Towards a Generic Simulator for Continuum Robot Control*

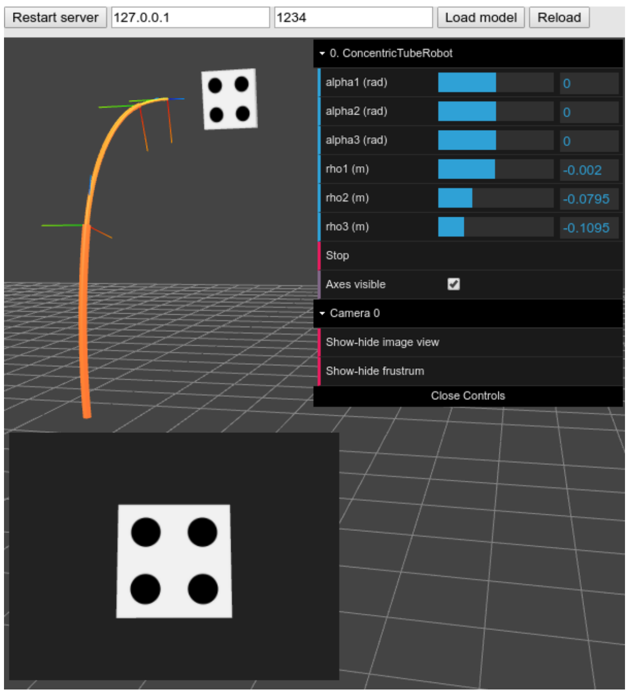
Andrey V. Kudryavtsev[1], Kanty Rabenorosoa[1], and Brahim Tamadazte[1]

I. INTRODUCTION

Nowadays, it became extremely difficult to deny the importance of continuum robotics, especially in medical applications [1]. In contrast to classical industrial robots, that are typically characterized by a series of discrete rigid links, continuum robots (CR) have a particular structure going from robotic arm inspired by elephant trunk to concentric tube robots (CTR). However, there is a common property which defines them as an actuated mechanism whose backbone forms curves with continuous tangent vectors. For developing advanced control and achieving complex tasks with CR, a generic simulator would be helpful. Indeed, building a realistic experimental platform containing continuum robots may be a challenging task both in terms of time and cost. One of the most popular CR for medical applications is CTR that is obtained by assembling elementary precurved tubes with different diameters in telescopic manner. Generally, these tubes are made of Nickel Titanium alloy but more recently 3D printed ones were introduced [2]. The tube assembly constitutes the effective part of the CTR and in order to get a functional robot, it is necessary to have an actuation unit including angular and linear stages. Various challenges remain to be tackled to obtain lightweight, compact, and high degrees of freedom actuation unit. Therefore, the core of this paper is the development of a generic simulation platform for CR, especially a CTR. Comparing to widely-used simulation platforms such as V-Rep or Gazebo, our simulator has an important advantage: it is possible, and easy, to add new deformable robots. Actually, the simulator can be used, for instance, to optimize the CTR mechanism design, to develop and validate advanced controls by using various feedbacks, and to achieve complex tasks by integrating anatomical models, etc. In the reminder of this paper, at first the software architecture is presented. Secondly, we discuss the capabilities of the simulator from the user point of view, i.e., what a person with no background in programming can achieve with it.

II. SOFTWARE ARCHITECTURE

The simulator presented here is written mostly in JavaScript (JS). JS is a web-oriented programming language supporting object-oriented paradigm. Inspite of the fact that JS is generally used on web-pages, since 2009 it became



127.0.0.1
1234
alpha1 (rad) 0
alpha2 (rad) 0
alpha3 (rad) 0
rho1 (m) -0.002
rho2 (m) -0.0795
rho3 (m) -0.1095
Axes visible ✓

Fig. 1. Main window screenshot of the simulator displaying: a 3D scene with CTR, target object, and an image coming from the virtual camera installed on the end-effector.

possible to develop stand-alone desktop applications thanks to two following elements:

- runtime environment called *Node.js* which allows executing JS code without a browser;
- framework *Electron* that made possible building desktop applications using web-development technology (HTML, CSS, and JS).

The fact of using an object-oriented structure adds very interesting features to our simulator. In particular, in order to add a new robot, one has only to create a new class containing the robot forward kinematics model.

III. USER-SIMULATOR INTERACTION

After the application starts, a user will see the main window displayed in Fig. 1. It contains the following elements:

- 3D scene containing CTR and a target object (white box with four black dots) in front of it.
- a menu allowing to change some parameters of the scene. In the given example, one can modify joint variables (mouse-based control) associated with the CTR;
- controls of server parameters defining the IP address and port number allowing the communication of simulator

*This work has been supported by the Labex ACTION project (contract ”ANR-11-LABX-0001-01”) and ANR NEMRO (contract ”ANR-14-CE17-0013”).

[1]Authors are with FEMTO-ST Institute, Univ. Bourgogne Franche-Comté, CNRS, Besançon, France. brahim.tamadazte@femto-st.fr (corresponding author)

TABLE I

EXAMPLES OF COMMANDS THAT CAN BE SENT TO THE SIMULATOR.

Command	Target
`SETJOINTVEL,0,0,0,0,0,0.001`	robot
`SETJOINTPOSREL,0,0,0,0,0,0.001`	
`GETJOINTPOS`	
`STOP`	
`GETIMAGE`	camera
`GETCALIBMAT`	

with the external world (external controller, software, etc.) via UDP communication;

- buttons for scenes management (loading a new scene or reloading the current one).

Also, the simulator gives the possibility to place a virtual camera either on the robot end-effector or freely in 3D space. This gives the possibility to generate a virtual image with known and chosen camera model (e.g., intrinsic parameters). This last point is of particular interest for simulating vision-based control schemes such as visual servoing [3] or automatic intracorporeal navigation [4].

A. Constructing the Scene

The 3D scene of the simulator must be defined in advance by a user and can contain three types of objects. Firstly, it may contain different static objects that are loaded from standard 3D-model files such as *.stl, *.obj or *.dae files. Secondly, it is a robot with its joint values that are directly accessible from the menu in the right-up corner. A scene may contain several robots. Finally, the third type of components concerns imaging systems such a standard white-light camera which allows generating images that can be then acquired using UDP protocol. The whole scene is defined using a text file in *json* format[1].

B. Robot Simulation

During simulation, a user has access to two tools allowing the interaction with the scene. The first one is the menu that allows some basic manipulations such as changing joint values, displaying or hiding frames associated with robot joints, etc.

The second tool is a UDP protocol. Actually, the simulator is a UDP server that can send and receive messages from external world. Among current possibilities, a user can change robot configuration (relative or absolute joint position), apply joint velocities, get the transformation matrix corresponding to the tool pose, acquire the image from a virtual camera, etc. Some examples of commands are represented in Table I.

There is a variety of ways for sending these UDP messages to the simulator. The easiest one is to use a small software *telnet* which allows sending UDP messages directly from terminal. Another way would be to use a *telnet* equivalent software with a graphic user interface. Finally, it is always possible to establish UDP communication using different

[1]for details, see https://github.com/avkudr/visa/wiki/Scene-modeling

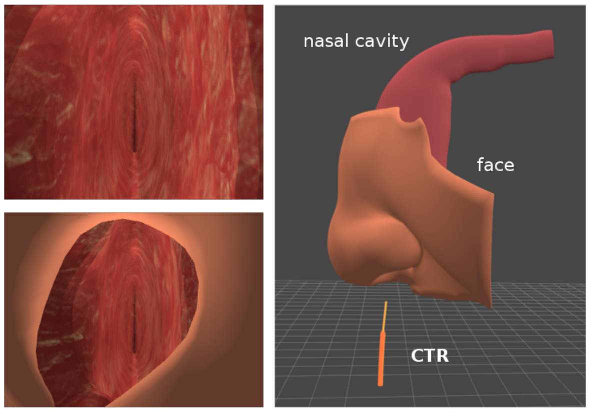


Fig. 2. Simulating a scene containing a 3D face with a nasal cavity, and a CTR. Images on the right generated by the camera located on the robot tip.

programming languages: python, MATLAB, etc. It is worth noticing, that an adapter for C++ communication is provided along with the simulator. This adapter, while acting like a UDP client, allows in addition to transform the image coming from simulator directly to OpenCV or ViSP libraries format.

C. Example of use

In this subsection, we present an example of a complete simulated scene (Fig. 2). The scene contains a part of human face as well as the 3D model of nasal cavity. Both objects were loaded from *.stl files. CTR is placed in a way allowing entering the cavity and performing an internal navigation. In the given example, a CTR model is based on piecewise constant curvature model. In addition, a camera is located on the robot tip which enables online generation of images. This setup can be used for development and testing of various visual servoing and/or SLAM algorithms.

IV. CONCLUSION

The goal of this work was to create a simulation environment for continuum robots. Nowadays, it contains already a concentric tube robot build using constant-curvature model. Moreover, thanks to simulator flexibility of adding new robots, one can use the existing pattern to build other robots and add them to 3D scene. As user can also acquire the image given by the virtual camera in real time, it opens new possibilities for testing such algorithms as visual servoing, SLAM or any other technique based on a visual feedback.

REFERENCES

[1] Burgner-Kahrs, J., Rucker, D. C., & Choset, H. (2015). Continuum robots for medical applications: A survey. IEEE Transactions on Robotics, 31(6), 1261-1280.

[2] Morimoto, T. K., & Okamura, A. M. (2016). Design of 3-D Printed Concentric Tube Robots. IEEE Trans. Robotics, 32(6), 1419-1430.

[3] Baran Y., Rabenorosoa K., Laurent G.J, Rougeot P., Andreff N., Tamadazte B., Preliminary results on OCT-based position control of a concentric tube robot, IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), Vancouver, Canada, 2017.

[4] Kudryavtsev, A.V., Chikhaoui, M.T., Liadov, A., Rougeot, P., Spindler, F., Rabenorosoa, K., Burgner-Kahrs, J., Tamadazte, B., Andreff, N. (2018). Eye-in-Hand Visual Servoing of Concentric Tube Robots. IEEE Robotics and Automation Letters, 3(3), 2315-2321.