Unity3D-MatLab Simulator in Real Time for Robotics Applications

Víctor Hugo Andaluz[1,2(✉)], Fernando A. Chicaiza[1], Cristian Gallardo[2], Washington X. Quevedo[1], José Varela[2], Jorge S. Sánchez[1], and Oscar Arteaga[1]

[1] Universidad de las Fuerzas Armadas ESPE, Sangolquí, Ecuador
{vhandaluzl, fachicaiza, wxquevedo, jssanchez, obarteaga}@espe.edu.ec
[2] Universidad Técnica de Ambato, Ambato, Ecuador
cmgallardop@gmail.com, jazjose@hotmail.es

Abstract. This paper presents the implementation of a new 3D simulator applied to the area of robotics. The simulator allows to analyze the performance of different schemes of autonomous and/or tele-operated control in structured environments, partially structured and unstructured. For robot-environment interaction is considered virtual reality software Unity3D, this software exchanges information with MATLAB to execute different control algorithms proposed through the use of shared memory. The exchange of information in real time between the two software is essential because the advanced control algorithms require a feedback from the robot-environment interaction to close the control loop, while the simulated robot updates its kinematic and dynamic parameters depending on controllability variables calculated by MATLAB. Finally, the 3D simulator is evaluated by implementing an autonomous control scheme to solve the problem of path following of a 6DOF robot arm, also the results obtained by implementing the tele-operation scheme for said robot are presented.

Keywords: Simulator 3D · Virtual reality simulator · Path following · Unity3d-MATLAB · Shared memory

1 Introduction

In recent years, robotics research has experienced a significant change. Research interests are moving from the development of robots for structured industrial environments to the development of autonomous mobile robots operating in unstructured and natural environments [1–5]. The robotic generally is classified according to their field of application, industrial robotics and service robotics [6–8]. In industrial and service applications it is necessary to avoid mistakes, they can cause economic and human losses; in this context it is necessary to have an environment in which to experience the performance of robots before they pass to perform any task in a real environment, for which it is considered a virtual simulation environment.

L.T. De Paolis and A. Mongelli (Eds.): AVR 2016, Part I, LNCS 9768, pp. 246–263, 2016.
DOI: 10.1007/978-3-319-40621-3_19

A virtual environment is an environment in which simulations activities found in everyday life are made, this is done with the purpose of bringing these activities to a controlled environment and analyze more deeply the stability and robustness of the systems designed, permitting in this virtual environment test, you may experience various system disturbances, and thus obtain a complete study of the operation of the system.

The advancement of technology has developed computers that let you simulations increasingly real and complex in different areas. A virtual environment would be divided into: *(i) interactive environment* it means that the user is "free" to navigate the virtual environment without having programmed the trajectory that you want to move, the system responds according to the user's wishes, this represents that the user can make decisions in "real time" in order to observe the scene from the selected viewpoint [9, 10]; *(ii) implicit interaction* this refers to the user must not learn commands or a procedure to perform some action in the virtual world, by contrast, the user performs movements that are natural to those used in the real world to move. It then searches the computer suits human nature and not the other, thereby ensuring that the experience in the virtual environment is as near as possible to the experience in the real world [9, 10]; and *(iii) sensory immersion* refers to disconnect sense the real world and the connection thereof to the virtual world [10].

The virtual environment was initially developed for application in computer games and consoles, recently the virtual environments are used to simulate different applications in the area of robotics. There are several commercial programs for the design and simulation of robots in virtual environments, between to simulate the behavior of any robot model are: Robcad, Robotstudio, Igrip, Workcell, Gazebo [11], etc.; specific for a robot in particular, *e.g.* V_CAT, V_TRAISIG y V_ISUAL of Staubli, not all programs are compatible with other CAD systems, do not support libraries all robots or other elements if any, and some are not sold under Windows, in this context, a software that is compatible with most CAD systems is sought, Unity3D for which the platform is analyzed.

Unity3D is a graphics engine developed by Unity Technologies in order to allow everyone to create attractive 3D environments, its creation was aimed at creating games. Unity3D possible to develop software for a wide range of platforms [12–14], so it is extremely attractive for a wide range of developers. For the simulation of a system is considered: *(i) 3D design*, this is done with special or general CAD programs; at this stage in addition to the three-dimensional drawing of the installation (environment modeling) the kinematic and dynamic characteristics of robots and other mobile elements of the system are defined; *(ii) trajectories following*, movements, velocities and sequences are determined; and *(iii) simulation of all movements,* the possibilities at this stage of the installation are checked, errors are corrected, the interference is detected and design are optimized.

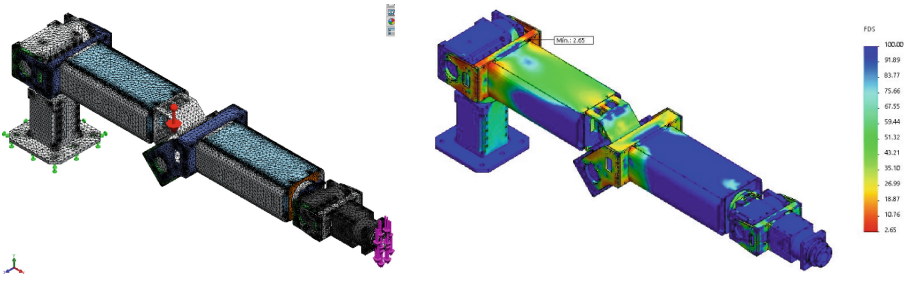
As mentioned above, this paper presents a new 3D virtual reality simulator for robotic applications. The proposed simulator allows real-time communication between Unity3D and MATLAB software. For bilateral communication it proposes to use shared memory between these two software; the method of shared memory is a technique easy to apply, with short delays and low use of computer resources by not calling functions third. In addition, the simulator allows to evaluate real-time

performance of different schemes of autonomous control and/or tele-operated in structured, partially structured environments, and unstructured; for tele-operation scheme 3D simulator accepts as input haptics devices that stimulate the senses of the human operator so that it can "transmit" their skill, experience and expertise to the robot to perform a task. Finally, to evaluate the performance of autonomous control simulator for monitoring roads proposed for a 6DOF robotic arm -system redundant-, as secondary objective is considered the maximum arm manipulability; also, the experimental test of a scheme bilateral tele-operation is performed.

This paper is divided into 6 Sections including the Introduction. In Sect. 2 the control problem is formulated. Next in Sect. 3 the modeling of the mobile manipulator robot and the controllers design for path following are presents. While the bilateral communication between MATLAB-Unity3D is present in Sect. 4. In Sect. 5 the experimental results for of autonomous control and tele-operated for a robotic arm are presented and discussed. Finally, the conclusions are given in Sect. 6.

2 Problem Formulation

The application development in the area of robotics requires accurately define the task to be performed to determine the needed characteristics of the robot. Determined these parameters, the execution of a task can be subdivided into the following steps: *(i) Modeling stage,* at this stage, it is essential to model the three-dimensional robot in a Computer Aided Design (CAD) software. The modeling lets to analyze the physical characteristics of the robot prior to the construction, in this context, there are tools such as SolidWorks, Autodesk Inventor, AutoCAD, among others, that allow to design mechanical elements and get results of mobility and strength of materials, among other mechanical characteristics, view Fig. 1;



(a) Gridding arm (b) Static analysis of the safety factor

Fig. 1. Ejemplo de un brazo robótico modelado en SolidWorks (Color figure online)

(ii) Construction stage, The main objective of this phase is to assemble each of the mechanical parts designed and incorporate the necessary electronics to move each joint, In addition to considering the signal conditioning for sensory perception of the external environment in which the robot will move, and internal sensors that must issue the

position, velocity, torque and strength of each link forming the robot. The information provided by the proprioceptive sensors and exteroceptive sensors will be used in the different advanced control algorithms proposed; *(iii) Controllers design stage,* to the design of advanced control algorithms it is essential to determine the mathematical model representing the robot kinematics and dynamics. The different mathematical models of robots are systems of multiple-input multiple-output, MIMO, so software tools that solve mathematical matrix operations to facilitate implementation of the proposed control scheme is required. MATLAB is a tool with its own programming language and development environment that offers the advantage of matrix manipulation and data processing. As a deployment scenario algorithms, MATLAB has libraries that can be extended according to programming needs [15]; *(iv) Simulation stage,* prior to the experimental implementation of the proposed control algorithms, it's necessary to check their performance in a three dimensional environment that emulates the actual conditions in which the robot operates, therefore, virtual development tools are required with the ability to support bilateral haptic devices, video output interfaces and audio, among others. Unity 3D is a tool for creating games, as well as development of virtual simulation and allows the incorporation of different haptic devices for manipulating its environment. Unity engine uses a script in C# language to manipulate the game objects with which you can modify the behavior of the simulated objects; and finally the *(v) Implementation stage,* it is the final phase in which the robot interacts with the environment where performs the task, this interaction is controlled via algorithms proposed control. The successful implementation of the planned task is based on compliance with each of the objectives of the above detailed steps; in this context, one can say that the design and simulation of control algorithms are the most critical stages for performing a task, so this work focuses on these two items.

In order to check the performance of control schemes in simulated/emulated environments, it is necessary to implement a communication channel between a bilateral graphics engine and mathematical software tool. The exchange of information in real time between the two applications is essential because the advanced control algorithms require the robot's feedback to close the control loop, while the emulated robot updates its kinematic and dynamic parameters depending on the robot-environment interaction, view Fig. 2.

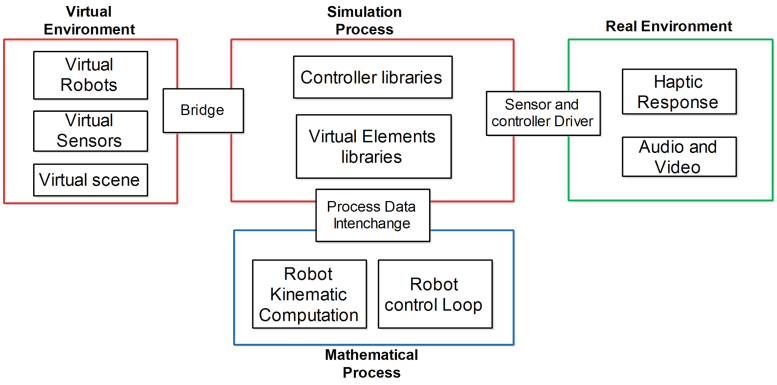


Fig. 2. Data interchange between Math software and 3D simulation software

In this context, the following sections show an emulator of advanced control algorithms implemented in MATLAB and displayed in real time in a virtual environment developed in Unity3D. It should be noted that the emulator allows bilateral interaction between MATLAB and Unity3D for any scheme or control technique implemented in MATLAB. As an example, autonomous control for tracking paths of a robotic arm 6 DOF is presented; and a tele-operated control from Unity's robotic arm.

Remark 1. A simulator represents reality in a similar way, while an emulator replica or improved conditions similar to the real ones.

3 Modeling and Control

The *instantaneous kinematic model of a robotic arm* sets the derivative of its location as a function of the derivative of its configuration (or its *operational velocities* as functions of its *generalized velocities*)

$$\dot{\mathbf{h}}(t)=\mathbf{J}(\mathbf{q})\dot{\mathbf{q}}(t). \tag{1}$$

It uses the Jacobian matrix $\mathbf{J}(\mathbf{q})$ of the function f: $\mathbf{J}(\mathbf{q}) = \frac{\partial f}{\partial \mathbf{q}}$. The configurations such that the rank of $\mathbf{J}(\mathbf{q})$ decreases are singular kinematic configurations and the problem, robotic arm and task, is redundant when $n > m$.

The mathematic model that represents the dynamics of a robotic arm can be obtained from Lagrange's dynamic equations, which are based on the difference between the kinetic and potential energy of each of the joints of the robot (energy balance). Most of the commercially available robots have low level PID controllers in order to follow the reference velocity inputs, thus not allowing controlling the voltages of the motors directly. Therefore, it becomes useful to express the dynamic model of the robotic arm in a more appropriate way, taking the rotational and longitudinal reference velocities as the control signals. To do so, the velocity controllers are included in the model. The dynamic model of the robotic arm, having as control signals the reference velocities of the system, can be represented as follows,

$$\mathbf{M}(\mathbf{q})\dot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) = \dot{\mathbf{q}}_{\mathbf{ref}} \tag{2}$$

where, $\mathbf{M}(\mathbf{q}) = \mathbf{H}^{-1}(\bar{\mathbf{M}} + \mathbf{D})$, $\mathbf{C}(\mathbf{q},\dot{\mathbf{q}}) = \mathbf{H}^{-1}\left(\bar{\mathbf{C}} + \mathbf{P}\right)$, $\mathbf{g}(\mathbf{q}) = \mathbf{H}^{-1}\,\bar{\mathbf{g}}(\mathbf{q})$. Thus, $\bar{\mathbf{M}}(\mathbf{q}) \in \Re^{\delta_n \text{ x } \delta_n}$ is a positive definite matrix, $\bar{\mathbf{C}}(\mathbf{q},\mathbf{v})\mathbf{v} \in \Re^{\delta_n}$, $\bar{\mathbf{G}}(\mathbf{q}) \in \Re^{\delta_n}$ and $\dot{\mathbf{q}}_{\mathbf{ref}} \in \Re^{\delta_n}$ is the vector of velocity control signals, $\mathbf{H} \in \Re^{\delta_n \text{ x } \delta_n}$, $\mathbf{D} \in \Re^{\delta_n \text{ x } \delta_n}$ and $\mathbf{P} \in \Re^{\delta_n \text{ x } \delta_n}$ are constant symmetrical diagonal matrices, positive definite, that contain the physical parameters of the robotic arm, *e.g.,* motors, velocity controllers.

In the other hand, a trajectory will be automatically generated and a trajectory tracking control will guide the robotic arm to the desired target. As indicated, the fundamental problems of motion control of robots can be roughly classified in three groups: (1) *point stabilization*: the goal is to stabilize the robot at a given target point, with a desired orientation; (2) *trajectory tracking*: the robot is required to track a time parameterized reference; and (3) *path following*: the robot is required to converge to a

path and follow it, without any time specifications. For more details about the modeling and control see [16].

4 Bilateral Communication MATLAB-Unity3D

This section describes the methods of inter-process communication and exchange of information between MATLAB and Unity3D for control of an emulated manipulator with a haptic device.

4.1 Windows Inter-process Communication (IPC)

IPC is a feature enabled in the operating systems on which processes can exchange information through memory segments or through own communication tools, allowing resource sharing. Generally these processes are developed to low level – allowing to interact with the operating system resources – and according to the protocols for such communication.

Table 1. Windows Inter-process communication methods [17]

Method	Advantages	Disadvantages	Resources
Named Pipe	Easy to use and works across the network	The source code is platform dependent	Medium
WinSock	Works on the same computer as well as across networks. Moreover, it can be used across various platforms and protocols	Requires a knowledge of relatively advanced networking concepts	Low
Mailslots	Works across a network and supports broadcast	Provides one-way communication only	Medium
Shared Memory	**Linking processes using memory registers previously allocated, without functions of third party**	**Works on the same computer**	**Low**

The techniques to develop IPC vary depending on the application. This function can be used for the transmission of messages, synchronization, shared memory and remote procedures. The method used to communicate processes depends on the transfer rate required and the type of data to be treated. There are several ways to implement communication between processes, among which are: *(i) Named Pipe* is a method of channeling data by creating a memory space in the operating system explicitly declared before the execution of processes to communicate.; *(ii) WinSock* provides very high level networking capabilities, it supports TCP/IP (the most widely used protocol) along with many other protocols like AppleTalk, DECNet, IPX/SPX, etc.; *(iii) Mailslots* processes messages between applications via datagrams and allows to communicate unidirectionally, this method is useful for transmitting information to multiple clients;

and finally, *(iv) Shared memory* allows to create segments of memory to be accessed by multiple processes, access restrictions may be defined, *e.g.*, read only, read and write, execute, access over inheritance, among others. Table 1 presents the differences between the methods described for implementation of inter-processes communication.

Remark 2. Datagram is a data set of the communication protocol packet switched used to route information between nodes in a network.

In reference to illustrated in Table 1 and the proposed implementation guidance in this work, the method of *shared memory* is a technique easy to apply, with short delays and low use of computer resources by not calling third party functions.

4.2 MATLAB – Unity Communication

The bidirectional data communication between MATLAB - Unity3D is performed by a dynamic-link-library, dll, in which the Shared Memory method is implemented, SM, in RAM. The Fig. 3 illustrates the implementation of shared memory, where the dll manages the SM space, besides providing permits for the applications, label the memory space, provide functions to modify/obtain the stored information and liberate the space when the application is terminated.

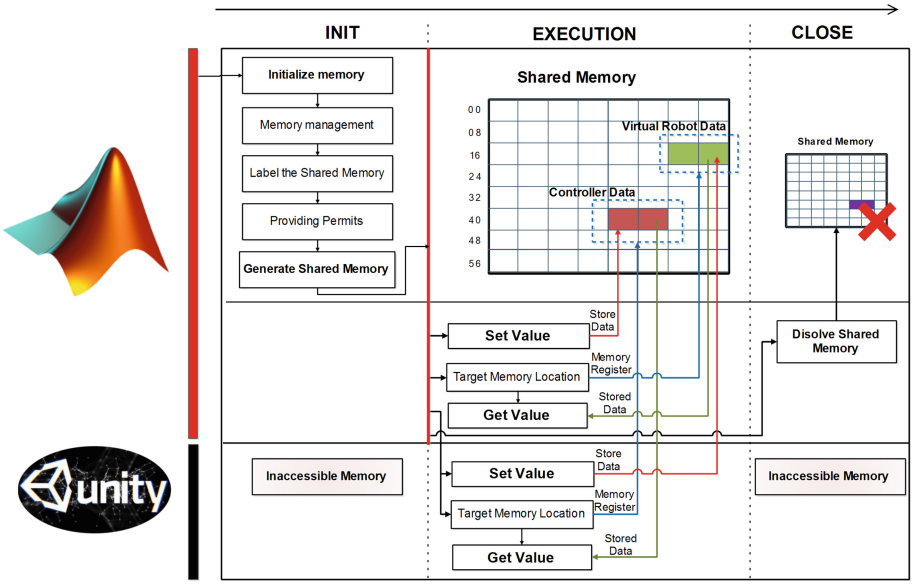


Fig. 3. Interprocess communication via shared memory

Using the dll between MATLAB and Unity3D is divided into three parts: ***(i) Init phase,*** the dll can be instantiated from an application through a handle, in which are set the security attributes and inheritance, permissions to read/write to memory registers reserved, RAM's space management and labeling. In this way, the client applications

can reference data wishing to modify or capture, provided it have access permissions, aware of their existence and location where it is staying.

Remark 3. The characteristics of the dll allows that the generated memory can be started from MATLAB or Unity3D. From this step, both applications must use functions to identify dedicated spaces of memory, modify data or get them.

```
void CreateSharedMemoryArea()
{
        hFile = CreateFileMapping(INVALID_HANDLE_VALUE,        //Handle to instantiate the dll
created
                NULL,                                          //Null Security attributes and heritage
                0x40,                                          //Read&Write permissions
                0,
                1024 * 4,                                      //Memory Space Managed
                _T("memoriaza"));                              //Label the shared memory
        if (hFile == NULL)                                     // Memory Validation
        {
                printf("Unable to create a file.");
                exit(1);
        }
}
```

(ii) Execution phase, at this stage MATLAB and Unity3D must invoke the function *OpenSharedMemory()* to find the handle through the label and create a memory view, defining the read/write permissions, the point where the view begins and the number of bytes to be mapped, view Fig. 4. The view points to handle, from which a casting to LPINT type variables is made to locate the index of each of the stored data. The handle is referenced by the view when the application desires to read or write in memory.

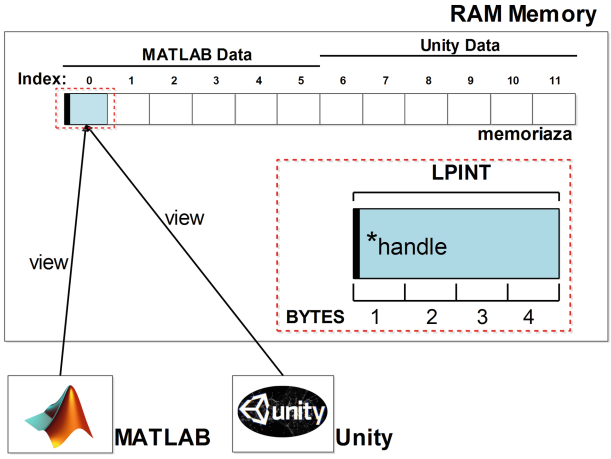


Fig. 4. Views of the Shared Memory

```
    void OpenSharedMemory()
    {
      hFile = OpenFileMapping(FILE_MAP_ALL_ACCESS,                    // Search tag memory
                              FALSE,
                              _T("memoriaza"));

            if (hFile == NULL)                                        // Validate the existence of
the memory
            {
                    printf("Unable to open the shared area.\n");
                    exit(1);
            }
            hView = (LPINT) MapViewOfFile(hFile,                      // Creating view type LPINT
                    FILE_MAP_ALL_ACCESS,                              // Read / Write definition
                    0,                                                // Point where the view
begins
                    0,
                    0);                                               // All memory map

            if (hView == NULL)                                        // View Validation
            {
                    printf("Unable to create a VIEW.\n");
                    exit(1);
            }
            aux = hView;                                              // Handle referenced to
Read or Write registers
    }
```

The view allows update dedicated registers of each application. For writing data, the application is based on indexes that the dll provides in static manner for each of the variables.

```
    void WriteOnSharedMemory(int data, int position)                  // data defines the value,
position defines the index
    {
            aux[position] = data;
}
```

```
    void ReadFromSharedMemory(int *data, int position)                // data defines the value,
position defines the index
    {
            *data = aux[position];
}
```

Finally ***(iii) Close phase,*** SM is reserved while the process runs. When the application *Close*, memory must be released. By invoking the function DestroySharedMemoryArea(), ends with the reservation and labeling of RAM for that any other system process can use it.

```
    void DestroySharedMemoryArea ()                    // Free the shared memory
    {
            if (!UnmapViewOfFile(hView))
            {
        printf("Could not unmap view of file.");
            }
            CloseHandle(hFile);
            printf("The end.\n");
}
```

4.3 Interaction MATLAB-Unity3D

The interaction between MATLAB and the graphics engine is divided into three stages, described in the following paragraphs as: import of three-dimensional design, interaction human-robot and bilateral communication processes (Fig. 5).

SolidWorks is the CAD tool used for mechanical design, but has no export formats supported by the virtual tool development. In the first phase, 3DS Max is used to modify parameters SolidWorks 3D modeling and hierarchies are established in the pieces that make up the assembly and supported file by Unity3D is exported. Once the three-dimensional model imported to Unity 3D environment, texture for each piece that makes up the prototype are established. In addition, the degrees of freedom of the 3D model is specified by activating the points of rotation and/or translation for objects that guides each.

In the second stage, the Unity3D environment performs the animation of virtual objects using Game Objects, scripts and plugins. The behaviour of Objects Game is controlled by scripts, which allow you to modify its properties and respond to user input as scheduled. The plugins allow you to use native functionality (support Oculus Rift) or include external code (support Novint Falcon). The human-robot interaction is achieved by information from input devices (Falcon encoders, Tracking Oculus HMD), the mathematical tool uses this data to return control actions and generate output responses (Falcon motors, Oculus HMD and audio).

Finally, in the third stage, information virtual robot is linked to MATLAB through the dll file and the invocation of the SM. When MATLAB requires send or retrieve information from the SM sector, its programming should include the lines:

```
    loadlibrary('./dll64MATLAB.dll','./simple.h');                    // Invoking the dll
    calllib ('dll64MATLAB','initMemory');                             // Initialize memory
    calllib ('dll64MATLAB','openMemory');                               // Create the view of the
shared memory
    calllib ('dll64MATLAB','setValue',v1,v2,v3,v4,v5,v6,v7,v8,v9,v10);              // Set values in the
memory
    val1 = calllib ('dll64MATLAB','getValue1');                              // Get values  from
memory
    calllib ('dll64MATLAB','destroy');                                // Free shared memory
```

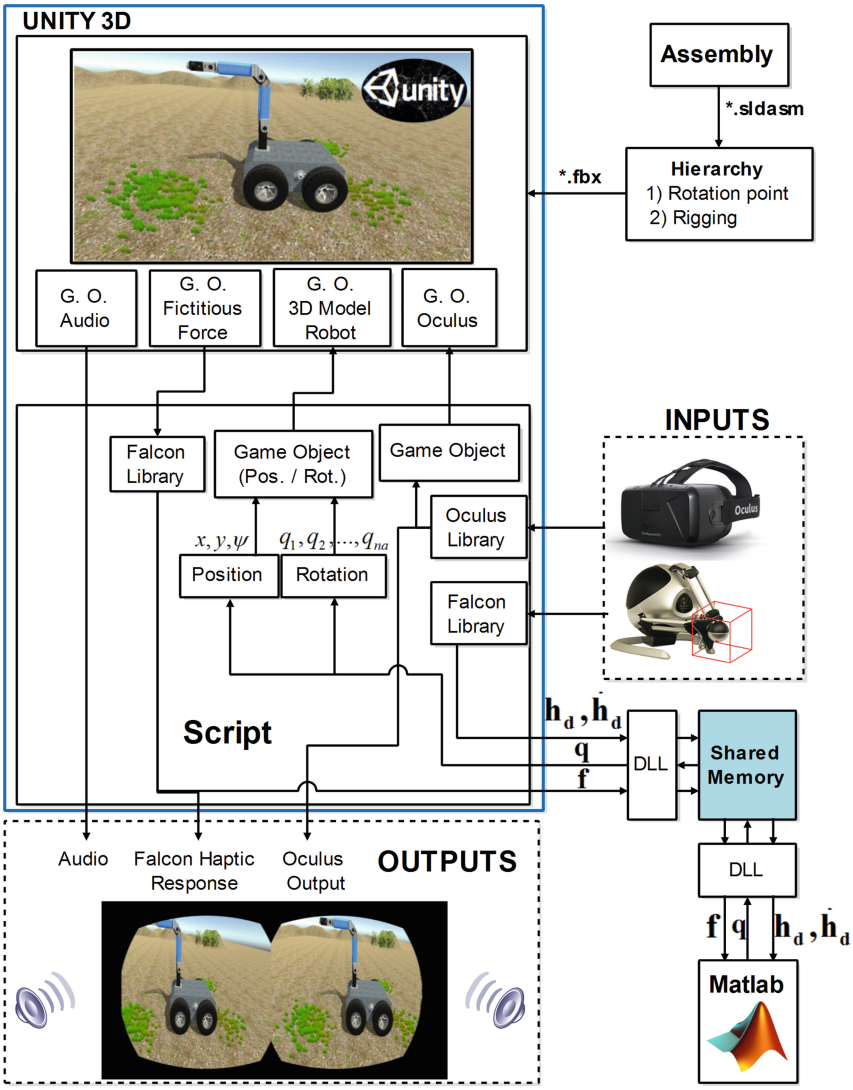


Fig. 5. Unity3D – MATLAB Interaction

While in each communication cycle, Unity receives velocities to control the rotation points and thus the operative end. Additionally, is sent the position of each actuator forming the modeled 3D robot and data human-robot, such as real interaction forces given by contact, effects of gravity, fictitious forces to avoid virtual obstacles, etc. Unity within the script, must contain the following lines of programming to invoke the read/write data in the SM.

```
    [DllImport(@"C:\Users\Public\Documents\Unity
Projects\Animación\Assets\Plugins\dll64MATLAB.dll")]
        private static extern void openMemory();                    // Create the view of the
shared memory
    [DllImport(@"C:\Users\Public\Documents\Unity
Projects\Animación\Assets\Plugins\dll64MATLAB.dll")]
        private static extern int getValue1();                      // Get values  from
memory
    PosX = -getValue1() / 100;                                      // Using the get value

    [DllImport(@"C:\Users\Public\Documents\Unity
Projects\Animación\Assets\Plugins\dll64MATLAB.dll")]
        private static extern int setValue(int v1, int v2,..., int v10);
    setValue(val1,val2,...,val10);                                  // Set values in the
memory
```

Remark 4: The libraries developed for information sharing allow interaction between Unity Game Objects with any software package of MATLAB like Script, Simulink, etc., once initialized the SM.

Remark 5: In the case of robotic applications, update time data is relatively low due at time of sampling used. This work do not try to raise synchronization methods of information in shared memory.

5 Simulation Experimental Results

In order to illustrate the performance of the proposed simulator 3D of an arm robotic 6DOF, several experiments were carried out for path following autonomous control and bilateral tele-operation of a robotic arm; the most representative results are presented in this Section. The experiments were carried with the kinematic and dynamic models of a robotic arm 6 DOF, view Fig. 6.

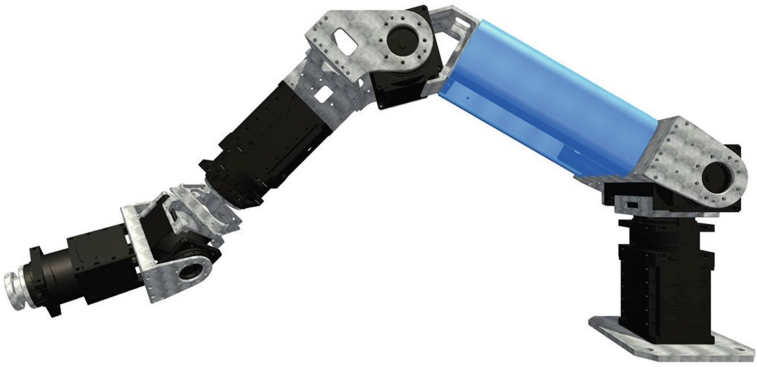


Fig. 6. Arm Robotic 6DOF developed in SolidWorks

On the other hand, the proposed simulator 3D consist in a desktop computer Core I7 3610QM running at 2.3 GHz, 8 GB RAM and a NVIDIA GeForce GT 630 M 1 GB dedicated graphics; also the local site has a haptic device Falcon^{MT} Novint. The evaluation of the latency in MATLAB takes into account the transmission, execution Unity and receiving data delay, which is within the desired sampling period of 100 [ms].

5.1 Autonomous Control

The performance of the control structure for path following is tested. The desired trajectory for the end-effector is described by $P(s) = (x_P(s), y_P(s), z_P(s))$, where $x_P = 0.35\sin(0.2s + \frac{\pi}{2})$; $y_P = 0.35\cos(0.2s + \frac{\pi}{2})$ and $z_P = 0.2 + 0.8\sin(0.1s)$. Note that for the path following problem, the desired velocity of the end-effector of the robotic arm will depend on the task, the control error, the joint velocity of the arm, among other design specifications. In this experiment, it is considered that the reference velocity module depends on the desired velocity of the end-effector on path P and the control errors. Then, reference velocity in this experiment is expressed as $|\mathbf{v_{hd}}| = \frac{v_P}{1+k\|\tilde{\mathbf{h}}\|}$, where k is a positive constant that weigh the control error module. Also, the desired location is defined as the closest point on the path to the end-effector of the experimental system.

Hence, Figs. 7, 8, 9 show the results of the experiment of autonomous control. Figure 7 shows the stroboscopic movement on the X-Y-Z space of Unity3D. It can be seen that the proposed controller works correctly; while the Fig. 8 shows the desired path and the current path of the end-effector of the robotic arm. It can be seen that the proposed controller presents a good performance;

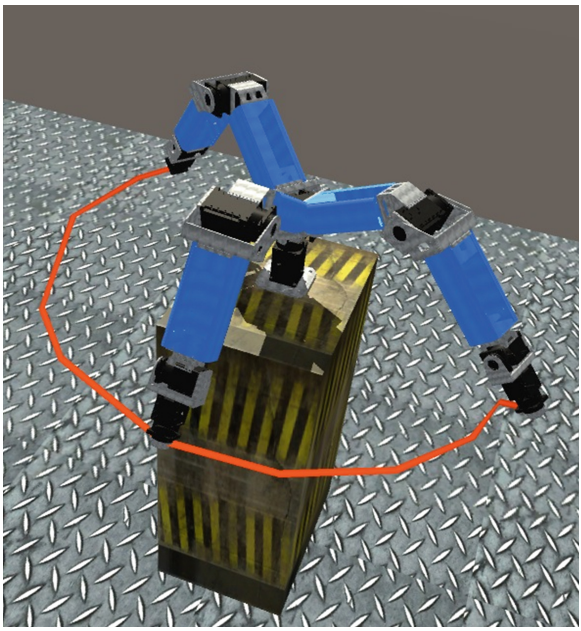


Fig. 7. Stroboscopic movement of the robotic arm in Unity 3D.

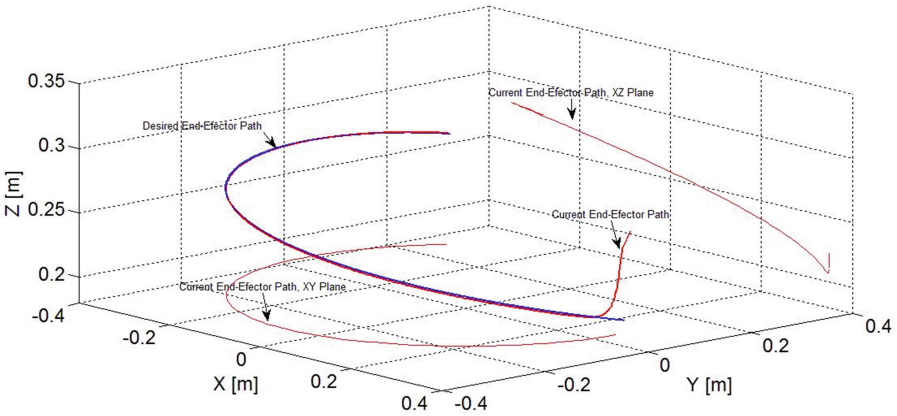


Fig. 8. Desired path and the current path of the end-effector

and finally Fig. 9 shows that the control errors of the robotic arm on the *X*-*Y*-*Z* space converge to values close to zero asymptotically.

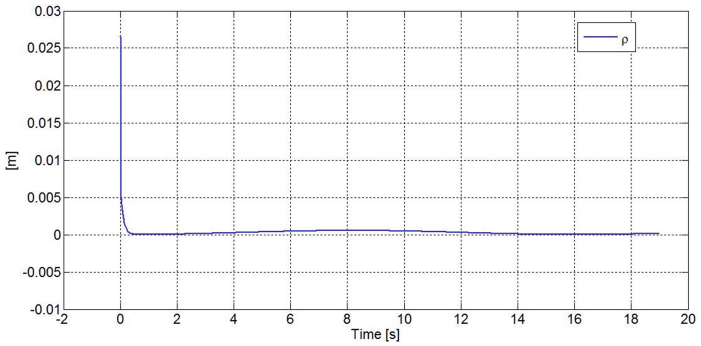


Fig. 9. Distance between the end-effector position and the closest point on the path

5.2 Tele-Operated Control

The feasibility of the proposed simulator 3D is tested through of bilateral tele-operation scheme using a robotic arm 6 DOFs. The local site has an Oculus and a haptic device Falcon^{MT} Novint. The human operator commands are generated with the use of a Falcon^{TM} haptic device from Novint Technologies Incorporated [18] as indicated in Fig. 10. Its positions are translated into desired velocities commands $P(s) = (x_P(s), y_P(s), z_P(s))$ of the end-effector of the robotic arm [19].

The simulation of a bilateral tele-operation scheme is presented, which consists on a grasping task. With this aim, the robot is guided near the object; then the user grasps the object opening the gripper; and finally the robot is guided to drop the object into a box. Obtained results are shown in Figs. 11, 12, 13. Figure 11 shows snapshots of the

Fig. 10. Local site of the tele-operation scheme

experiment on Unity 3D. Figure 12 shows a comparison between the reference generated by the human operator and the actual velocities of the end-effector. While Fig. 13 depicts the time evolution of the control error of the robotic arm.

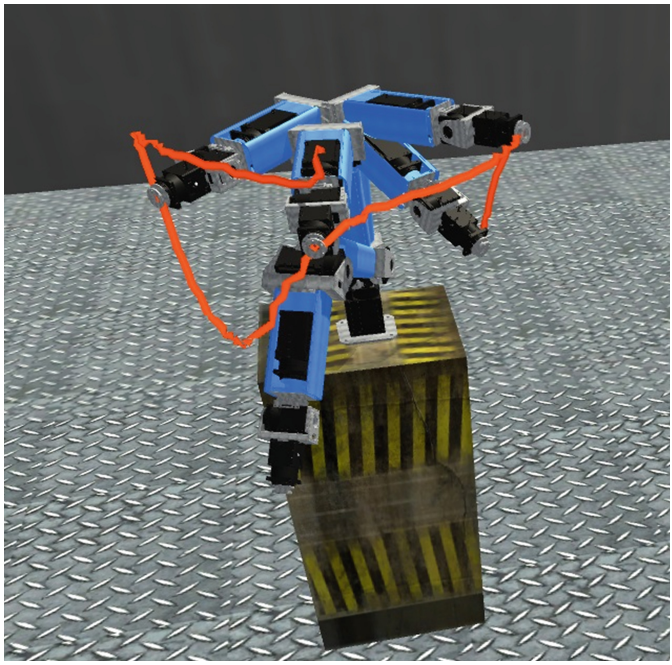


Fig. 11. Bilateral tele-operation: Grasping task on Unity 3D

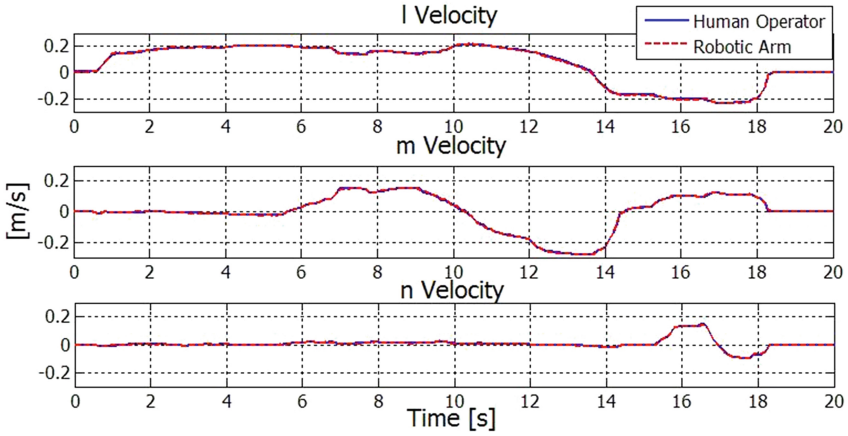


Fig. 12. Comparison between the reference generated by the human operator and the actual velocities of the end effector

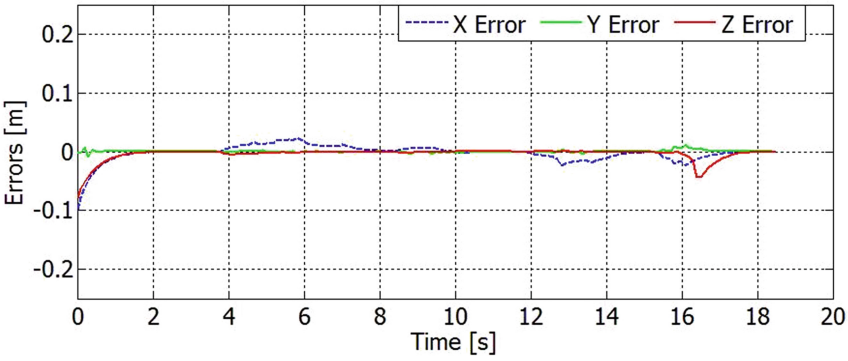


Fig. 13. Evolution of the control errors of the robotic arm. If $\lim_{t\to\infty} \tilde{x}(t) = 0$, $\lim_{t\to\infty} \tilde{y}(t) = 0$ and $\lim_{t\to\infty} \tilde{z}(t) = 0$ then $\lim_{t\to\infty} \rho(t) = 0$.

6 Conclusions

In this paper a 3D simulator in real time for robotics applications is proposed. This simulator considers the bilateral communication between MATLAB-Unity3D through of a dynamic-link-library, dll, in which the method of Shared Memory in RAM is implemented. The dll manages space SM, enable permissions to applications, puts the label to memory space, provide functions to modify/obtain the stored information and freeing the space when the application is terminated. Experimental results were also presented showing the feasibility and the good performance of the proposed simulator 3D; the experiments were carried out for path following autonomous control and bilateral tele-operation of a robotic arm 6DOF.

Acknowledgment. The authors would like to thanks to the Consorcio Ecuatoriano para el Desarrollo de Internet Avanzado -CEDIA-, Universidad de las Fuerzas Armadas ESPE, Universidad Técnica de Ambato and the Escuela Superior Politécnica del Chimborazo for financing the project *Tele-operación bilateral cooperativo de múltiples manipuladores móviles – CEPRAIX-2015-05*, for the support to develop this paper.

References

1. Andaluz, V.H., López, E., Manobanda, D., Guamushig, F., Chicaiza, F., Sánchez, J.S., Rivas, D., Pérez, F., Sánchez, C., Morales, V.: Nonlinear controller of quadcopters for agricultural monitoring. In: Bebis, G., et al. (eds.) ISVC 2015. LNCS, vol. 9474, pp. 476–487. Springer, Heidelberg (2015). doi:10.1007/978-3-319-27857-5_43
2. Andaluz, V.H., Chicaiza, F.A., Meythaler, A., Rivas, D.R., Chuchico, C.P.: Construction of a quadcopter for autonomous and tele-operated navigation. In: IEEE-DCIS Conference on Design of Circuits and Integrated Systems, Portugal (2015)
3. Andaluz, V.H., Canseco, P., Varela, J., Ortiz, J.S., Pérez, M.G., Roberti, F., Carelli, R.: Robust control with dynamic compensation for human-wheelchair system. In: Zhang, X., Liu, H., Chen, Z., Wang, N. (eds.) ICIRA 2014, Part I. LNCS, vol. 8917, pp. 376–389. Springer, Heidelberg (2014)
4. Andaluz, V.H., Ortiz, J.S., Roberti, F., Carelli, R.: Adaptive cooperative control of multi-mobile manipulators. In: IEEE–IECON Industrial Electronics Society, pp. 2669–2675, USA (2014)
5. Andaluz, V.H., Roberti, F., Marcos, T.J., Ricardo, C.: Adaptive unified motion control of mobile manipulators. J. Control Eng. Pract. 1337–1352 (2012). Elsevier Editorial System
6. Andersen, R.S.; Bogh, S.; Moeslund, T.B.; Madsen, O.: Intuitive task programming of stud welding robots for ship construction. In: 2015 IEEE International Conference on IEEE Industrial Technology (ICIT), pp. 3302–3307, March 2015
7. Andaluz, V.H., Ortiz, J.S., Sanchéz, J.S.: Bilateral control of a robotic arm through brain signals. In: De Paolis, L.T., Mongelli, A. (eds.) AVR 2015. LNCS, vol. 9254, pp. 355–368. Springer, Heidelberg (2015)
8. Ying, J.L., Peng, J.S., Qi, Z., Chang, C.L., Yong, H.: The review of workpiece loading and unloading robot in the catenary shot blasting. In: Research and Design of Machinery, Equipment and Technological Processes in Mechanical Engineering, Applied Mechanics and Materials, Vols. 496–500, pp. 578–581 (2014)
9. Freund, E., Rossmann, J.: Proyetive virtual reality: Bringing the gap between virtual reality and robotic. IEEE Trans. Robot. Autom. **15**(3), 411–422 (1999)
10. Brunet, P., Vinacua, A.: Sistemas Gráficos Interactivos. Universidad Politécnica de Cataluña, Barcelona, España, Mayo de 2006. http://www.lsi.upc.edu/∼pere/SGI/guions/ArquitecturaRV.pdf
11. Meyer, J., Sendobry, A., et. al.: Comprehensive simulation of quadrotor UAVs using ROS and Gazebo. In: SIMPAR 2012 Proceedings of the Third International Conference on Simulation, Modeling, and Programming for Autonomous Robots, Berlin, Germany, vol. 12, pp. 400–411 (2015)
12. Oliveira, M., Pereira, N., Oliveira, E., Almeida, J.E., Rossetti, R.J.: A Multi-player Approach in Serious Games: Testing Pedestrian Fire Evacuation Scenarios. Oporto, DSIE15, January (2015)

13. Bartneck, C., Soucy, M., Fleuret, K., Sandoval, E.B.: The robot engine—Making the unity 3D game engine work for HRI. In: 2015 24th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN), pp. 431–437. IEEE (2015)
14. Indraprastha, A., Shinozaki, M.: The investigation on using Unity3D game engine in urban design study. J. ICT Res. Appl. **3**(1), 1–18 (2009)
15. MATLAB and Simulink for Technical Computing. http://www.mathworks.com/
16. Andaluz, V., Salinas, L., Roberti, F., Toibero, J., Carelli, R.: Switching control signal for bilateral tele-operation of a mobile manipulator. In: 2011 9th IEEE International Conference on Control and Automation (ICCA), Santiago, Chile, 19–21 December 2011
17. Interprocess Communications. https://msdn.microsoft.com/en-us/library/windows/desktop/aa365574(v=vs.85).aspx
18. Andaluz, V., Roberti, F., Toibero, J., Carelli, R.: Adaptive unified motion control of mobile manipulators. Control Eng. Pract. **20**(12), 1337–1352 (2012)
19. Martin, S., Hillier, N.: Characterization of the novint falcon haptic device for application as a robot manipulator. In: Australasian Conference on Robotics and Automation (2009)