Hybrid force-vision control for da Vinci simulator

Flavio Lorenzi Michele Ciciolla

March 2020

Abstract

Recent technological developments in robotics have made possible to perform important tasks in medical field. This is the case of surgical robots (e.g. Da Vinci Robot, Mako Striker, ROBODOC) capable of reaching where man alone cannot. In this context it is usually necessary to guarantee short approaching times, stability, low impact forces, avoid any bounce between the end-effector and the surface to be reached.
To fully achieve this, it was thought to combine the efficiency of Visual Servoing control techique with a Force feedback. This led to a hybrid system.
Our work therefore consists of a first part of study and explanation on this very particular hybrid system (section 1 and 2), focusing on the importance of Visual Servoing and its limits. Then we have a second part of implementation: starting from a previous university work [8] we have implemented the inverse kinematics (section 3a), solving all the problems we have encountered with careful tuning of the parameters; finally we focused on minor tasks (section 3b), trying to refine our model.

1 Visual Servoing

The controller implemented in this project is a image-based visual servoing (IBVS) type which aims to generate and minimize a 2D error in the image space given the current image and the desired one. In doing so the robot should move so as to bring the current image to the desired one [4][5].
In this particular example the 'image' is represented as a feature vector s composed by 4 couples of pixel coordinates (u,v) of each landmark on the target skin on the table. The feature vector s is generated extracting the position of every landmark in the image plane of the camera sensor and saving its coordinates (u,v) according to the perspective camera model. This process is named feature extraction and in this particular project is done via Vrep functions [1].
Since the desired final image (s^d) is known, it's now possible to generate an image error associated to the current feature s as

$$e = s^d - s \tag{1}$$

A fundamental role in image servoing is the interaction matrix L that plays the same role of a classical robot Jacobian: it creates a relation between camera velocity $\dot{p}$ and pixel velocity $\dot{U}$ which is the following:

$$\dot{U} = L \cdot \dot{p} \tag{2}$$

The interaction matrix has this form:

$$L = \begin{pmatrix} \frac{f}{Z_1} & 0 & \frac{u_1}{Z_1} & \frac{u_1 v_1}{Z_1} & -\frac{f^2+u_1^2}{f} & v_1 \\ 0 & \frac{f}{Z_1} & \frac{v_1}{Z_1} & \frac{f^2+v_1^2}{f} & -\frac{u_1 v_1}{Z_1} & -u_1 \\ \frac{f}{Z_2} & 0 & \frac{u_2}{Z_2} & \frac{u_2 v_2}{Z_2} & -\frac{f^2+u_2^2}{f} & v_2 \\ 0 & \frac{f}{Z_2} & \frac{v_2}{Z_2} & \frac{f^2+v_2^2}{f} & -\frac{u_2 v_2}{Z_2} & -u_2 \\ & & & \vdots & & \\ \frac{f}{Z_n} & 0 & \frac{u_n}{Z_n} & \frac{u_n v_n}{Z_n} & -\frac{f^2+u_n^2}{f} & v_n \\ 0 & \frac{f}{Z_n} & \frac{v_{1n}}{Z_n} & \frac{f^2+v_n^2}{f} & -\frac{u_n v_n}{Z_n} & -u_n \end{pmatrix}$$

Figure 1: Interaction matrix [7].

It is clear that this is based on the fact that the user knows the focal lenght of the camera $\hat{f}$ and the 3D distance (depth) z of the object wrt the camera. In our work this latter parameter is extracted using a Vrep function and its value is than almost exact. Obviously in real projects this is not possible and in fact the interaction matrix is approximated or the z values is measured by an external sensor.

In the eye-in-hand configuration the camera and the end effector velocity are the same and so $\dot{p}$ is converted to joint velocity through inverse kinematics, but since our setup is an eye-to-end one the end effector velocity will be the opposite of the camera velocity velocity.

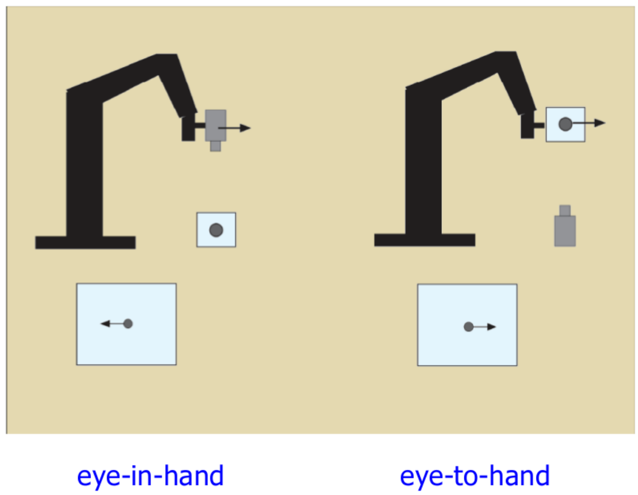

Figure 2: camera positioning types

From

$$\dot{U} = L \cdot \dot{p} \tag{3}$$

we derive

$$\dot{p} = L^{\dagger}\dot{U} \tag{4}$$

where $L^{\dagger}$ is the pseudo-inverse of L using the Matlab function pinv(L).
$\dot{U}$ is achieved via a proportional controller setting

$$\dot{U} = K(s^d - s) \tag{5}$$

and finally

$$\dot{p} = KL^{\dagger}(s^d - s) \tag{6}$$

to betaken as $-\dot{p}$ according to the eye-to-hand setting.
As explained above to compute the Interaction matrix we need to know the depth z of a point w.r.t the vision sensor. Since this approximation takes our experiment away from a practical situation, it's now functional to introduce a way to compensate this issue i.e. installing 'artificially' a force sensor on the daVinci end effector and incorporating it in the control loop we can fill the gap given by the lack of the depth knowledge.
That's why it's important to compensate the defects of Visual Servoing using the force sensing.

2 The Hybrid Force-Vision Control

The *main paper* [7] introduces a Control Scheme composed by two main branches: the Vision Control Law (VCL) in green and the FORCE CONTROL LAW (FCL) in red.
The former takes in input the current feature vector (s) coming from the sensor, and the desired 'updated' feature (s*) coming from the FCL. Is then computed the image error (s*- s) which leads to task velocities τ (we named this $\dot{p}$ in the previous chapter) and then joint velocities via the inverse differential kinematics.
The latter branch deals with the force sensor artificial mounted on the end effector of the PSM. This control loop takes in input the force feedback from the sensor and the desired contact force which could be intended like the maximum force that a surface can resist to (skin of fat layer for example). Even in this case an error is computed $(fd - f)$. This calculus gives in output a force-based image correction ds. The sum of ds and sd will give the current desired feature to achieve named s^*. In fig.1 is shown the feedback control law: we can see that the direct chain with the visual control law is update by the vision system loop which returns an actual position to guarantee optimal joint velocity values. In addition there is the force feedback loop that will correct the position by continuously checking the stiffness.

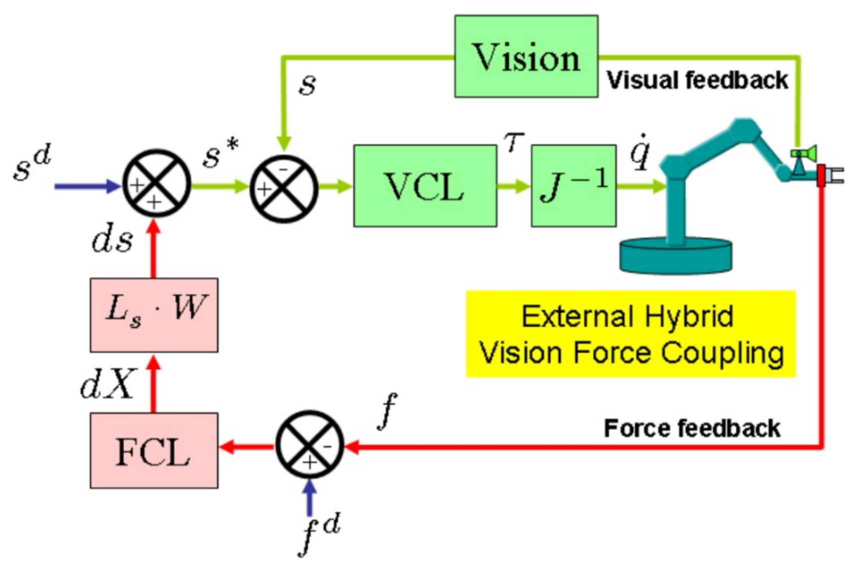

Figure 3: Control scheme of our work

We have here as inputs fd, f, S^d and S, namely the desired and the actual contact force and image features respectively; then L is an estimation of the

interaction matrix and K, W are standard error-proportional gain matrices and C is a compliance matrix; so $\dot{p}$ is the expression of end effector velocity.
We can see that the force measurement is proportional to the environmental stiffness K and to the displacement dX.
S^d is then modified according to the output of the control law in force (relative portion) projected on the sensor space by means of interaction matrix L.
Given

$$dX = K^-1(f^d - f) \tag{7}$$

the force controller only modifies the reference trajectory of visual observations S^d, so we have

$$S^* = S^d + dS \tag{8}$$

where S^* is the modified reference trajectory of image features and dS can be computed by projecting dX by means of the interaction matrix as

$$ds = LWdX \tag{9}$$

(note that ds can also be computed using the camera and object models if they are available). For more details about it, see the papers [7] and in [8].
Combining vision and force data in a closed loop feedback control system becomes extremely important, specially if the task requires the tools to come into contact with the external environment.
So coupling is done in sensor space: the reference trajectory generated by visual control is modified by the force control loop.

$$\dot{p} = \hat{L}^{\dagger} K(s^* - s) \tag{10}$$

with

$$s^* = s^d - \hat{L}WC(f^d - f) \tag{11}$$

3 Experiments

In this section we focus on the experimental part of this paper: starting from the work done by the students of the cited paper [8], we first carried out debugging operations on the code relating to the $eye-in-hand$ model; then we focused on the $eye-to-hand$ model (only sketched in [8]). Here,first we guaranteed a good fluidity of code thanks to personal re-implementation and careful tuning of the parameters, then we also implemented on board the inverse kinematics. In addition, we have also dedicated ourselves to developing additional support tasks to make our work complete and to make user understanding easy.

3.1 V-REP, Matlab and da Vinci Robot

We implemented the control scheme described in the previous section by interfacing Matlab and V-REP by means the standard API. In particular, a Matlab script implements the control unit, and a V-REP scene provided by [6] implements the physics of the daVinci robot. This robot consists of three actuated robot arms attached to a non-actuated common base. The central arm is named Endoscopic Camera Manipulator (ECM), whereas the lateral arms are identical and are named Patient Side Manipulators (PSMs). So simply by executing scripts of tasks in Matlab it is possible to perform a complete simulation with V-REP, working with its nodes and the different features taken into account [3].

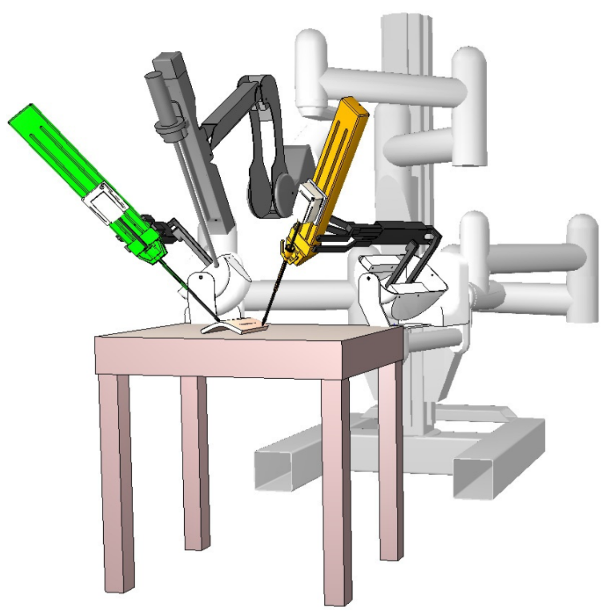

Figure 4: da Vinci robot

Our project is mainly characterized by three scripts: **utils.m** where we have implemented several useful and recurring functions, such as the method $getPose(who, wrtWho, ID, vrep)$ which returns the pose of one object with respect to another. Here, very important is the *syncronize* function which allows us to perform the physical connection between any nodes in VREP. Then we have **kinematicsRCM.m**, where the direct and the inverse kinematics are implemented; here we computes Jacobian of PSM current configuration (6x6 matrix) after the setting of each joint (the inverse kinematic will be explained into next section). In the end there is **test.m**: here we can find the main loop that execute each task we are talking about.
Finally, in order to guarantee continuity for future work on this environment, we have implemented our personal scene in the classic VREP environment with da Vinci robot, supplied by Sapienza [6].

3.2 Convergence with Inverse Kinematics

Inverse kinematics is a technique that calculates the required or optimal motion of a connected system of objects so that one arrives at a certain destination. So it can determine how a robotic arm should move so that the end effector is correctly positioned. So the inverse kinematics plays a very important role in robotics thanks to which we are able to carry out any kind of research experiment.

3.2.1 Task problem and solutions

We want to carry out the yet implemented task of *suturing* [8] in which we have to design a trajectory for PSM end effector that must visit each set of landmarks (features) starting from its home position: but this time converging through the inverse kinematics. Our function $inverseKinematics(Q, err, mode)$ take as input the current configuration Q (vector 1x6), the error (in pose) and the current mode; if mode is 1 we are in visual servoing; if mode is 0, go to home position (proportional control).

Before proceeding an important remark must be made. The output $\dot{p}$ is generated from $\dot{p} = \hat{L}^{\dagger} K(s^{*} - s)$ and it's important to understand that this is expressed w.r.t. the frame of the vision sensor. Since we cannot deal with such a reference in the inverse kinematic process we have to make a transformation first. This transformation is based on the fact that the Jacobian we extracted from [6] is expressed with the RCM has base frame. According to this it is fundamental to convert $\dot{p}_{VS}$ in $\dot{p}_{RCM}$ before going forward.

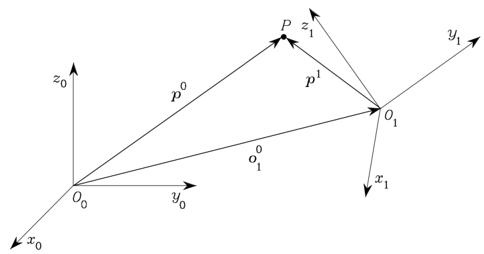

Figure 5: Representation of a point P in different coordinate frames

The foundation of such a conversion is in our 'Robotics Bible' [2].
As shown in Fig. 5, consider an arbitrary point P in space. Let p^0 be the vector of coordinates of P with respect to the reference frame O_0. Consider then another frame in space O_1. Let o_1^0 be the vector describing the origin of Frame 1 with respect to Frame 0, and R_1^0 be the rotation matrix of Frame 1 with respect to Frame 0. Let also p1 be the vector of coordinates of P with respect to Frame 1. On the basis of simple geometry, the position of point P with respect to the reference frame can be expressed as:

$$p^0 = o_1^0 + R_1^0 p^1 \tag{12}$$

that represents the coordinate transformation (translation + rotation) of a bound vector between two frames. The inverse transformation can be obtained by premultiplying both sides of (eq) by R_1^{0T}. The homogeneous representation of a generic vector p can be introduced as the vector $\hat{p}$ formed by adding a fourth unit component. So the coordinate transformation (eq) can be compactly rewritten as:

$$\hat{p}^0 = A_1^0 \hat{p}^1 \tag{13}$$

where A is termed homogeneous transformation matrix.

All of this can be applied to our case where with the method $getPoseInRCM$ defined in *util.m*, that takes as input a 6x1 vector of position and orientation of RCM w.r.t. the VS and the pose of the end effector w.r.t. the VS respectively and it returns as output the pose in RCM frame. The trick is to use the $rotm2eul$ and $eul2rot$ commands to compute the rotation and the new orientation of the frame: it allows us to extract rotation matrix associated to orientation described in euler angles of RCM w.r.t the VS.

Indeed once we solve this problem, we can compute the error with the next pose and then, through the Jacobian send the necessary information to the joints and continue with our task.

So the error is achieved through a function (see *util.m*) that computes the difference between desired and actual position, while the angular difference is used for the orientation. The $inverse_kinematics$ function will return as output the next configuration to converge to desired pose and it will describe the path. We decide to implement the Newton method starting from the following formula

[2]:

$$q^{k+1} = q^k + J_r^{-1}(q^k)[r_d - f_r(q^k)] \tag{14}$$

where the inverse Jacobian is replaced in this case with its pseudo-inverse. Very soon, however, we noticed through several tests that if we multiply a scaling factor α (just like in the Gradient method) we obtain a hybrid inverse control law that takes the advantages of two above methods. Acting like this it's been possible to give more priority to some joints and remove 'power' to others. This scaled method revealed to obtain the best performance and convergence is guaranteed if q0(initial guess) is close enough to some q* with a quadratic terminal convergence rate.

An unexpected problem has arisen at this time: although the control laws were correctly implemented (many checks were made to look for any code errors), before reaching the target, the robot made a strange movement going away from the goal and diverging. After careful research we understood that the problem was due to the way the end effector frames were initially imposed wrongly: therefore essentially VREP received distorted data which obviously led the simulator to diverge. So changing the orientation of the vision frame in according to Sapienza paper [6] we reached the desired task and solved the problem.

3.2.2 Results

The inverse kinematic task over the suturing experiment is then completed. We identify 11 steps of this process that is repeated for every landmark spot. The control law will:

1. put the PSM end effector pose at home;
2. select next landmark to converge to;
3. start mode 1;
4. detect landmark attached to the end effector;
5. compute the interaction matrix for every landmark of the EE;
6. get the image error w.r.t. vision sensor;
7. consider force correction if force > 0;
8. get $\dot{p}$ and dx w.r.t. vision sensor;
9. next desired pose w.r.t. vs is calculated as current pose + dx;
10. desired pose w.r.t. vision sensor is converted in next pose w.r.t. RCM frame;
11. next des. pose w.r.t. RCM is converted in joint values via inverse kinematics;

In order to let the reader to visualize the experiment, some screenshots of are collected below (Fig.6).

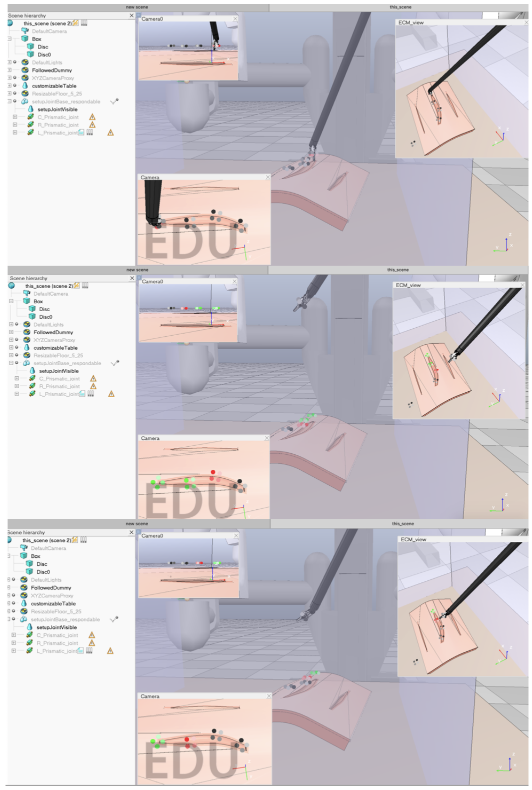

Figure 6: some screenshots of the path followed by the end-effector

Furthermore the convergence toward each landmark is shown thanks to the function plotData (Fig. 8a) that highlights how the position of landmarks on the end-effector changes (black circles) till convergence is reached, that is the overlap with the target's landmark (red circles).

3.3 Additional tasks

Tuning of parameters
To ensure a smooth functionality and fast convergence, we carefully tuned some important parameters in our model.
In order to solve the main problem of jerky movement during the descent phase of the PSM, we reduce the value of the compliance matrix C. Indeed, after several experiments we set
$C = diag([0.1, 0.1, 0.06, 0.01, 0.01, 0.01]*5)$ from previous value $C = eye(6)*(10)$; and we solved the problem of bouncing due to normal force at touchdown.
Another important tuning was the one about the gain matrix K for the mode 1 convergence. From $K = eye(6) * (10^-3)$; we adopted
$K = diag(0.9*[0.20.20.23.53.55.5]*10^-1)$ giving more importance to orientation

error than the position one. One of our goal was to guarantee a perpendicular landing on the skin simulating a real approaching problem in surgical problems. Of course this was a personal interpretation of the problem.
Then, we notice that tuning the value of the scaling factor α into our hybrid Newton-Gradient formula (4), the PSM descent is less rough and more fluid: in fact, finding a good α value allows the robot to reach its destination avoiding local minimum points and possible departures from the target.

Green and Red Features

To let the user understand better which landmarks has been pointed for convergence, we introduced a Lua function that light up in red the quartet of landmarks. Instead, once they are reached, they turn green; when all features are green the final convergence is reached (Fig.5). All this can be done in Matlab in a very particular way thanks to the function
vrep.simxCallScriptFunction: this will call a CoppeliaSim script function (lua code in V-REP) in which we set an if-else cycle that will perform our task.

Highlight the force with plotData

To better explain the behaviour of the convergence when a positive force is read we decided to add a plot with the aim to represent the norm of the force and torque inputs. In fact, every time the end effector comes into contact with the surface instead of converging, the force of this touch will be shown in red with a discrete signal over time (fig. 8b). As we said, the force will help the end effector to find the right way to get closer to the features and converge in the desired target. When this happens, again thanks to this plot function, it will be shown through a green signal.

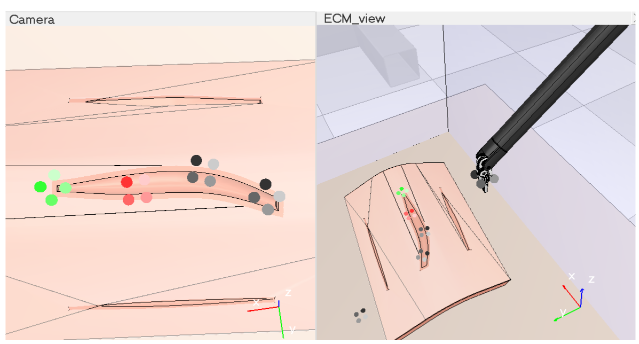

Figure 7: Example of coloring features: the first set of landmarks is reached (green) while the second not yet (red).

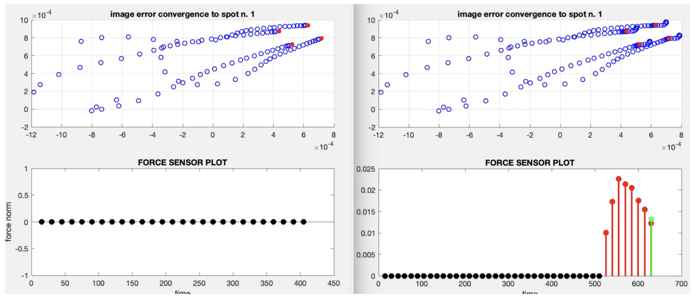

Figure 8: main plot over time: at start and when the first spot is reached. Above (a), plotData value is shown, while down (b), the plot of the force.

4 Future development and conclusion

In future we could think to extend our work to the other ”agents” of da Vinci like in real cases, where the robot is used by making all the arms interact with each other; so it could be very interesting being able to move togheter the two PSMs and the ECM too. For example the ECM could perform the task to follow the movement of the PSM dynamically, performing a better task of visual servoing, changing not only the orientation around the axes but also performing a translation on z to get more closely to it (e.g. complex internal surgical operations).

As we know, surgery is making great strides thanks to technological development and the introduction of cutting-edge techniques in this field, using robots or innovative tools that can help solve problems like never before. Contributing to the development of techniques and tasks in this area, with an engineering approach is very fascinating. We have seen in this project how stimulating it is to use virtual platforms where, through simulators, important tasks can be developed which one day can be taken into account in real life too.

A Algorithm flow in details

In this section it's described the complete process of the algorithm from its startup to the end.
Once the vrep scene has been open and launched from the start button, it's time to start the matlab script.
The first process is the connection to the VREP scene done by the init-connection() module which builds a client-server connection with the scene thanks to the vrep.simxStart function. With a verified connection it's time to retrieve the handles of the objects we will be used inside the matlab script. The handle can be considered as a pointer to the object inside vrep from which it's possible to read data like velocity of the joint, joint value, pose in the space ecc. The synchronization phase has the role to wait until non-zero values are kept from the scene because vrep and matlab need few seconds until non-zero values are read. Without the synchronization phase a joint value for example will be 0 for few seconds before to read valid values

References

[1] Ruiz M. Bellaccini M., Gagliardi M. *ROS/Gazebo simulations on eye to hand*. Sapienza, 2012.

[2] G. Oriolo L. Villani B.Siciliano, L. Sciavicco. *Robotics: Modelling,Planning and Control*. Springer McGraw-Hill, 2008.

[3] CoppeliaSim. *CoppeliaSim User Manual*. version 4.0.0, 2019.

[4] Seth Hutchinson; D. Hager; Peter Corke;. *A tutorial on Visual Servo Control*. IEEE member, 1996.

[5] S. Hutchinson François Chaumette. *Visual Servo Control Pt1 and Pt2*. IEEE member, 2006.

[6] M. Vendittelli F. Ficuciello M. Ferro B. Siciliano G.A. Fontanelli, M. Selvaggio. *A V-REP Simulator for the da Vinci Research Kit Robotic Platform*. BioRob, Avaliable online at: https://github.com/unina-icaros/dvrk-vrep, 2018.

[7] Youcef Mezouar;Mario Prats;Philippe Martinet. *External Hybrid Vision/Force Control*. The 13th International Conference on Advanced Robotics August 21-24, 2007, Jeju, Korea, 2007.

[8] Cappella L; Cirillo M; Rinaldi R;. *Hybrid force-vision control for da Vinci simulator*. Sapienza, 2019.