Self-Organizing Map (KSOM) is used to build a hint generator to solve the inverse kinematic problem of a redundant manipulator. A feed-forward network is then used to learn the obstacle free kinematic map of the robotic arm.

2.3. Robotics and VR technologies

Working with robots assumes the existence of several important factors: space, time, equipment and determination. While the last one can be reasonably compensated by scientists, the others require resources which may not be at the disposal of many robotic laboratories and industrial facilities out there. Generating simulations of robot behaviour in VR might be the answer to this problem. So far, simulation has been used with success in education (developing learning platforms for students [34], commanding robots within virtual laboratories [35] or reconstructing historical sites [36]), industry (manufacturing applications [37]) and entertainment [38]. Scientists have analysed the way in which users perceive virtual objects in immersive environments (measuring basic values like perception of distances and sizes [39], quantifying the immersion quality [40] or assessing more complex attributes like the perception of idle times of virtual industrial robots [41]), interact with virtual objects or team up to solve some specific scenarios [42].

Modelling robots in VR solves hardware troubleshooting problems. From a programmer's point of view, when designing for example, a new attribute of a physical robot, plenty of time is spent on setting up collateral systems, time which can be saved using simulation software [43].

VR technologies solve uniqueness problems. Most of the times and for trivial reasons, research laboratories have one, maybe two study platforms, which have to be shared between several researchers. Using simulation eliminates the problem of concurrent use [44].

Simulation lowers the entry barrier for young scientists and improves education process. Using only a personal computer, inexperienced robot researchers can develop interesting applications in which they can program physical constrains within the virtual environments, for obtaining results close to reality.

Although simulation in virtual environments has so many good aspects, there are also some drawbacks that should be mentioned. The main weak point of any simulation is the inexact representation of reality. This translates into secondary issues, like the lack of noisy data or the usage of incomplete models of the virtual robots/environments [45]. Other issues relate to the processing power of the computer on which the simulation is running, which produces inconstancies between time in virtual environment and real life [43]. Another variable that should be taken into consideration is time spent to design and improve the usability and the realism of simulations. This is mainly occupied by the parameter tuning process [46].

Only a few studies address the problem performing manipulation experiments outside the real world. Ong et al. propose i.e. the usage of augmented reality (AR) for performing a path planning experiment with a 6 DOF robotic arm [47]. Although there have been several attempts to develop experiments with fixed or mobile virtual robots [48,49], little interest is shown in building virtual models into stereo vision mode, within an immersive virtual environment such as CAVE [50].

3. Hardware and software prerequisites for theoretical and experimental studies

When building experiments presented in this paper, the following facts were taken into account: the need of robust software that can be easily distributed to other operating systems, the need for flexible software based on object oriented programming paradigm, the need of easily-configurable software and the need of performance. The working paradigm presented in Fig. 1 has been accepted as optimal for this study.

C++/OGRE programming environment was used for constructing the virtual design part of experiments, as OGRE supplies the stereo rendering necessary to include simulations in CAVE [51]. Some other CAVE tests were conducted using XVR.

Considering the requirements and the software used, the application developed has the architecture presented in Fig. 2. Two configuration files provide the information necessary to simulate the robot and the working scene in both programming environments. MATLAB was used for building the neural network controller. The control information provided by MATLAB program is sent to either XVR or C++/OGRE, using a TCP/IP communication. Results are assessed from both sides of the developed application.

3.1. Physical structure of experimental robot

While the aim of this study is to propose an obstacle avoidance algorithm suitable for manipulation tasks in unknown environments, the objective of the experiments presented in this paper is to virtually simulate and control a specific robotic arm (PowerCube manipulator). The secondary goal of the experiments presented in this paper is to reduce accident probabilities.

The physical structure for which this study was conducted is a PowerBot robot from Active Robots—an US company specialized in building mobile robots. PowerBot comes equipped with a 6 DOF robotic arm called PowerCube (see Fig. 3).

The joints include a rotating base, a pivotating shoulder, two rotating links, a pivotating elbow, a pivotating wrist and a 1 DOF gripper that can grasp objects 6 cm wide. All the joints except gripper are rotary.

3.2. Modelling the robot

According to Fig. 1, the first step consists in modelling in VR the scene, the objects inside the scene and the robot. The robot arm is the most important entity inside the virtual environment. When modelling PowerCube, due to special attention given to kinematic details, several assumptions have been made.

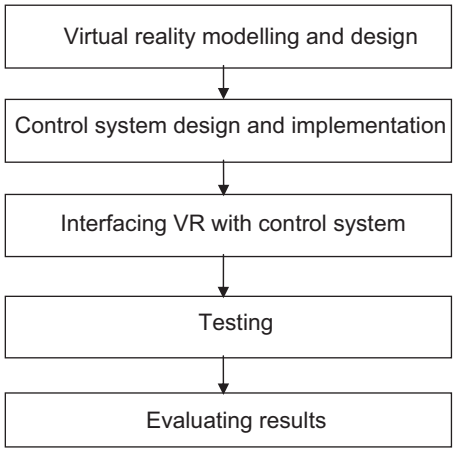

Fig. 1. Experiment steps.

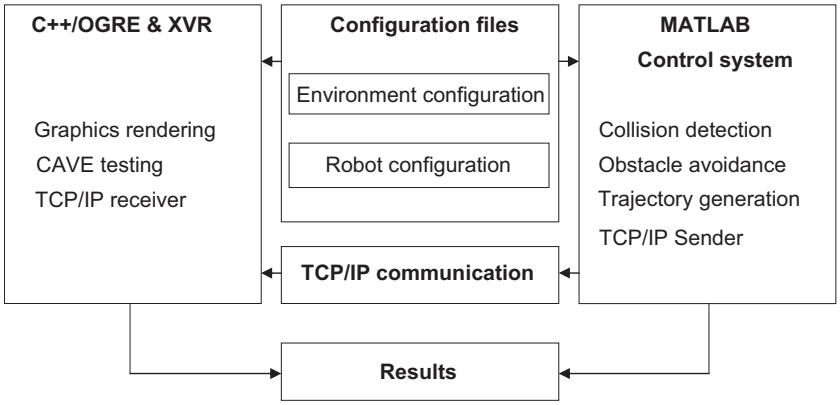

Fig. 2. Application architecture.

Fig. 3. PowerBot with PowerCube arm.

3.2.1. Assumptions

The robot arm which needs to be controlled is activating in a working environment in which are also present static obstacles. The arm has a predefined starting configuration, and after each movement, the arm retains the final configuration as the starting one for the next assignment. The arm tries to reach the goals given while avoiding obstacles. Aside this scenario information, the following assumptions are made:

Assumption 1. The arm is activating in an unknown working space. The collision detection is achieved using a mathematical method which can be translated in real environments as a supposition that on each link is installed a network of proximity sensors.

Assumption 2. The joint angle limits are considered $(-\pi, \pi)$ for simplicity reasons, although in the specific case of PowerCube, the manual specifications are slightly different [52].

Assumption 3. The orientation of the arm's EE is arbitrary—the algorithm focuses on reaching the goal without caring for a special EE angle configuration.

Assumption 4. At each time instant, the distance between the EE and target and the distance between each obstacle and each link are known.

3.2.2. Modelling the robot in VR

Nowadays, scientists have at their disposal several pieces of software that can help them to achieve a satisfying simulation of their real life scenario. Starting from modelling and ending with the simulation itself, one can use various programs such as SolidWorks or CATIA for CAD design, VRML/X3D or COLLADA as low level coding languages for animating their designs, or more focused robot simulators like Player Project, Webots or Microsoft Robotics Developer Studio which can tremendously help in reducing programming time.

For experiments presented in this paper, the robot was modelled using CATIA software (see Fig. 4). The CAD model of the arm is exported as meshes and imported into C++/OGRE or XVR for VR testing.

3.2.3. Modelling the robot in MATLAB

In order to control it, the robot arm was modelled in MATLAB. A commonly used convention for selecting frames of reference in robotic applications is the Denavit–Hartenberg (DH). The basic idea behind DH algorithm is to assign coordinate frames to each joint of the kinematic chain [53]. Each new frame provides a new homogeneous transformation matrix. When applying the algorithm for solving the DH parameters for the case of PowerCube arm, the link structure presented in Table 1 is found.

Based on these parameters, PowerCube equivalent arm scheme can be constructed, with the following amendments:

- the width of the links is considered very small, all the links are lines.
- 3rd and 4th joints are double, as described by the DH parameters (see Fig. 5).

Having the information from Table 1, any transformation matrix from link n-1 to link n may be written as

$$T_n^{n-1} = \begin{pmatrix} \cos\theta_n & -\sin\theta_n\cos\alpha_n & \sin\theta_n\sin\alpha_n & r_n\cos\theta_n \\ \sin\theta_n & \cos\theta_n\cos\alpha_n & -\cos\theta_n\sin\alpha_n & r_n\sin\theta_n \\ 0 & \sin\alpha_n & \cos\propto_n & d_n \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (5)$$

where d_i, θ_i, r_i and α_i are the DH parameters obtained for link i. Thus, the coordinates of the EE based on the homogenous

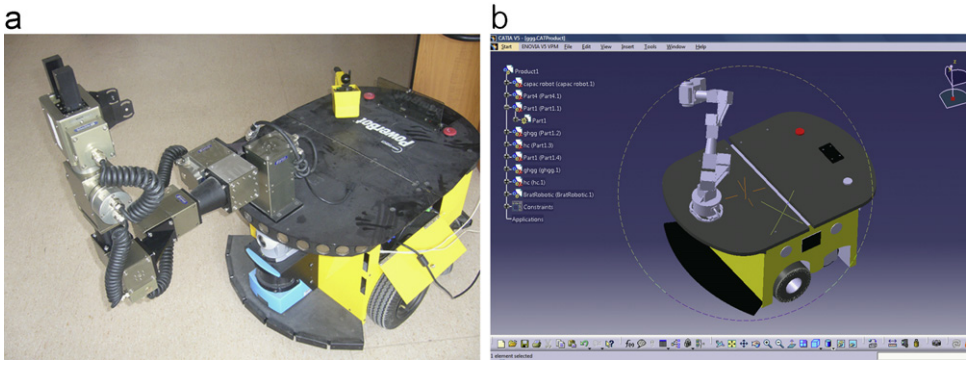

Fig. 4. PowerBot in realty (a) and the resulted CAD model (b).

Table 1
DH table for PowerCube.

Link	a (cm)	α (rad)	d	θ	Limits
1	$L_1=11.5$	-1.571	0	θ_1	$-1.571<\Phi<1.571$
2	0	1.571	0	θ_2	$-1.571<\Phi<1.571$
3	0	1.571	$L_{23}=30$	θ_3	$-1.571<\Phi<1.571$
4	$L_4=17.25$	0	0	θ_4	$-1.571<\Phi<1.571$
5	0	-1.571	0	θ_5	$-1.571<\Phi<1.571$
6	0	0	$L_{56}=17.7$	θ_6	$-1.571<\Phi<1.571$

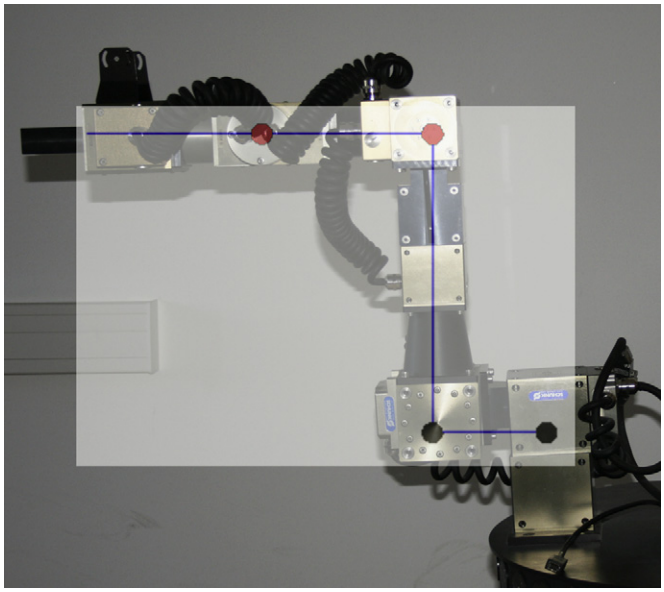

Fig. 5. The equivalent MATLAB arm structure, with last 2 joints as double rotary. The MATLAB scheme is applied with 50% opacity over arm real picture.

transformation matrices are

$$\begin{pmatrix} EE_x \\ EE_y \\ EE_z \\ 1 \end{pmatrix} = T_6^5 T_5^4 T_4^3 T_3^2 T_2^1 T_1^0 \tag{6}$$

3.2.4. Connecting MATLAB with C++

In order to achieve a stand-alone application, a link between MATLAB and C++ is needed. Unfortunately, creating a shared C++ library using MATLAB compiler is not a valid solution, as custom neural network directives cannot be deployed. Using MATLAB Engine to directly call for .m files is also not suitable for bigger projects. In the end, a less-conservative method was chosen: a TCP/IP server–client communication. The MATLAB sender transmits trajectory details (angle vectors captured at discrete amounts of time) to C++ receiver. The equivalence between the two different programming environments is illustrated in Fig. 6, from *XOY* and *XOZ* perspectives.

3.3. Modelling the environment obstacles

In real world, working environments have irregular shaped obstacles. These may vary in size and location with respect to the arm position. For simplicity reasons, we have defined 3 classes of obstacles which may be added to the workspace using a configuration file designed in the following way (Fig. 7a):

- *Spheres*: (*x*,*y*,*z*, sphere radius).
- *Parallelograms*: (*x*,*y*,*z*, length, width, height).
- *Cylinders*: (*x*,*y*,*z*, radius, height).

The collision detection is implemented using sphere covering technique. The representation is a set of radii and centres that models each object as a set of spheres. During arm movement, the world is checked to verify that no collisions happen between the spheres (the distance between any 2 circles belonging to different sets is higher than the sum of their radii).

The clustering into spheres is done using *k*-means clustering algorithm [54]. A number of clusters is chosen based on the resolution of the world. Since this approach is not normally affected by regular *k*-means issues, the proposed implementation of *k*-means algorithm retries on separation failure, ignores failure to converge and has a low and defined number of iterations. The resulted environment presented in Fig. 7b is equivalent with the environment from Fig. 7a.

4. Neural network trajectory planner

The trajectory planning for obstacle avoidance approach proposed by this study is taking a step forward from most path-oriented research conducted in this scientific area, by using an innovative solution that one may call "self-learning". Based on reinforcement learning paradigm, *Q*-learning online dynamic algorithm [55] is combined with a double neural network for achieving the target with the robotic arm's EE, while avoiding environment obstacles.

4.1. Designing the neural network trajectory planner algorithm

For better understanding of the approach presented in this paper, the problem has been reformulated as follows. Let the

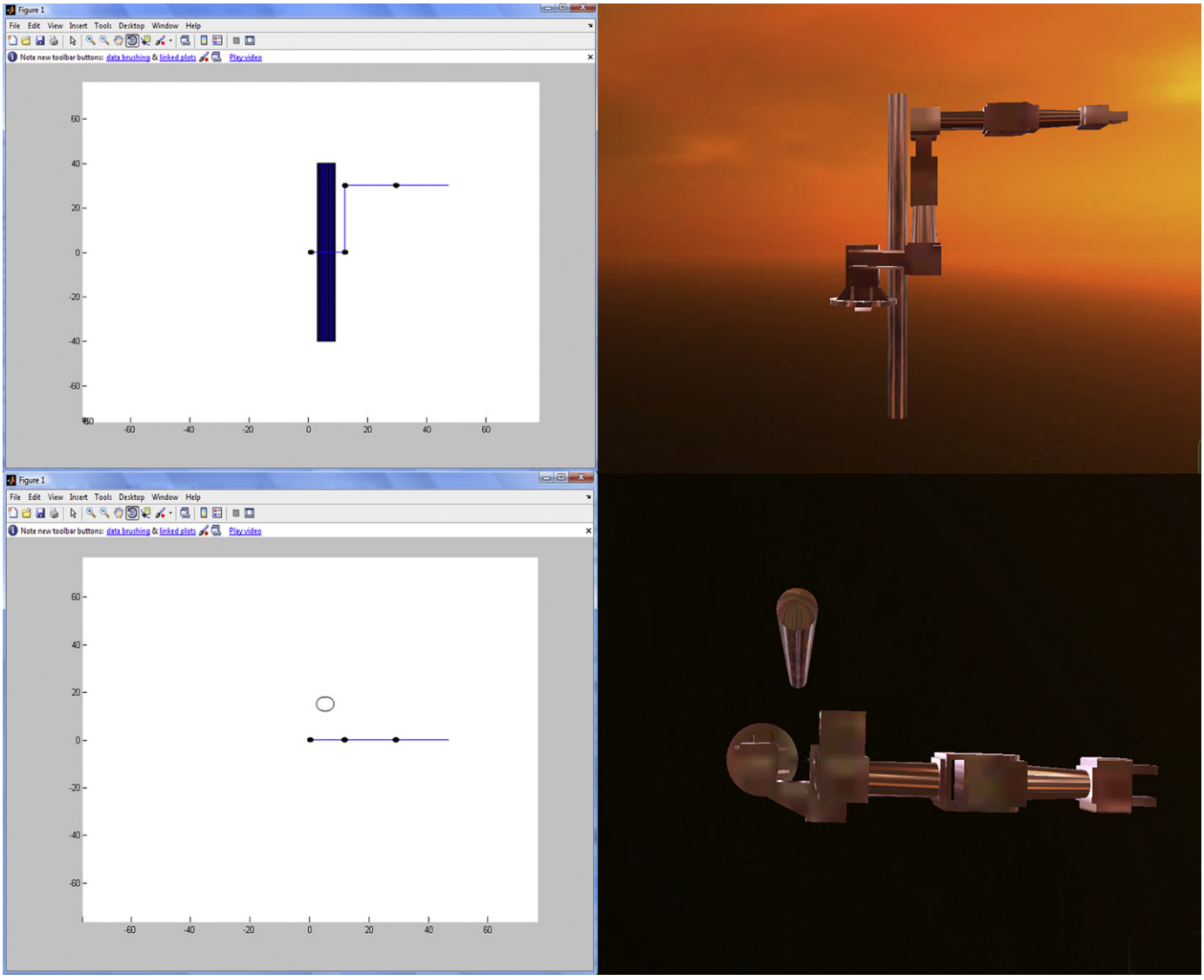

Fig. 6. MATLAB—C++ equivalence of experiment start configuration.

forward kinematics mapping be some surjective and non-injective function $f(q)$, $f: Q^6 \rightarrow X^3$, where Q is the set containing the angle vectors and X the set containing the Cartesian coordinates of the EE. The set of all possible inverse mappings G is described as

$$G = \{\cup g \,|\, g = q = f^{-1}(x)\} \tag{7}$$

where $q \in Q^6$, $x \in X^3$.

Let the angle trail for the robot motion start from a configuration q_1 and end to destination q_e. The trajectory $T(A)$ is the curve of points representing the kinematic mapping of the angle trail A described as

$$A(q_1, q_e) = \left\{ \bigcup_{t=0}^{t\text{-}final} q^t \,|\, q^t \in G - \text{angle configuration at moment } t \right\} \tag{8}$$

Let the sets of points O_1, O_2, …, O_m represent the m obstacles located in the working environment. The problem is that, given the angle configuration q_1 and the goal x_e, q_e needs to be found such that both equations are satisfied

$$f(x_e) = q_e \tag{9}$$

$$T(A(q_1, q_e)) \cap \left\{ \cup \frac{m}{i=1} O_i \right\} = \phi \tag{10}$$

The problem above can be solved using Q-learning [7]. At any given angle configuration $q \in T$, the robot is at in an intermediate state and has to choose among all different possible states. The two conditions formulated in Eqs. (9) and (10) can be represented within the neural network trajectory planner (NNTP) schema proposed in Fig. 8

– The cost function represents the first condition expressed in Eq. (9). Here, the cost function is represented by the standard Jacobian solution from the current state. The cost function is modelled by "Theta Net" neural network, which is a 35-20-6 Multi-Layer Perceptron (MLP), a configuration that resulted from a set of preliminary tests (see Fig. 9). "Theta Net" receives as input θ vector which contains the current position of the arm, and outputs the Jacobian solution, a 6 element vector which holds joint angle values. The hidden layer contains two separate levels, for better mapping of the nonlinear problem of inverse kinematics [6]. Between these 2 levels, *tansig* transfer function is used, whereas between the hidden and the output layer, it is used *purelin* transfer function (see Fig. 10). The weights of "Theta Net" neural network are updated on each iteration, using the adapt function, which receives as input the Cartesian coordinates of the goal. The process uses gradient

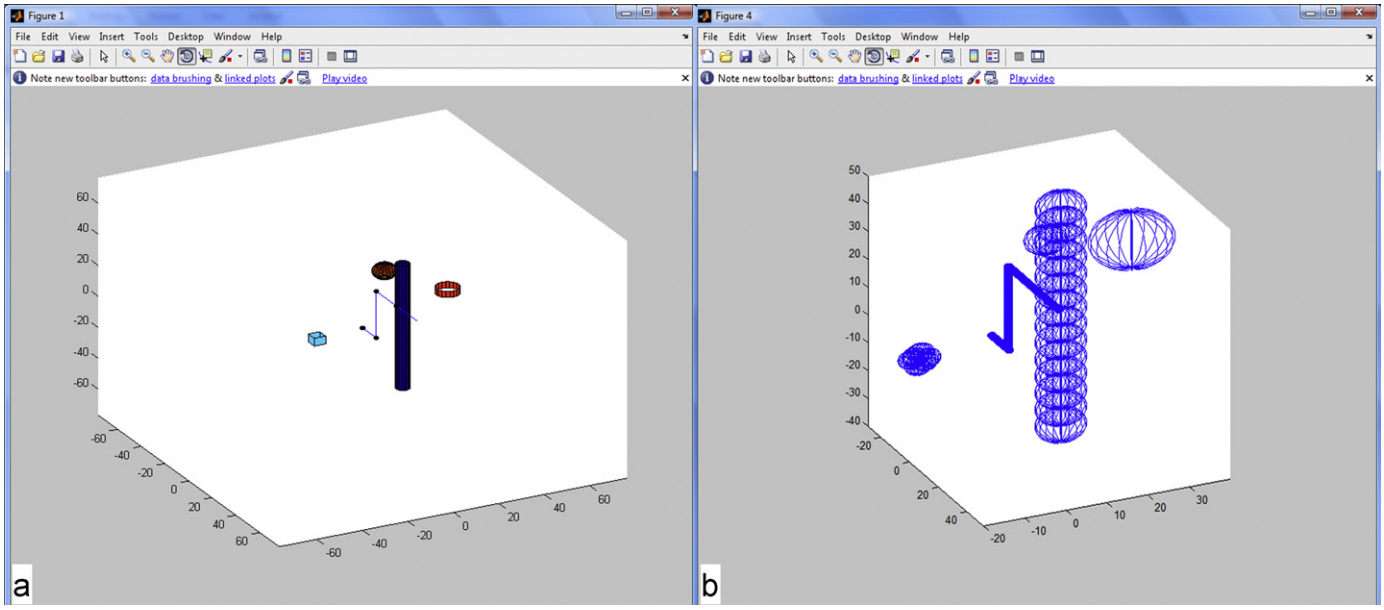

Fig. 7. Working environment with 4 obstacles, 2 cylinders, one sphere and one parallelogram (a) and equivalent scene with active collision detection (b).

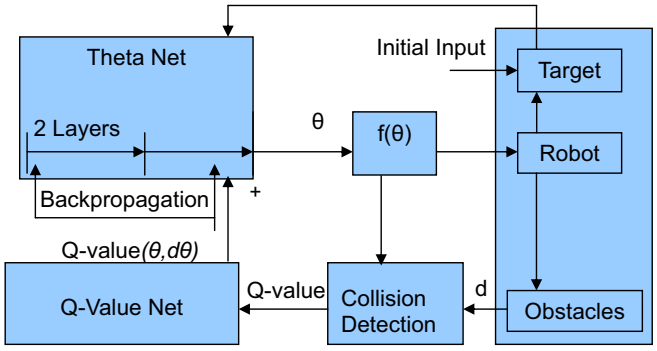

Fig. 8. NNTP structure.

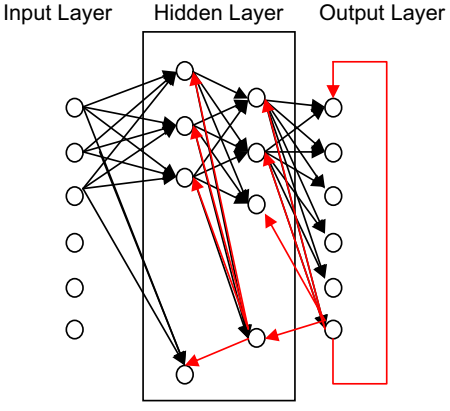

Fig. 9. "Theta Net" structure.

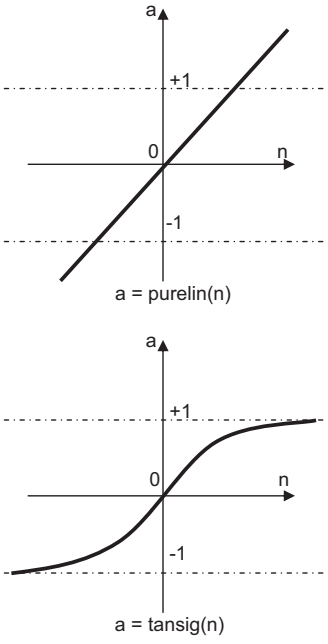

Fig. 10. Transfer functions for "Theta Net" and "Q-value Net".

descent optimization algorithm. After adapt function is applied, a 6 value output vector is obtained. If, by moving the arm, a collision is obtained or the maximum number of epochs has been reached without reaching the goal, the weights of "Theta Net" neural network are reinitialized and a new iteration is started.

– The *Q*-value of each state represents the second condition expressed in Eq. 10. *Q*-values are computed in the following way: upon encountering an obstacle or reaching the maximum number of epochs, each *Q*-value is updated in a direction opposite to the current for each *Q* belonging to the candidate *T*. If the move succeeds, the *Q*-value is diminished to let the inverse kinematics dominate. Thus, as long as the number of failures increases, *Q*-value also increases. After each iteration, the weights of "*Q*-value Net" neural network are computed with respect to *Q*-value table. "*Q*-value Net" neural network is chosen as a 20-6 MLP which receives as input the current angle configuration θ and outputs a Φ vector that is added to the output of "Theta Net", for achieving the next state in arm's trajectory.

The user inputs the goal coordinates x_e. "Theta Net" outputs θ vectors. The Collision Detection module generates *Q*-values for each θ, to which are then used to train "*Q*-value Net". "*Q*-value

Net" outputs a 6 elements vector that is added to the θ vector generated by "Theta Net", to form the final solution. The use of neural networks described above has certain similarities with lookup Dyna tables [56] or potential fields [57,58]. The advantage of the design proposed by this study relies in the capacity of the network to learn and its effect on neighbouring points, resulting in faster convergence. As the robot goes obstacle free, the *Q*-values are sent towards zero at a slow pace to counter the effect of having high values dominating the cost evaluation. After a determined maximum number of epochs, "*Q*-value Net" is reused but the "Theta Net" weights revert to original values.

4.2. Implementing the NNTP

The proposed solution is implemented according to the flow chart presented in Fig. 11. Both "Theta Net" and "*Q*-value Net" are first initialized. On a while loop, it is tested whether the reaching error e is small enough $e<0.5$ (meaning that the EE is close enough to goal) and whether the maximum number of iterations has been reached. "Theta Net" is adapted using the latest position of the arm with respect to the goal. In the following step, collision is tested. In case a collision does occur, "Theta Net" is reinitialized, the number of epochs is increased and "*Q*-value Net" is adjusted accordingly. If the number of epochs reaches the maximum number allowed (MaxEpochNo), "Theta Net" is again reinitialized along with the epoch number, and "*Q*-value Net" is again adjusted accordingly. In the event that no collision occurs and the number of epochs is lower than MaxEpochNo, the arm is moved to resulted angle configuration, "*Q*-value Net" is adapted and the iteration number is increased. If the maximum number of iterations (MaxIterations) is not reached and the reaching error is lower than a chosen threshold (0.5), a valid trajectory is discovered.

5. Simulation and study results

Before starting the tests, it is worth to say that the algorithm performance varies with computer configuration. For tests presented in this paper, an Intel Core 2 Duo Processor at 2.5 GHz with 2 GB SDRAM and with Windows Vista operating system has been used. Another thing that should be pointed out is the neural networks working parameters. These values have been chosen from data resulted from the pre-testing process of the neural network solution. The maximum number of epochs before a new iteration is MaxEpochNo=200 and the maximum number of iterations is MaxIterations=100.

Tests have been performed in MATLAB (for assessing the algorithm performance), in VR for assessing the quality of the solution with respect to human–robot interaction (HRI) and also, on the real robot.

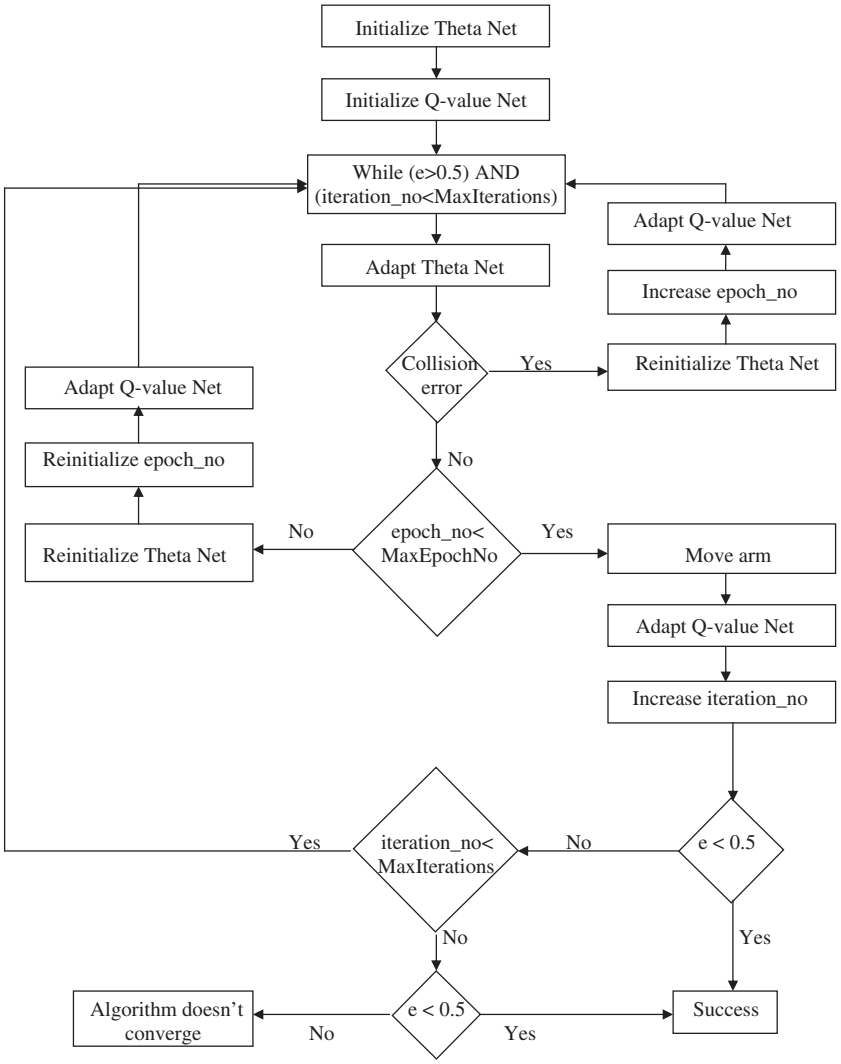

Fig. 11. NNTP algorithm flow chart.

5.1. Testing in MATLAB

In order to assess the performance of the NNTP, a series of tests were conducted, focusing on the following indicators:

- Number of iterations necessary to reach the goal.
- Number of epochs necessary to reach the goal during last iteration.
- Time spent between start and finding the solution.
- Percent of convergence fails.
- Distance error from goal.

These parameters have been evaluated in different cases, with respect to

- The number of obstacles present in the working environment: no obstacles, 1 obstacle, several obstacles.

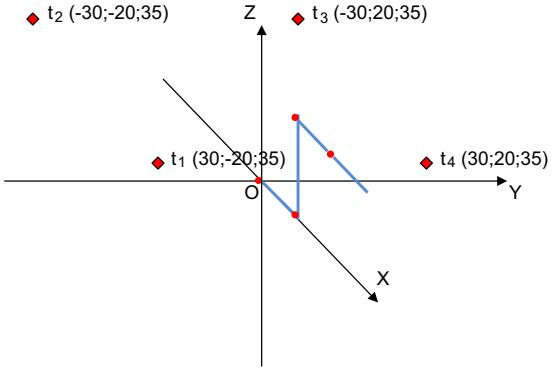

Fig. 12. Targets from the 4th sections of Cartesian coordinate system.

- The target position—each of the 4 sections of the Cartesian coordinate system are tested, as in Fig. 12.

As the values that initialize the network weights are randomly assigned, given the same start conditions (the angle configuration q_1 and the goal x_e), repeated tests produce different trajectories $T(A(q_1,q_e))$. In Fig. 13, the presented final arm configurations differ in all 6 cases. Thus, a fairly large number of tests is need in order to assess the average working performance of the proposed algorithm. For illustrating the aspects presented above, 3 test cases have been chosen:

- *case 1*—no obstacles in the working field
- *case 2*—a cylinder bar in the working field
- *case 3*—4 obstacles (a cylinder bar and 3 spheres) in the working field

For a complete assessment, the arm is given the command of reaching all 4 points t_1, t_2, t_3 and t_4 in ascending order, while the number of epochs, iterations and time between each 2 targets is measured.

5.1.1. Test scenario 1—no obstacles in working field

The results presented in Table 2 are collected during tests performed with no obstacles in the working environment. The starting configuration of the robot influences the way in which the robot achieves the targets. Note the average epochs number is directly proportional with the average time spent by the learning algorithm to find the first suitable $T(A(q_1,q_e))$ for each of the targets. After the first target point t_1 is reached, in most cases, the performance of the algorithm improves as the network is already trained when searching for the next target points t_2, t_3 and t_4. Best performance is reached when searching for the last target. This is

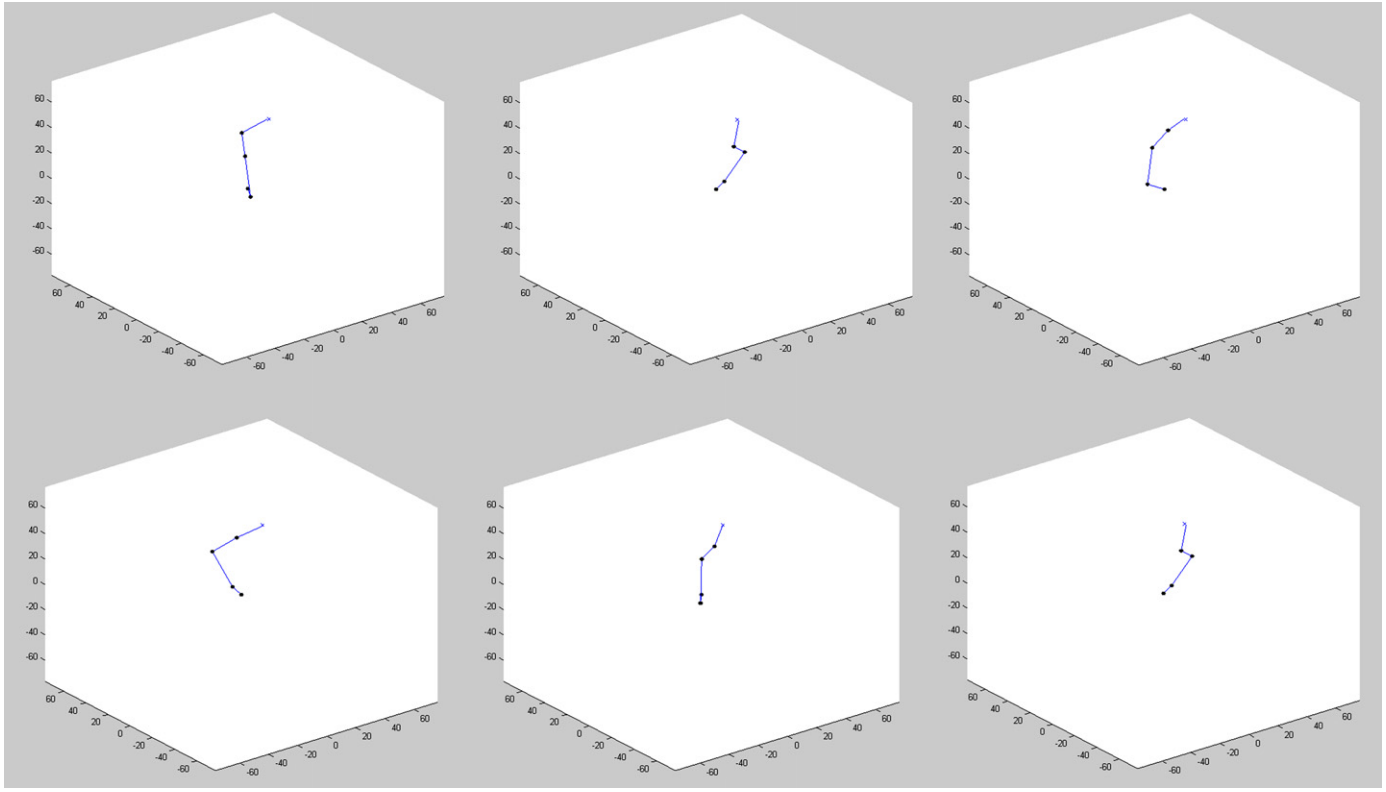

Fig. 13. Different results while achieving same goal t_1 (30;20;35), with no obstacles in working environment (case 1).

Table 2
Testing results in Case 1.

Number of tests	Variable	t_1	t_2	t_3	t_4	Average	Total
5	Iterations no. (average)	1	1	1	1	1	4
	Epochs no. (average)	86.400	36.800	27	17	41.800	167.200
	Time in seconds (average)	25.980	11.550	12.430	8.979	14.735	58.942
10	Iterations no. (average)	1	1	1	1	1	4
	Epochs no. (average)	70.800	36.300	25.700	20.400	38.300	153.200
	Time in seconds (average)	21.878	12.451	12.150	9.810	14.073	56.292
20	Iterations no. (average)	1	1	1	1	1	4
	Epochs no. (average)	71.150	34.750	24.600	20.050	37.637	150.550
	Time in seconds (average)	21.495	11.969	11.769	9.380	13.653	54.613
40	Iterations no. (average)	1	1	1	1	1	4
	Epochs no. (average)	63.825	33.925	27.150	20.750	36.412	145.650
	Time in seconds (average)	20.260	12.683	11.666	9.841	13.613	54.453

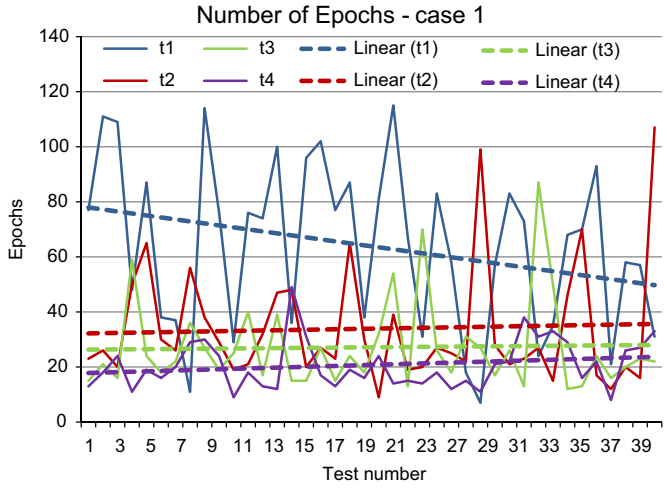

Fig. 14. Number of epochs analysed by NNTP for reaching each target, with no obstacles in working environment (case 1).

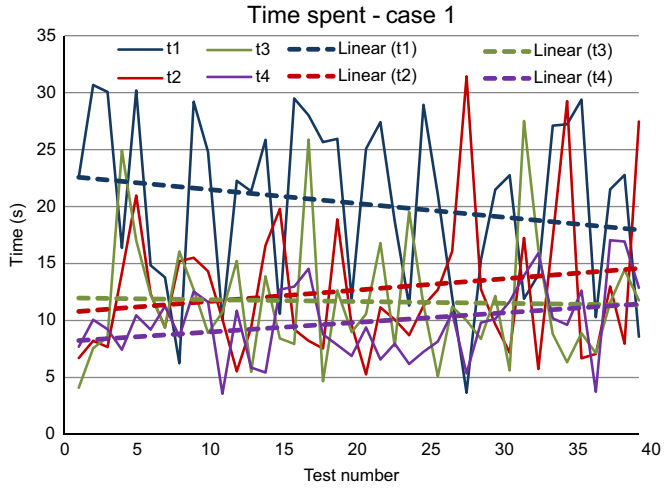

Fig. 15. Time spent (in seconds) by NNTP for reaching each target, with no obstacles in working environment (case 1).

showed in Figs. 14 and 15 by the linear trend lines. After 40 tests, we conclude that the network converges to solution (with a small distance error $e \leq 0.5$ for each target) after 1 iteration, with an average of total number of epochs of 14.565 and with an average of total time spend of 54.4532 s. The average time for reaching one target is 13.613 s.

5.1.2. Test scenario 2—a cylinder bar in the working field

The results presented in Table 3 are collected during tests with a cylinder bar in the working environment, located at Cartesian coordinates (5, 15, −40), of radius 3 and with height 80 (see Fig. 16).

As before, we have same 4 target points, each in different sections of Cartesian coordinate system. After 40 tests, the average number of total epochs to reach all 4 solutions with same small error $e \leq 0.5$ is 209.475 (but now the average number of total iterations is 12.3) and the average time spent to reach all 4 targets is 84.79915 s. The last 2 targets have the biggest average number of iterations, 3.65 and 4.925.

As one can see, the performance of the proposed solution is fairly influenced by the position of the target with respect to the position of the obstacles and the start configuration of the arm. Imposing a small distance error e ($e \leq 0.5$) translates into slower convergence times, but with higher precision, which is usually necessary in any manipulation application. What is to be noticed is that the average performance values are influenced by just a few cases (7.5% of tests have the number of iterations higher then 7, with target t_4 being the hardest to achieve) where the random chosen weights are bad enough to ask more than 5 iterations for training (see Fig. 17).

As one can see in Fig. 18, the average time spent to reach targets t_1, t_2, t_3 and t_4 is actually, in most tests, less than 20 s. Extreme cases (when time spent is higher than 30 s) have a big impact in this brief data collection.

5.1.3. Test scenario 3–4 obstacles in the working field

The results presented in Table 4 are collected during tests performed within a working environment which contains a cylinder bar located at the same coordinates as in case 2, and 3 spheres located at Cartesian coordinates (−20, 20, 20), (−20, −20, −20) and (20, −20, −20), all with radius 4 (see Fig. 19).

After 40 tests, the average time spend to reach all 4 targets is 101.9573 s. Due to custom configuration of the working environment, the fastest target reached is t_2 (see the linear trend lines in Fig. 20), within 19.11505 s on average. Also, the convergence of the proposed solution is 100%.

As in case 2, the performance of the neural network controller is influenced by just a few cases (10.6% of tests have the number of iterations higher then 7) where the random chosen weights ask more than 5 iterations for training (see Fig. 21).

5.2. Testing in VR

The experiments developed in VR focus on 2 issues: testing the usability of the NNTP and assessing the performance of the solution.

Table 3
Testing results in Case 2.

Number of tests	Variable	t_1	t_2	t_3	t_4	Average	Total
5	Iterations no. (average)	1.600	1	5.200	6	3.450	13.800
	Epochs no. (average)	71.400	24.400	73.400	14.6	45.95	183.800
	Time in seconds (average)	23.952	10.239	32.925	13.991	20.277	81.108
10	Iterations no. (average)	2.500	3.700	3.400	4.100	3.425	13.700
	Epochs no. (average)	95.700	54.400	64.600	16.400	57.775	231.100
	Time in seconds (average)	30.967	20.507	26.139	12.747	22.590	90.361
20	Iterations no. (average)	2.150	2.550	2.500	5.400	3.150	12.600
	Epochs no. (average)	85.450	54.050	50.400	37.550	56.862	227.450
	Time in seconds (average)	27.096	19.546	20.616	21.374	22.158	88.633
40	Iterations no. (average)	1.775	1.950	3.650	4.925	3.075	12.300
	Epochs no. (average)	74.125	43.900	48.125	43.325	52.368	209.475
	Time in seconds (average)	24.244	16.986	21.367	22.200	21.199	84.799

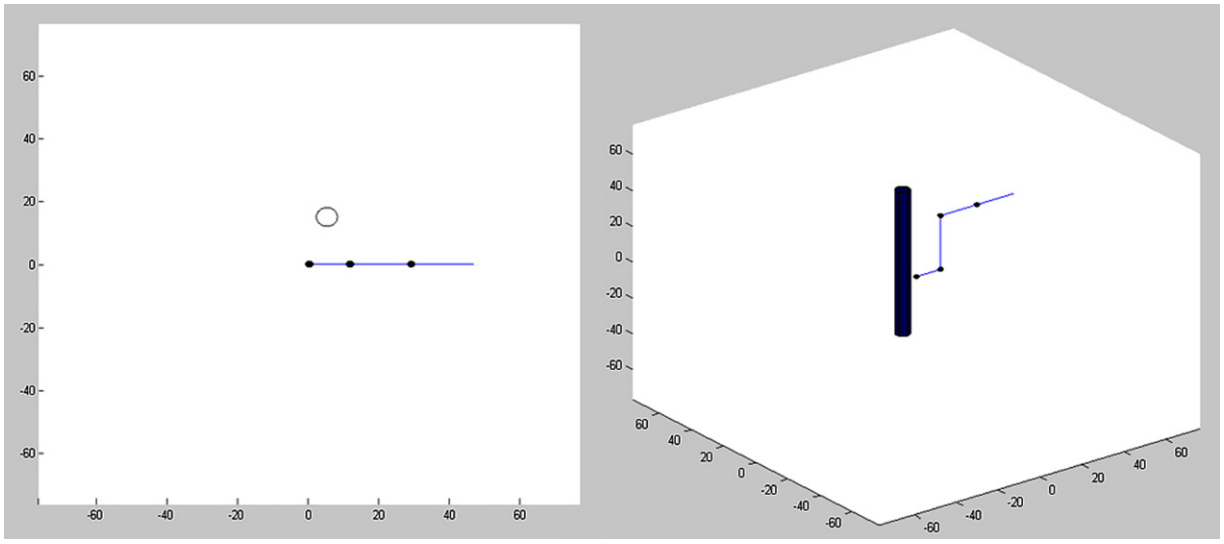

Fig. 16. Case 2 scenario—a cylinder bar present in the working field.

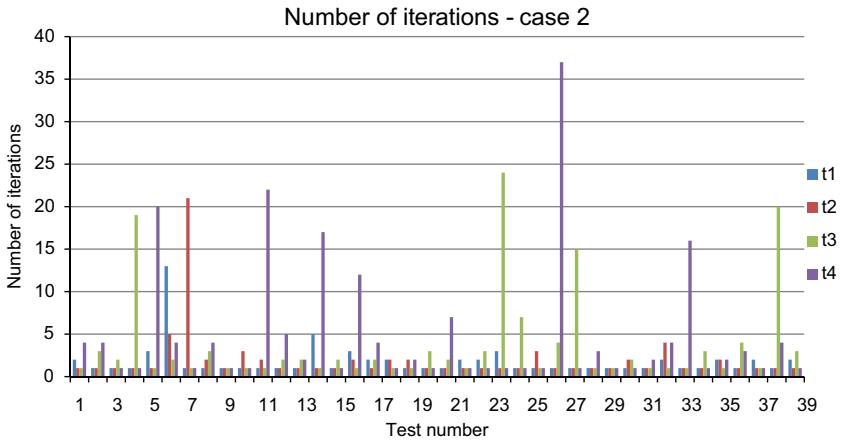

Fig. 17. Number of iterations required by NNTP for reaching each target, with one obstacle in working environment (case 2).

The main problem that arises when performing experiments in immersive VR is detecting the target (in this case, target has been chosen as the position of the user's right hand). This can be achieved through various technologies: using magnetic fields [59], video processing of fiduciary markers [60], sound processing by acoustical trackers [61], optical trackers [62] and others. The method used in this research is optical tracking. Optical trackers use infrared light sources coming from optical sensors (cameras) that bounce off from small reflective spheres attached to the object that needs to be tracked. Knowing the orientation and position of the spheres with respect to a reference point, the pose of the target can be computed [63].

Within CAVE, the testing program was extended to include the usage of ArtTrack optical tracker system. With the help of ArtTrack, a rigid body attached to subject's hand can be localized inside CAVE (Fig. 22a). Thus, the user is able to move the goal position within the working environment of the robotic virtual arm (Fig. 22b).

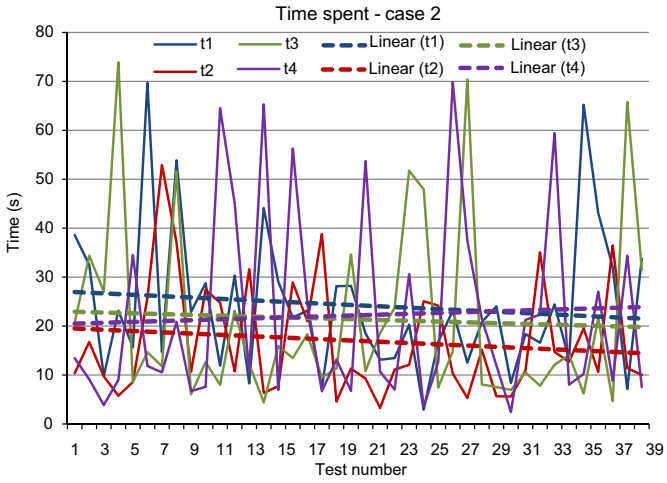

Fig. 18. Time spent (in seconds) by NNTP for reaching each target, with one obstacle in working environment (case 2).

After assuming that the user is located at a certain distance in front of the virtual manipulator, several tests have been conducted for continuously reaching the target, which in the proposed case, is a virtual bottle attached to the detected position of the rigid body (Fig. 23). The subject is asked to sequentially place his hand in different positions and to hold each position for a small amount of time (5 s), for validating the goal. The robot reaches the goal with an average time of 21 s (data obtained in test scenario number 2, after 40 trials).

5.3. Testing in real environment

Achieving a safe testing is the first reason for which this study was needed. Testing real life manipulation scenarios with PowerCube (and other robotic manipulators) imposes additional work in solving security issues, foreseeing and solving possible hardware and software malfunctions and preparing additional equipment for possible injuries. The proposed virtual solution eliminates all these problems. However, in order to validate virtual tests, real experiments also need to be performed.

Although the virtual model presented above is close to the real life robot, there still are some issues that need to be handled. The

Table4
Testing results in Case 3.

Number of tests	Variable	t_1	t_2	t_3	t_4	Average	Total
5	Iterations no. (average)	3.600	2	8.400	8.600	5.650	22.600
	Epochs no. (average)	110.400	61.200	56.600	64.800	73.250	293
	Time in seconds (average)	38.594	25.721	40.232	28.719	33.316	133.267
10	Iterations no. (average)	2.800	1.900	5.100	8.600	4.600	18.400
	Epochs no. (average)	92.900	58.300	42.900	60.900	63.750	255
	Time in seconds (average)	31.299	23.286	28.168	31.369	28.531	114.124
20	Iterations no. (average)	2.400	1.600	4.700	7.850	4.137	16.550
	Epochs no. (average)	98.650	48.500	47.400	56.500	62.762	251.050
	Time in seconds (average)	31.607	19.623	26.455	31.487	27.293	109.173
40	Iterations no. (average)	2.200	1.575	5.425	6.475	3.918	15.675
	Epochs no. (average)	78.500	48.625	55.375	46.975	57.368	229.475
	Time in seconds (average)	26.171	19.115	28.264	28.406	25.489	101.957

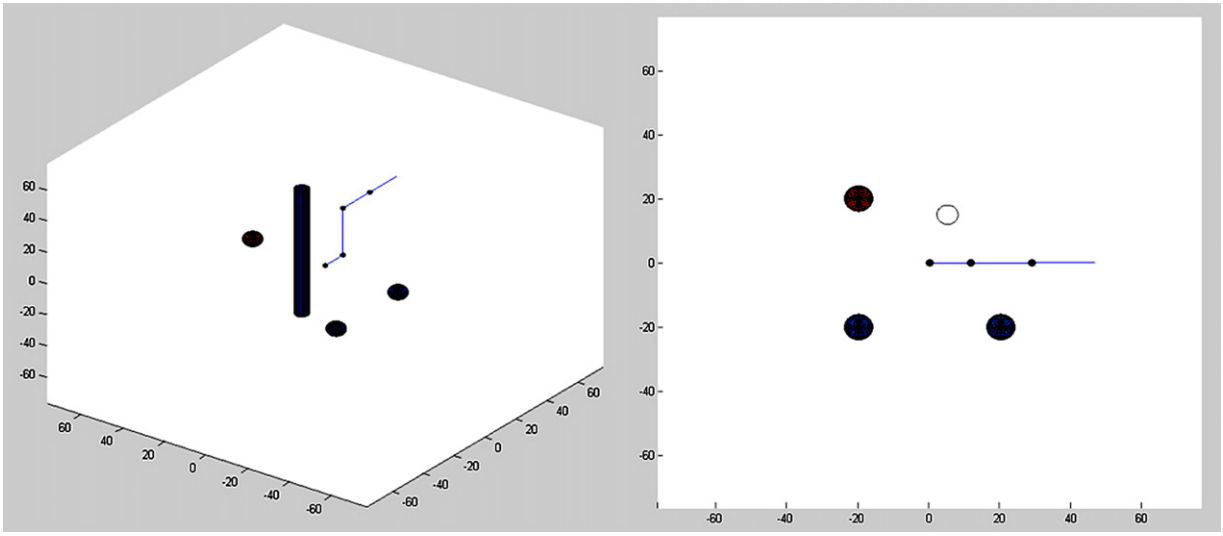

Fig. 19. Case 3 testing—a cylinder bar and 3 spheres in the working field.

real robot has wires between each link, wires that have not been integrated into the simulated model. Another issue is the inconstancy between the real environment and the simulated one. Careful measures have been taken in order to have a good virtual replica the real setup, however, due to the nature of the measuring process (which is inexact), the simulated arm slightly differs in some dimensions, as well as the virtual working environment, compared with the real one. The real values of the angles of each joint are read via a serial data interface (which is also used to give motion commands)—CAN-BUS. These may be slightly different from the actual values given for movement, due to encoder errors, physical constrains and so on. Voltage pays a high role in PowerCube movement. Because PowerCube is a relatively massive structure mounted on a mobile platform, it relies on batteries which may give variable voltage, depending on their degree of charge. There may be cases when, because of the low voltage, the arm will execute an unexpected movement geometrically different from the simulated one.

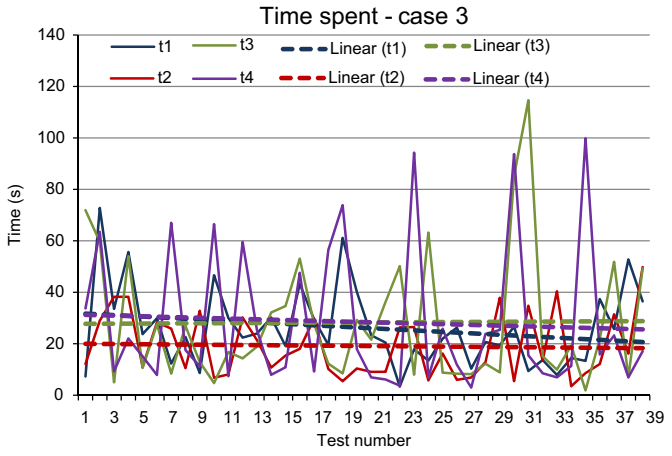

Fig. 20. Time spent (in seconds) by NNTP for reaching each target, with 4 obstacles in working environment (case 3).

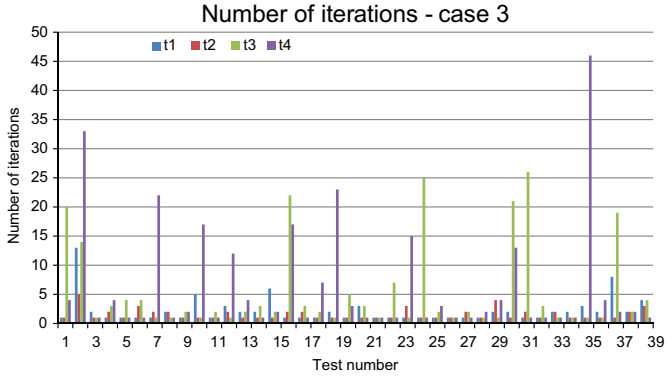

Fig. 21. Number of iterations required by NNTP for reaching each target, with 4 obstacles in working environment (case 3).

In order to solve most of these problems, the trajectory generated by the proposed solution is spitted into several frames, each asking for manual user approval before actually performing the movement. Based on this, testing scenario 2 was replicated in laboratory conditions. The simulated working environment had to be modified to include the robot body, chairs and the ground level as obstacles. The arm is given 3 different target positions (t'_1(55, 20, 0), t'_2(55, 40, 0) and t'_3(5, 40, 0)). The movement was closely supervised by a human operator. The resulted movements for all 3 targets are presented in Fig. 24.

6. Conclusions and further development

As final words, it is concluded that this study has focused on solving the problem of performing a robotic manipulation experiment. A new neural network path planning algorithm based on reinforcement learning was proposed for solving obstacle avoidance problem. Taking advantage of several VR techniques, the usage of physical structures was eliminated to the gain of software implementation. A redundant manipulator was designed, modelled and animated.

There are several further developments which the authors intend to pursuit:

1. Eliminate the cases in which the randomly chosen values used to initialize the weights of the neural network lead to a high number of iterations before achieving convergence.
2. Assess the quality of virtual HRI, a useful step for validating the presented approach.
3. Assess whether collision detection may be improved by replacing the sphere covering technique with an algorithm based on "artificial potential fields" method [21].
4. Assess whether the proposed solution may be used for solving the path planning problem in the case of having moving obstacles inside the workspace [64].
5. Develop complex industrial scenarios for further testing of the proposed solution.

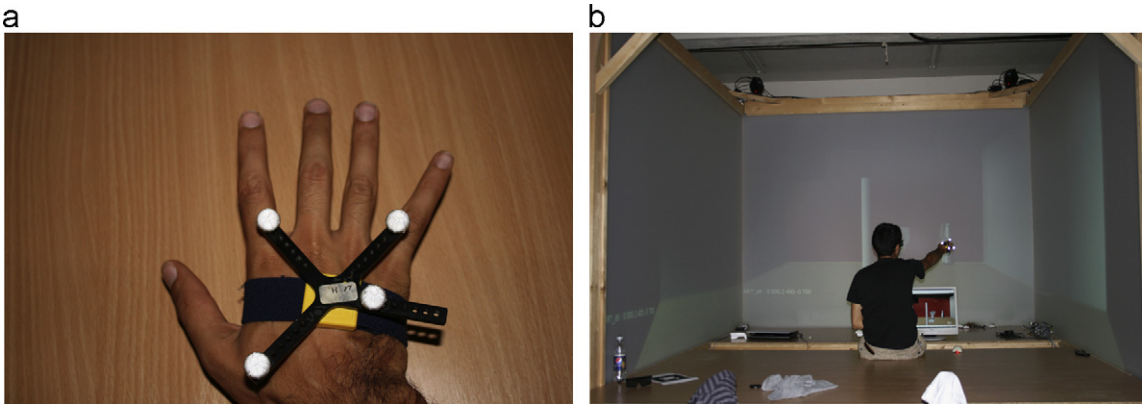

Fig. 22. Using a rigid body with passive markers attached to its hand (a), the subject is testing scenario 2 (cylinder bar present in the working field) inside CAVE (b).

Fig. 23. Testing in VR.

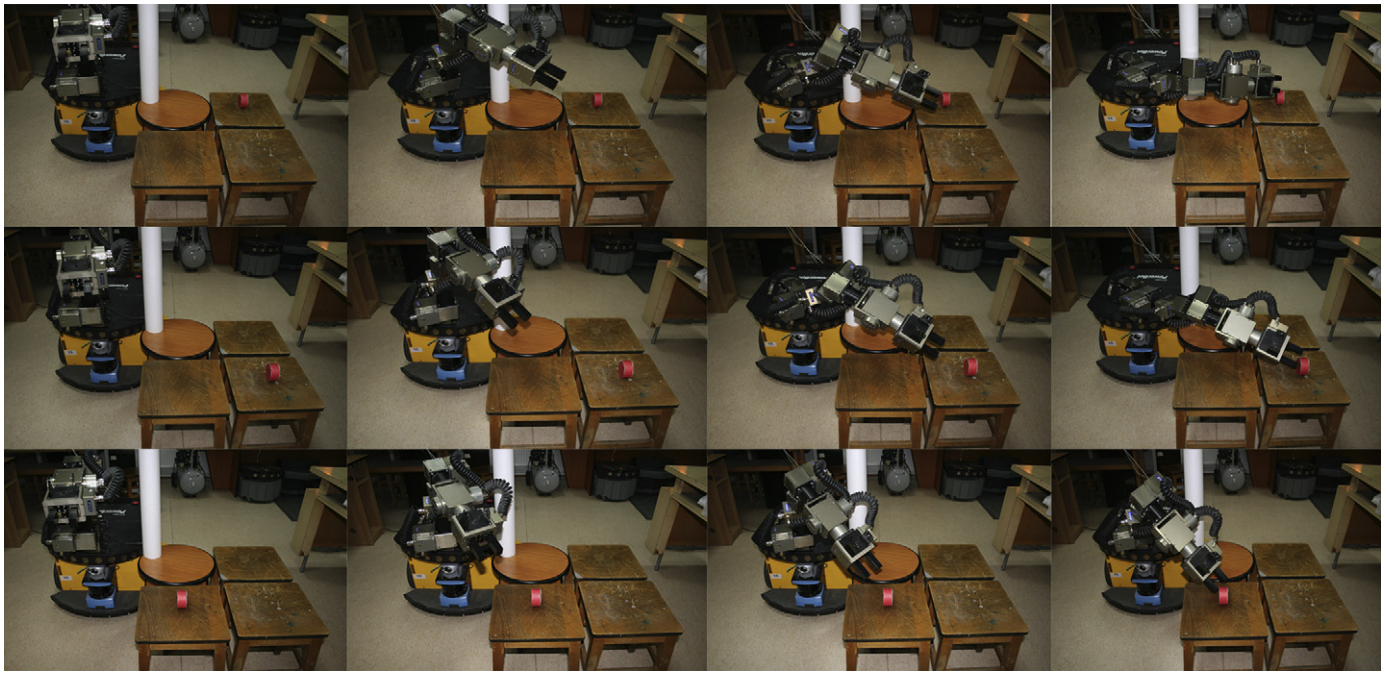

Fig. 24. PowerCube movement during 3 different testing cases.

Acknowledgements

This paper is supported by Romanian Government, CNCSIS—UEFISCSU, project numbers PNII-IDEI 775/2008.

References

[1] Edsinger A. Robot manipulation in human environments. PhD thesis. Massachusetts Institute of Technology; 2007.
[2] Kang TG. Solving inverse kinematics constraint problems for highly articulated models. Master thesis. University of Waterloo; 2000.
[3] Ding H, Chang. SE. A real-time planning algorithm for obstacle avoidance of redundant robots. Intelligent and Robotic Systems 1996;16:229–43.
[4] Penrose R. A generalized inverse for matrices. Proceedings of the Cambridge Philosophical Society 1955;51:406–13.
[5] Sciavicco L, Siciliano. B. A solution algorithm to the inverse kinematic problem for redundant manipulators. IEEE Transactions on Robotics and Automation 1998;4:403–10.
[6] Hecht-Nielsen R. Conterpropagation networks. Proceedings of the International Conference on Neural Networks 1988;2:121–39.
[7] Watkins C. Learning from delayed rewards. PhD thesis. University of Cambridge; 1989.
[8] Martin P, Millan J. Combining reinforcement learning and differential inverse kinematics for collision-free motion of multilink manipulators. Lecture Notes in Computer Science 1997;1240:1324–33.
[9] Russell SJ, Norvig. P. Artificial intelligence: a modern approach. 2nd ed. New Jersey: Prentice-Hall; 2003.
[10] Tang W, Lam L, Wang J. Kinematic control and obstacle avoidance for redundant manipulators using a recurrent neural network. In: Proceedings of the international conference on artificial neural networks; 2001. p. 922–9.
[11] Kumar S, Behera L, McGinnity TM. Kinematic control of a redundant manipulator using an inverse-forward adaptive scheme with a KSOM based hint generator. Robotics and Autonomous Systems 2010;58:622–33.
[12] Chiaverini S. Singularity-robust task-priority redundancy resolution for real-time kinematic control of robot manipulators. IEEE Transactions on Robotics and Automation 1997;13(3):398–410.

[13] Chung CY, Lee BH, Kim MS, Lee CW. Torque optimizing control with singularity-robustness for kinematically redundant robots. Intelligent and Robotic Systems 2000;28(3):231–58.
[14] Jamisola RS. Optimization of failure-tolerant workspaces for redundant manipulators. Philippine Science Letters 2010;3(1):66–75.
[15] Rotenberg S. Inverse kinematics courses. Available via 〈http://graphics.ucsd.edu/courses/cse169_w05/〉; 2005.
[16] Ding H, Chang. SE. A real-time planning algorithm for obstacle avoidance of redundant robots. Intelligent and Robotic Systems 1996;16:229–43.
[17] Baillieul J. Kinematic programming alternatives for redundant manipulators. Robotics and Automation 1985:722–8.
[18] Zhang Y. Minimum-energy redundancy resolution unified by quadratic programming. In: Proceedings of the IEEE international conference on robotics and automation; 2003. p. 2747–52.
[19] Kim J, Khosla PK. Real-time obstacle avoidance using harmonic potential functions. IEEE Transactions on Robotics and Automation 1992;3:338–49.
[20] Ramalho MA. A fuzzy approach to obstacle avoidance for small robots. Bulletin of the Transilvania University of Brasov 2008;15(50):461–7.
[21] Khatib O. Real-time obstacle avoidance for manipulators and mobile robots. Robotics Research 1986;5:90–9.
[22] Volpe R, Khosla P. Manipulator control with superquadric artificial potential functions: theory and experiments. Proceedings of IEEE Conference on Systems, Manufacturing and Cybernetics 1990;20:1423–36.
[23] Zhang Y, Wang J. Obstacle avoidance for kinematically redundant manipulators using a dual neural network. Proceedings of IEEE Conference on Systems, Manufacturing and Cybernetics 2004;34(1):752–9.
[24] Zhang Y, et al. Inequality-based manipulator-obstacle avoidance using the LVI-based primal-dual neural network. In: Proceedings of the IEEE international conference on robotics and biomimetics; 2006. p. 1459–64.
[25] Zhang Y, Ma S. Minimum-energy redundancy resolution of robot manipulators unified by quadratic programming and its online solution. In: Proceedings of the IEEE international conference on mechatronics and automation; 2007. p. 3232–7.
[26] Zhang Y, et al. More illustrative investigation on window-shaped obstacle avoidance of robot manipulators using a simplified LVI-based primal-dual neural network. In: Proceedings of the IEEE international conference on mechatronics and automation; 2009. p. 4240–5.
[27] Bingul Z, Ertunc HM, Oysu C. Applying neural network to inverse kinematic problem for 6R robot manipulator with offset wrist. In: Proceedings of the seventh international conference on adaptive and natural computing algorithms; 2005. p. 112–5.
[28] Norwood JD. A neural network approach to the redundant robot inverse kinematic problem in the presence of obstacles. PhD thesis. Rice University; 1991.
[29] Mao Z, Hsia TC. Obstacle avoidance inverse kinematics solution of redundant robots by neural networks. Robotica 1997;15(1):3–10.
[30] Tsuji T, Tanaka. Y. On-line learning of robot arm impedance using neural networks. Robotics and Autonomous Systems 2005;52(4):257–71.
[31] Moriarty DE, Miikkulainen R. Evolving obstacle avoidance behaviour in a robot arm. In: Proceedings of the fourth international conference on simulation of adaptive behaviour; 1996. p. 468–75.
[32] Touzet CF. Neural networks and *Q*-learning for robotics. In: Proceedings of the international joint conference on neural networks (IJCNN'99); 1999.
[33] Hasan AT, Hamouda AMS, Ismail N, Al-Assadi HMAA. An adaptive-learning algorithm to solve the inverse kinematics problem of a 6 D.O.F serial robot manipulator. Advances in Engineering Software 2006;37(7):432–8.
[34] Ramaswamy S VERAM: virtual environments for education, robotics, automation and manufacturing. In: Proceedings of the NSF design and manufacturing grantees conference; 1999.
[35] Dragulescu D, Stoicanescu C, Toth-Tascau M, Dreucean M, Stoia I. Mobile robot motion simulation in a virtual laboratory. Machine Building and Electrical Engineering 2006;4–5:31–5.
[36] Popovici DM, Bogdan CM, Matei A, Voinea V, Popovic N. Virtual heritage reconstruction based on an ontological description of the artifacts. Computers, Communications and Control 2008;3:460–4.
[37] Zachmann G. Virtual reality in assembly simulation—collision detection, simulation algorithms, and interaction techniques. PhD thesis. Darmstadt University; 2000.
[38] Colon E, Salhi H, Baudoin Y. MoRoS3D, a multi mobile robot 3D simulator. In: Proceedings of the ninth international conference on climbing and walking robots; 2006. p. 722–8.
[39] Interrante V, Anderson L, Ries B. Distance perception in immersive virtual environments, revisited. In: Proceedings of the IEEE conference on virtual reality; 2006. p. 3–10.
[40] Shapiro M. Comparing user experience in a panoramic HMD vs projection wall virtual reality system; 2006.
[41] Calvin KLOr, Vincent G, Cheung CC. Perception of safe robot idle time in virtual reality and real industrial environments. Industrial Ergonomics 2009;39(5):807–12.
[42] Johnson A, Leigh J. Tele-immersive collaboration in the CAVE research network. In: Collaborative virtual environments: digital places and spaces for interaction; 2001. p. 225–43.
[43] Johns K, Taylor T. Professional microsoft robotics developer studio. Wrox Press; 2008.
[44] Haton B, Mogan G. Enhanced ergonomics and virtual reality applied to industrial robot programming. Scientific Bulletin of Politehnica University of Timisoara 2008.
[45] Morgan S. Programming microsoft robotics studio. Microsoft Press; 2008.
[46] Sheridan TB. Constraint, intelligence, and control hierarchy in virtual environments. In: Intelligent motion and interaction within virtual environments; 2005. p. 9–19.
[47] Ong SK, Chong JWS, Nee. AYC. A novel AR-based robot programming and path planning methodology. Robotics and Autonomous Systems 2010;26:240–9.
[48] Duffy BR, O'Hare GMP, O'Donoghue RPS, Rooney CFB, Collier RW. Reality & virtual reality in mobile robotics. In: First international workshop on managing interactions in smart environments MANSE'99; 1999.
[49] Monferrer A, Bonyuet D. Cooperative robot teleoperation through virtual reality interfaces. In: Proceedings of the sixth international conference on information visualisation; 2002. p. 243–8.
[50] Duguleana M, Barbuceanu FG. Designing of virtual reality environments for mobile robots programming. Solid State Phenomena 2010;166–167:185–90.
[51] Vajta L, Juhasz. T. The role of 3D simulation in the advanced robotic design, test and control. Cutting Edge Robotics 2005:47–60.
[52] Tschakarow R. Programmers guide for PowerCube; 2007.
[53] Spong MW, Hutchinson S, Vidyasagar M Robot modelling and control; 2005.
[54] Ar S, Chazelle B, Tal A. Self-customized BSP trees for collision detection. Computational Geometry: Theory and Applications 2000;15(1–3):91–102.
[55] Watkins C, Dayan. P. *Q*-learning. Machine Learning 1992;8:279–92.
[56] Sutton RS. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In: Proceedings of the seventh international conference on machine learning; 1990. p. 216–24.
[57] Brock O, Khatib O. Real-time re-planning in high-dimensional configuration spaces using sets of homotopic paths. In: Proceedings of the international conference on robotics and automation, vol. 1; 2000. p. 550–5.
[58] Park D, Hoffmann H, Pastor P, Schaal S. Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields. In: Proceedings of the eighth IEEE-RAS international conference on humanoid robots; 2008. p. 91–8.
[59] Ascension Technology Corporation. Flock of birds six degrees-of-freedom measurement device. Available via 〈http://www.ascension-tech.com〉; 2010.
[60] de Ipina DL, Mendoca PRS, Hopper. A. TRIP: a Low-Cost Vision-Based Location System for Ubiquitous Computing. Personal and Ubiquitous Computing 2002;6(3):206–19.
[61] Intersense. Technical Overview IS-900 Motion Tracking System. Available via 〈http://www.intersense.com〉; 2010.
[62] Advanced Realtime Tracking GmbH, A.R.T. Tracker flyer. Available via 〈http://www.ar-tracking.de〉; 2010.
[63] Lovell-Smith CD A prototype optical tracking system: investigation and development. Master thesis. University of Canterbury; 2009.
[64] van der Berg Jur, Overmars M. Planning time minimal safe paths amidst unpredictable moving obstacles. International Journal of Robot Research 2008;27(11–12):1274–94.