Control and Trajectory Design for Bone Surgeries

André Neto (Student ID: 2019237495) and João Ferreira (Student ID: 2019237648)

Medical Robotics
Department of Electrical & Computer Engineering
University of Coimbra

08 Oct, 2026

I Introduction

The main goal of this project is to implement control and trajectory architectures for bone surgeries, taking into account geometric, kinematic and dynamic models of planar robots.

1 Contextualization

The planar surgical robot represented in Fig. 1 interacts with a virtual bone for surgery. The surgeon grabs directly the tip for operation (comanipulation mode), which should be done along a specific Cartesian direction.

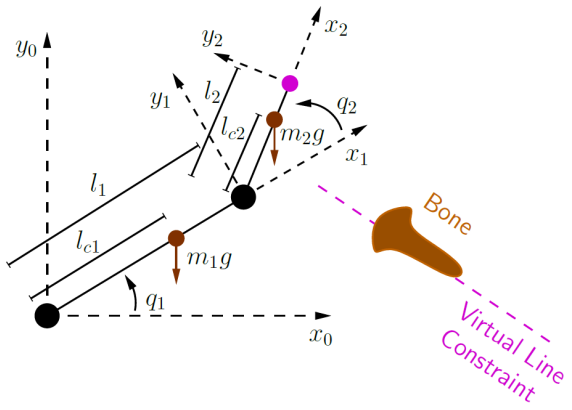

Figure 1. Two-link planar robot for bone surgeries

II Control Design

Looking to the figure 1 we have the Robot dynamics given by:

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = \tau \tag{1}$$

where $M(q)$ is the inertia matrix, $C(q,\dot{q})$ represents centripetal and Coriolis terms and $g(q)$ is the gravity term. τ is the generalized torque applied to the joint vector q, for example,

$$\tau = \tau_c + \tau_f + \tau_e \tag{2}$$

where τ_c is the control torque, τ_f is the torque due to friction and τ_e represents all other external torques. Therefore, (1) can be written as

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) + F_v\dot{q} = \tau_c + \tau_e \tag{3}$$

where F_v is viscous friction (we are neglecting other friction sources). For this surgical robot consider that $m_1 = 0.4$, $m_2 = 0.2$, $l_1 = 0.3$, $l_2 = 0.15$, $l_{c1} = l1/2$, $l_{c2} = l2/2$ and $F_v = 0.05$. The moment of inertia of link i, I_i , about the center of mass is given by $I_i = \frac{1}{12}m_i l_i^2$ along z (all quantities are in SI units).

1 Forward Kinematics

In robot kinematics, forward kinematics refers to the use of the kinematic equations of a robot to compute the position of the end-effector from specified values for the joint parameters.

The kinematics equations for the series chain of a robot are obtained using a rigid transformation $[Z]$ to characterize the relative movement allowed at each joint and separate rigid transformation $[X]$ to define the dimensions of each link. The result is a sequence of rigid transformations alternating joint and link transformations from the base of the chain to its end link, which is equated to the specified position for the end link,

$$[T] = [Z_1][X_1][Z_2][X_2]\dots[Z_n][X_n], \tag{4}$$

where [T] is the transformation locating the end-link.

The *Denavit-Hartenberg* convention (DH) is used to define the joint matrices $[Z]$ and link matrices $[X]$ to standardize the coordinate frame for spatial linkages. This convention positions the joint frame so that it consists of a screw displacement along the Z-axis, eq. (5a), and it positions the link frame so that it consists of a screw displacement along side the X-axis, eq. (5b)

$$[Z_i] = \mathrm{Trans}_{Z_i}(d_i)\,\mathrm{Rot}_{Z_i}(\theta_i) \tag{5a}$$
$$[X_i] = \mathrm{Trans}_{X_i}(a_{i,i+1})\,\mathrm{Rot}_{X_i}(\alpha_{i,i+1}) \tag{5b}$$

Each transformation link goes along a serial chin robot, and can be described by the coordinate transformation,

$$\begin{aligned} {}^{i-1}T_i &= [Z_i][X_i] \\ &= \mathrm{Trans}_{Z_i}(d_i)\,\mathrm{Rot}_{Z_i}(\theta_i) \\ &\quad \mathrm{Trans}_{X_i}(a_{i,i+1})\,\mathrm{Rot}_{X_i}(\alpha_{i,i+1}), \end{aligned} \tag{6}$$

where the matrices associated with these operations are:

$$\mathrm{Trans}_{Z_i}(d_i) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7a}$$

$$\mathrm{Rot}_{Z_i}(\theta_i) = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & 0 \\ \sin\theta_i & \cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7b}$$

$$\mathrm{Trans}_{X_i}(a_{i,i+1}) = \begin{bmatrix} 1 & 0 & 0 & a_{i,i+1} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7c}$$

$$\mathrm{Rot}_{X_i}(\alpha_{i,i+1}) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha_{i,i+1} & -\sin\alpha_{i,i+1} & 0 \\ 0 & \sin\alpha_{i,i+1} & \cos\alpha_{i,i+1} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7d}$$

The DH parameters necessary for this convention are:

- l, which represents the offset along the previous z axis, from the old x axis to the new x axis;
- q, which represents the angle about the previous z axis, from the old x axis to the new x axis;
- r, which represents the length of the common normal;
- α, which represents the angle about the common normal, from the off z axis to the new z axis.

Using these parameters, we can define Z_i and X_i as shown in eqs. (5a), (5b) and eqs. (7a) throughout (7d) which gives a relation between the link $n-1$ and n as shown in eq. (8), where R and t represent the rotation matrix and translation vector, respectively.

$$
\begin{aligned}
{}^{i-1}T_i &= \left[\begin{array}{ccc|c} c(q_i) & -s(q_i)c(\alpha_i) & s(q_i)s(\alpha_n) & r_i c(q_i) \\ s(q_i) & c(q_i)c(\alpha_i) & -c(q_i)s(\alpha_i) & r_i s(q_i) \\ 0 & s(\alpha_i) & c(\alpha_i) & l_i \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \\
&= \left[\begin{array}{c|c} R & t \\ \hline 0 & 1 \end{array}\right]
\end{aligned} \tag{8}
$$

For these first point, we developed the function "forward-kinematics.m", that computes the forward kinematics taking as input, the values of q and l.

2 2 - Jacobian Matrix

$$
\mathbf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \dots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \dots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} \tag{9}
$$

The Jacobian matrix is one of the most important quantities in the analysis and control of robot motion. It encodes relationships between velocities and can be thought of as the vector version of the ordinary derivative of a scalar function.

Consider a n-link manipulator with joint variables $\mathbf{q} = [q_1 \dots q_n]$ and let $T_n^0(\mathbf{q})$, expressed in eq. 10, denote the transformation from the end-effector frame to the base frame. As the robot moves, the joint variables and the end-effector position and orientation will be functions of time.

$$
T_n^0(\mathbf{q}) = \begin{bmatrix} R_n^0(\mathbf{q}) & t_n^0(\mathbf{q}) \\ 0 & 1 \end{bmatrix} \tag{10}
$$

The Jacobian matrix seeks to express the relation between the (linear and angular) velocities and the joint variables, as expressed in equations 11a and 11b.

$$
v_n^0 = J_v \dot{\mathbf{q}} \tag{11a}
$$

$$
\omega_n^0 = J_\omega \dot{\mathbf{q}} \tag{11b}
$$

Combining both equations, we obtain the Jacobian matrix, J_n^0, express in eq. **??**.

$$
\begin{bmatrix} v_n^0 \\ \omega_n^0 \end{bmatrix} = \left[\begin{array}{c} J_v \\ \hline J_\omega \end{array}\right] \dot{\mathbf{q}} = J_n^0 \dot{\mathbf{q}} \tag{12}
$$

The determination of the Jacobian o anu manipulator is simple once the forward kinematics are worked out, since all the quantities needed are available through that. The i-th column of the Jacobian matrix is given by eq. **??** and **??** if the joint i is revolute or prismatic, respectively. The only calculations that are required are of the unit vectors z_i and the coordinates of the origins $t_1, \dots, t_n$, which are given by the first three elements in the third column of T_i^0 and the first three elements in the fourth column of T_i^0.

$$
J_i = \begin{bmatrix} z_{i-1} \times (t_n - t_{i-1}) \\ z_{i-1} \end{bmatrix} \tag{13a}
$$

$$
J_i = \begin{bmatrix} z_{i-1} \\ 0 \end{bmatrix} \tag{13b}
$$

Finally, we did a function for matlab to calculate the different jacobians, the jacobian referred to the end-effector, and the jacobians referred to each center of mass for both linear and angular motions. This function is called "jacobian.m".

3 3 - Compute $M(q), C(q,\dot{q}), g(q)$

To compute these functions we created a matlab function named "Symbolic.m", where we computed all these matrices but symbolic, so we can use it later in the representation of the robot arm.

4 3.1 - Gravity term

Starting with the gravity term, this term was computed by equations given by the professor as we can see next:

$$
u(robo) = \sum_{i=1}^{n} -m_i r_i^T g \tag{14}
$$

$$
Gravityterm : g(q) = \frac{\partial u}{\partial q} \tag{15}
$$

To compute this we used:

$$
r_i^T g = |r_i||g|cos(r_5, g)| = -gh_i \tag{16}
$$

and for the values of h:

$$
h_1 = lc_1 \times sin(q_1) \tag{17}
$$

$$
h_2 = l_1 \times sin(q_1) + lc_2 \times sin(q_1 + q_2) \tag{18}
$$

Calculating this with symbolic values we got the next expression as can be seen in the matlab function "termo-gravidade":

$$
tg = \begin{bmatrix} g \times m_2(lc_2 \times cos(q_1 + q_2) + l_1 \times cos(q_1) + g \times lc_1 \times m_1 \times cos(q_1) \\ g \times lc_2 \times m_2 \times cos(q_1 + q_2) \end{bmatrix} \tag{19}
$$

5 3.2 - Inertia matrix

Second term is the Inertia matrix. This matrix was given by the professor, we had a matlab function "Mass Computation2LinkRobot.m" that computed the symbolic Inertia matrix with the next code:

```
% Mass Matrix computation
%of a 2-link planar arm

clear all
syms m1 m2 q dq q1 dq1 q2 dq2  real
syms l1 l2 lc1 lc2 I1c I1cxx I1cyy I1czz I2c
I2cxx I2cyy I2czz  real

% Inertia tensors referred
%to the center of mass
I1c = [I1cxx 0 0;0 I1cyy 0; 0 0 I1czz];
I2c = [I2cxx 0 0;0 I2cyy 0; 0 0 I2czz];


% symbolic variables

q=[q1;q2];
dq=[dq1;dq2];


% Jacobians referred to the center of mass

Jvc1 = [-lc1*sin(q1) 0; lc1*cos(q1) 0;0 0];

Jvc2 = [-l1*sin(q1)-lc2*sin(q1+q2),
-lc2*sin(q1+q2);
l1*cos(q1)+lc2*cos(q1+q2),
lc2*cos(q1+q2); 0 0];

Jw1=[0 0;0 0;1 0];
Jw2=[0 0;0 0;1 1];

 % rotation matrices associated to
 %each coordinate frame (rotation along z)
R1=[cos(q1) -sin(q1) 0; sin(q1) cos(q1) 0;
0 0 1];
R2=[cos(q1+q2) -sin(q1+q2) 0;
sin(q1+q2) cos(q1+q2) 0; 0 0 1];

MassSymbolic = simplify(m1*Jvc1'*Jvc1
+Jw1'*R1*I1c*R1'*Jw1
+m2*Jvc2'*Jvc2  + Jw2'*R2*I2c*R2'*Jw2)
```

The final result for this matrix is given by:

$$M_q = \begin{bmatrix} m_2 \times l_1^2 + 2 \times m_2 \times cos(q_2) \times l_1 \times lc_2 + m_1 \times lc_1^2 + I1czz + I2czz & m_2 \times lc_2^2 + l_1 \times m_2 \times cos(q_2) \times lc_2 + I2czz \\ m_2 \times lc_2^2 + l_1 \times m_2 \times cos(q_2) \times lc_2 + I2czz & m_2 \times lc_2^2 + I2czz \end{bmatrix} \quad (20)$$

6 3.3 - Coriolis Matrix

The last term is the Coriolis Matrix, that can be calculated by the next 2 equation:

$$C(q,\dot{q}) = \dot{M}(q) - 0.5 \times \dot{q}^T \times \frac{\partial M}{\partial q} \quad (21)$$

$$\dot{M}(q) = \frac{\partial M}{\partial q_1} \times \dot{q}_1 + \frac{\partial M}{\partial q_2} \times \dot{q}_2 \quad (22)$$

So, our final expression for the Coriolis Matrix is:

$$C(q,\dot{q}) = \frac{\partial M}{\partial q_1} \times \dot{q}_1 + \frac{\partial M}{\partial q_2} \times \dot{q}_2 - 0.5 \times \dot{q}^T \times \frac{\partial M}{\partial q} \quad (23)$$

Any of these three terms where calculated using symbolic terms and have their own functions to be used in the work, "termo-gravidade.m", "Inertia.m" and "coriolis.m".

7 4 - First tests with the robot

After having all the terms needed to control the robot, we are now starting to do some tests to see how it reacts. In this section we are going to use the next initial conditions:

$$q = \begin{bmatrix} \frac{\pi}{2} & \frac{\pi}{4} \end{bmatrix}^T \quad (24)$$

$$\dot{q} = \begin{bmatrix} 0 & 0 \end{bmatrix}^T \quad (25)$$

For this test part, we need to use simulink blocks, here we are going to calculate all the accelerations using joints' angles and velocities. For that we created a matlab function "compute-q" that makes these calculations and is implemented in the block "Interpreted Matlab Fcn" in the next simulink file.

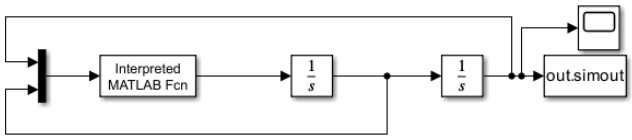

Figure 2. Robot simulator using simulink

In the function "compute-q" we use the inputs (q,dq) to compute the gravity term, the Inertia matrix and the Coriolis matrix, and then we proceed to the calculations for the accelerations:

$$\ddot{q} = M^{-1}(q) \times [\tau - g(q) - C(q,\dot{q})\dot{q}] \quad (26)$$

where τ is only equal to the viscous friction term, because in these tests $\tau_c = 0$ and $\tau_e = 0$.

Now we have all the system built to run some tests. As mentioned in the statement we used different values for the viscous friction term to see how it responds to it and to show this behaviour we created a function "showRobot.m" that is called several times to show the movement of the robot with the conditions given earlier.

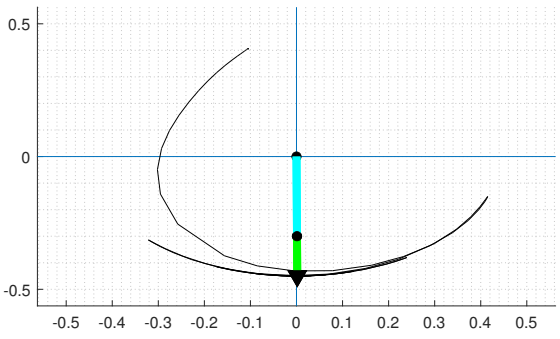

Figure 3. Robot simulator using simulink with Fv =0.05

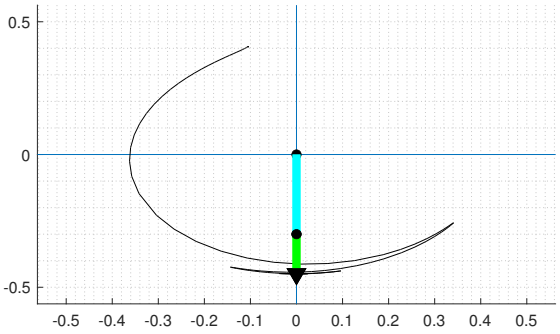

Figure 4. Robot simulator using simulink with Fv =0.1

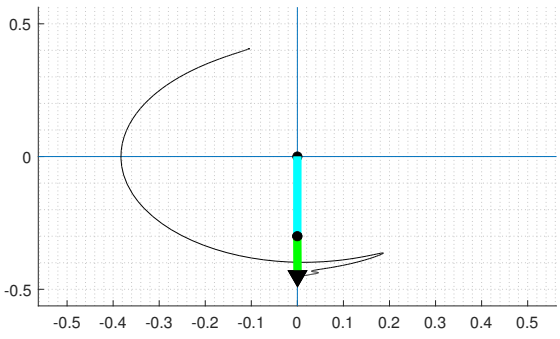

Figure 5. Robot simulator using simulink with Fv =0.2

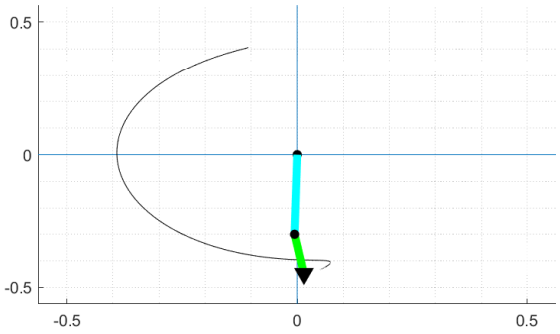

Figure 6. Robot simulator using simulink with Fv =0.5

Analysing the graphics we can see that the viscous friction term has some influence in the robot's behaviour. The higher this term is the slower is the move and it does not bounce to much in the bottom. So, it will need greater forces to vary its position, if we put an higher viscous friction term it won't move unless we provide higher forces. It can be seen in the images, but it can also be better visualized in the videos sent with the submission.

8 5 - Tests with $\tau = g(q)$

Now we are going to use gravity compensation and for that we define the torque, τ, equal to the gravity term, $g(q)$ (this is the only change in the system comparing to the last one).

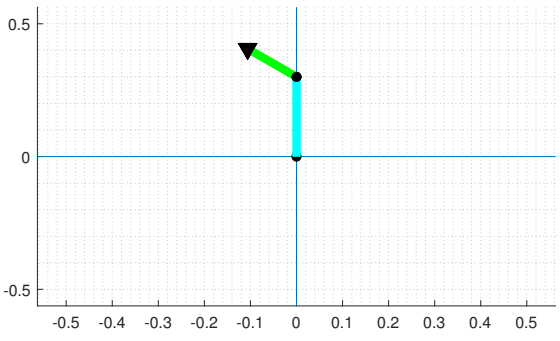

Figure 7. Robot simulator using simulink with $\tau = g(q)$

As expected the robot does not move as it can be seen in the figure(7) and in the vídeo, because the gravity is compensated and q1 and q2 don't vary.

9 6 - Tests with different Cartesian forces

Here we used different Cartesian forces applied to the robot's end-effector and continued to use the gravity compensation. The forces are $F_c = [0, -10]^T$, $F_c = [10, 0]^T$ and $F_c = [10, -10]^T$.

To do these tests we just changed the torque expression as expressed in the notes given by professor in the theoretical classes.

$$\tau = g(q) + J^T F_c \tag{27}$$

This equation uses the Jacobian of the end-effector, J, calculated before, the cartesian forces applied and the gravity term, as it continues to be applied. Nothing else changes in the system, unless this equation

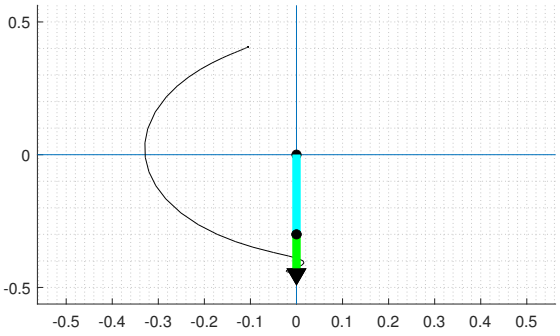

Figure 8. Robot simulator using simulink with $F_c = [0, -10]^T$

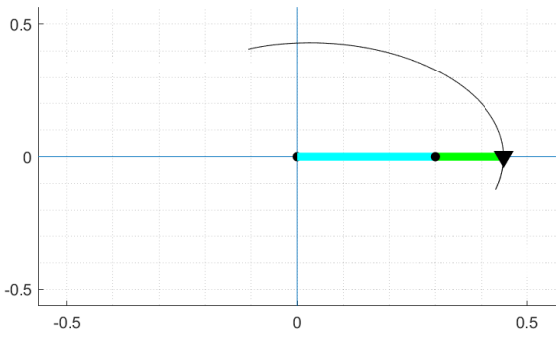

Figure 9. Robot simulator using simulink with $F_c = [10, 0]^T$

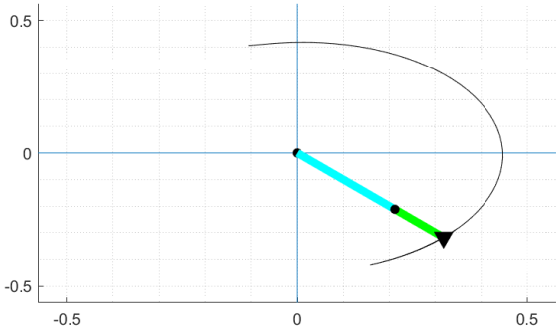

Figure 10. Robot simulator using simulink with $F_c = [10, -10]^T$ and $F_v = 0.5$

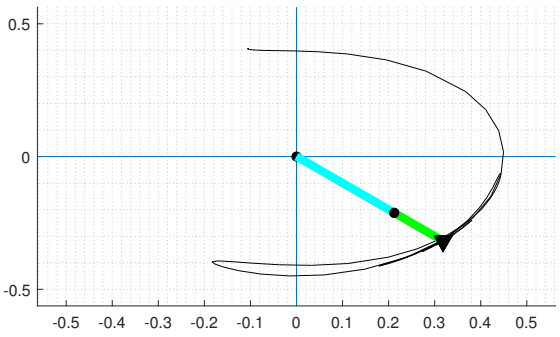

Figure 11. Robot simulator using simulink with $F_c = [10, -10]^T$ and $F_v = 0.1$

These figures(8)(9)(10) are the results when using each one of the cartesian forces and the viscous friction term equal to 0.5, because was the last used. We can see that the end-effector goes to the position predicted, because of the use of cartesian forces. It is like the end effector is being pulled towards those directions. Another conclusion that we can get is that using a higher viscous term the behaviour is more smooth than using a lower viscous term, as we can compare in the figures(10), (11) using a $F_v = 0.5$ and $F_v = 0.1$, respectively.

10 7 - Tests for position control schemes in the task space

There are two different methods to apply position control mentioned in the slides. Here we used the more basic approach, where a PID control is applied to the tracking error of the robot, which is then converted to the joint space through the Jacobian matrix. We implemented these approach using the next simulink diagram.

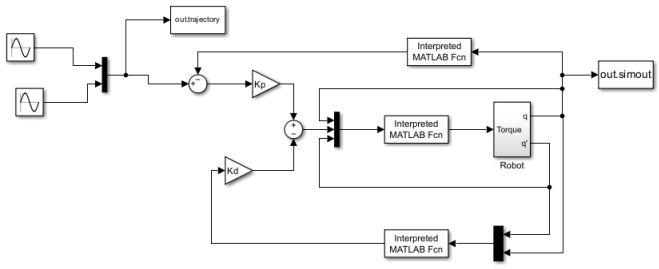

Figure 12. Simulink for PD control

It can be seen in the figure (12), a PD control that is used to provide the position control. There we use functions like "X-7", "Xderivative-7" and "Dynamic-model-7" to make this control possible. This last function is where we implemented the basic approach for this control, and we only used P and PD control because is what is required for this part.

To test these control is said in the statement to generate an elliptical trajectory centered around (0.25, 0.2) with major axis $m_x = 0.12m$ and minor axis $m_y = 0.07m$ and to start with a cyclic frequency of 1/4 [Hz] with Simulink´s block "Sine Wave" on time based type.

To finally see the results we used PD control, but also, only a P control to see the differences, and tried to get some good values to his control.

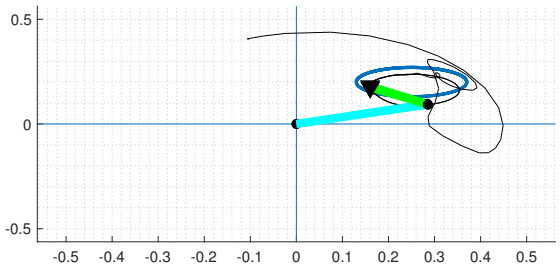

Figure 13. Robot simulator using simulink Kp=100

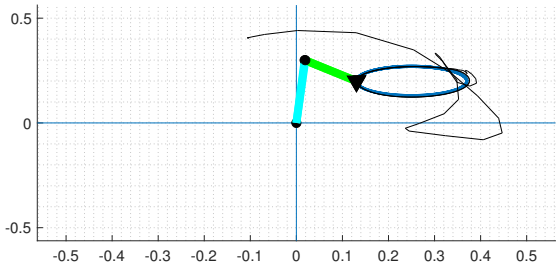

Figure 14. Robot simulator using simulink Kd=400

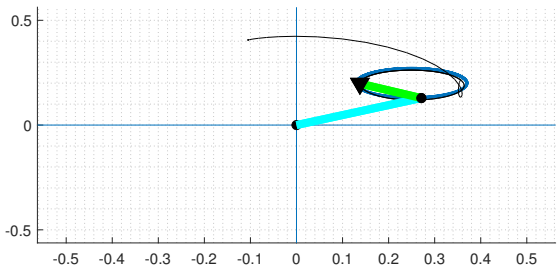

Figure 15. Robot simulator using simulink Kp=400 and Kd=10

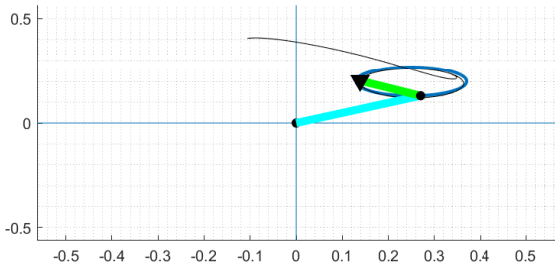

Figure 16. Robot simulator using simulink Kp=400 and Kd=30

Analysing the figures (13)(14) we can see that the position control is better using Kp=400, because the trajectory passes above the ellipsis while with Kp=100 the trajectory is a little outside the ellipsis. The steady-state tracking error is a little higher, but it is equal for the two of them, so concluding, with a Kp=400 we have a better performance.

Looking now for the figures (15)(16) we cannot see to many differences, the trajectory is full-filled but with Kd=30 the steady-state tracking error is small than using Kp=10, so the figure (16) has a better performance.

Comparing now the P control with the PD control, we can say that the P control has worse response than the PD control, and the bigger difference is the minor steady-state tracking error.

It is also asked to make some tests with a vertical trajectory, for that we changed the parameter m_x to 0 so that we can get a vertical line, and m_y to 0.1 so we can get some thickness.

To test this control we used different values for the PD control (this time we used only PD control).

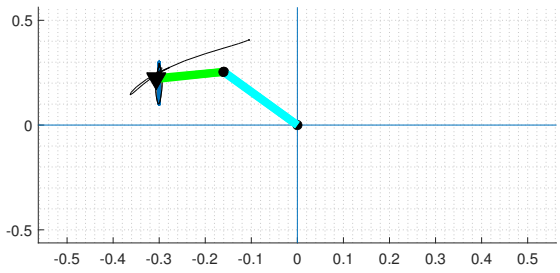

Figure 17. Robot simulator using simulink Kp=500 and Kd=10 in a vertical line

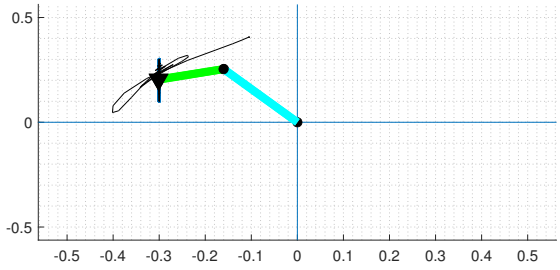

Figure 18. Robot simulator using simulink Kp=2800 and Kd=10 in a vertical line

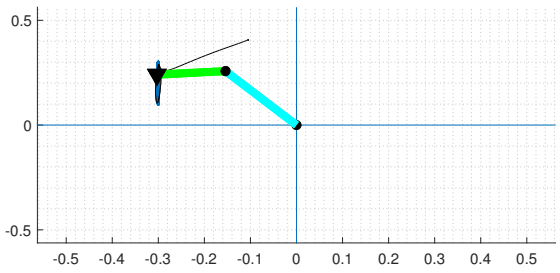

Figure 19. Robot simulator using simulink Kp=500 and Kd=90 in a vertical line

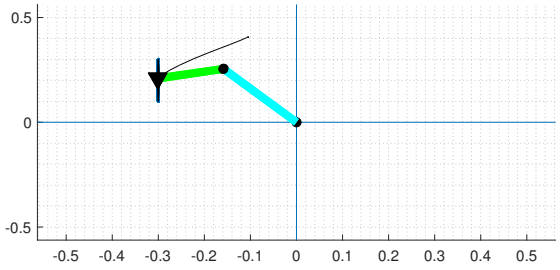

Figure 20. Robot simulator using simulink Kp=2800 and Kd=90 in a vertical line

Looking to the figures (17) (18) we can conclude that for the same Kd, a higher Kp makes the robot have a better trajectory over the vertical line but increases a little the steady-state tracking error. When we only change the parameter Kd the steady-state tracking error changes too and to decrease this error we use higher values for Kd like 90.

Finally we can conclude that the ideal parameters are Kp=2800 and Kd=90, where the performance is very good, the trajectory is over the vertical line, and there is not the steady-state tracking error. We also tried with higher values of Kp and Kd, but when increasing Kp higher than 2800, we start having some overshoot, and when we increase Kd for values over 90, the system turns to be more expensive computationally speaking, and the movement becomes slower too.

11 8 - Tests for Compliance Control

To implement the compliance control we first started by the data needed to the implementation and then we worked on the simulink so we could simulate the compliance control. First, $x_e = [0.25, 0.2]$ (is the position of the virtual bone), $k_e = \begin{bmatrix} 2000 & 0 \\ 0 & 2000 \end{bmatrix}$, (is the stiffness of the bone), $k_p = 2800$,$K_d = 90$, and $Kp = \begin{bmatrix} 10000 & 0 \\ 0 & 10000 \end{bmatrix}$ (the values for the Pd control).

Now passing to the simulink, we did as the simulink used for the previous exercise figure(12), but we did some adaptations to the Function inside the simulink so we could use the external force applied to the robot that simulates an interaction from the surgeon, and the force applied by the virtual bone. The external force is applied with $F_c = [80; 80]$ at a specific interval of time (between 4.5s and 5.5s). To see the effect of this forces applied to the robot we created a trajectory that passes through the bone (that is represented as a black circle) and the supposed it's that the robot does not pass past that point, that is the center of the black circle.

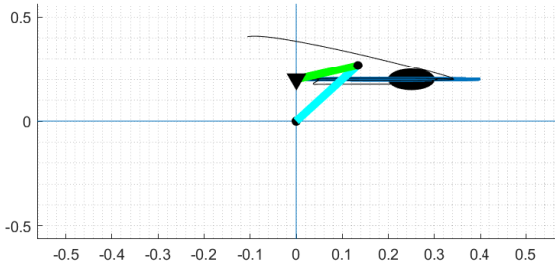

Figure 21. Robot simulator using simulink for Compliance control with the specific Ke

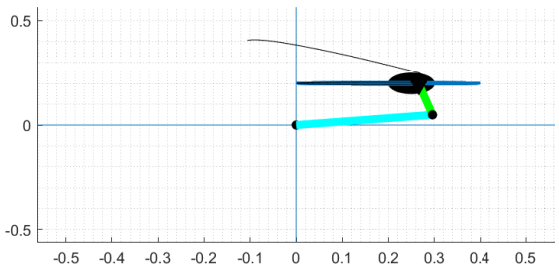

Figure 22. Robot simulator using simulink for Compliance control with a higher Ke

As results we have the figures (21) (22), where in the first we used the supposed $K_e = 2000$ and in the second we used a higher value, $K_e = 20000000$ to see if it affected the robot. Analyzing the first one, the compliance control starts in the center of the black circle (virtual bone) and the robot decreases it's velocity, not completing it's trajectory, it could go a little more but the force doesn't let him. If the objective is to the robot to stop in the center of the virtual bone the K_e needs to be higher, for that we used $K_e = 20000000$ to see it's reaction. It can be seen in the figure and video that the robot when touches the center of the virtual bone, it decreases a lot it's velocity and stops after, not getting to the desired point, just like it's supposed.

Talking now about the external force, the cartesian force, it can be seen, if we maximize that region, that we have a higher steady-state tracking error, that the robot does not fix because the control is only PD.

12 9 - Tests for Impedance Control

To implement the Impedance Control we started by the data needed and then we built the simulink. About the data we used the next values: $\dot{X}_D = [0; 0]$ for the velocity and $\ddot{X}_D = [0; 0]$ for the acceleration, a proportional gain $Pg = 1500$, a derivative gain $Dg = 40$, and the data for the impedance control: $K_d = \begin{bmatrix} Pg & 0 \\ 0 & Pg \end{bmatrix}$, $D_d = \begin{bmatrix} Dg & 0 \\ 0 & Dg \end{bmatrix}$, $M_d = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, $M_a = M_d/2$, $K_a = 0$ and $D_a = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$.

Next we built the next simulink's block diagram with the matlab function "Dynamic-model-9" inside the block "Interpreted MATLAB Fcn".

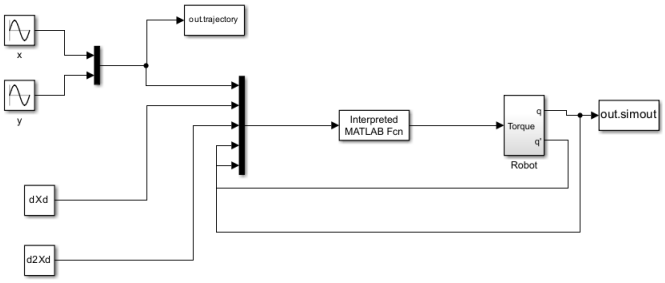

Figure 23. Simulink for Impedance control

To observe if the behaviour of the robot is correct we applied a Cartesian force to the end-effector, equal to the one used before, in the compliance control, so that we can compare later, $F_c = [80; 80]$, between 4.5s and 5.5s, therefore 1 second of external force applied.

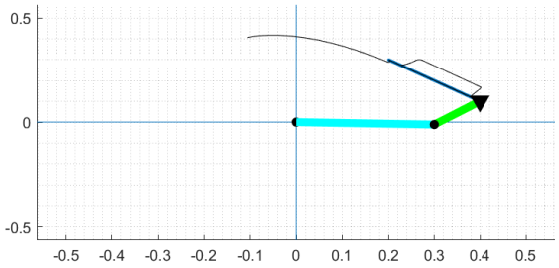

Figure 24. Robot simulator using simulink for Impedance Control

Analysing the figure (24) we can see that the system has a good performance and it behaves as it is supposed. We tested for several different gains to see the best performance and we believe those values are good for the test, to decrease the overshoot and the steady-state tracking error.

We can now compare the Compliance control with the Impedance control. Looking to the figures 21,24, we can say that the compliance control could control better the external force, because the trajectory was closer to the one desired. These conclusion can only be make because it was used the same external force for both cases.

III Trajectory Design

This section has as objective explore null-space motion with a planar three-link arm surgical robot, with a vital organ represented by a red circle and a green line as the trajectory. It is also important to maximize robot manipulability to avoid the vital organ, so that there is no contact. Here we are going to analyze the system with null-space and without it.

Our main file is the matlab file "main.m", where we initialize all the data and test the system. About the data, we used a Kp equal to 100 (to have a good performance doing the pretended trajectory), an initial position equal to $[\pi/5; \pi/4; pi/3]$, so it is similar to the robot in the image (25), a vertical line in $x = 1.5$ and a vital area centred at (-0.5,-0.75) with 0.3 as the value for radius. We also made some changes in our function "showRobot" because here we are working with a three-link arm and not a two-link arm.

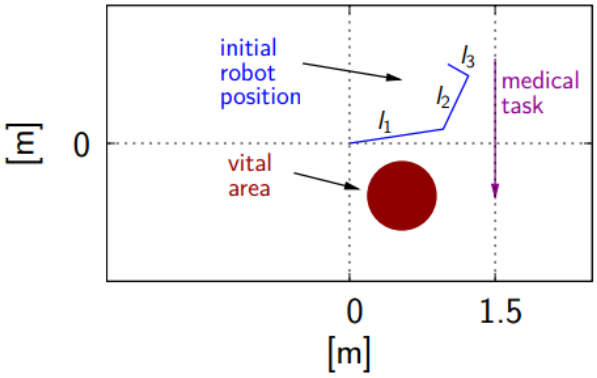

Figure 25. Scheme for the Trajectory Design

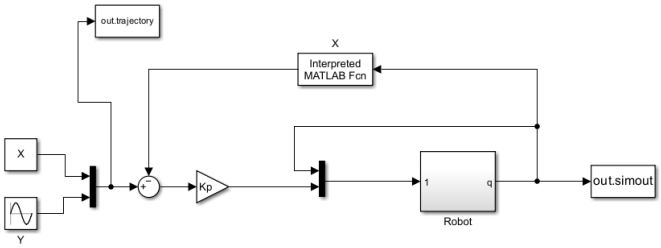

Figure 26. Simulink for Trajectory Design

In the figure (26) we have the simulink used for this part where we used the function "forward-kinematics".

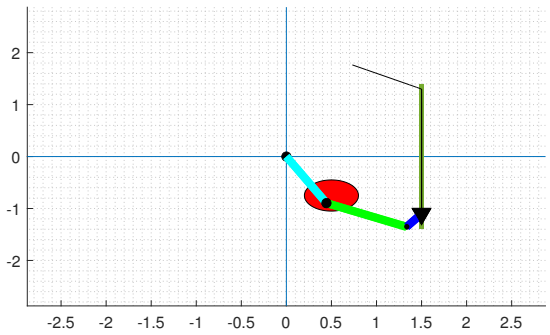

Figure 27. Robot simulator using simulink without null-space

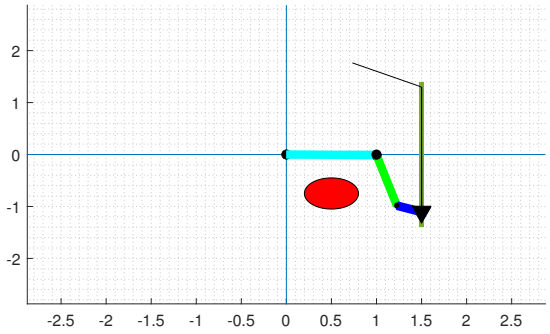

Figure 28. Robot simulator using simulink with null-space

In the figures (27) (28) we have the system without the null-space and with the null-space, respectively and for that we just changed the value K_q between 0 and 10.

It can be seen in the figures that the robot behaves as it is supposed, in the figure (27) it does the trajectory without caring about the vital area, but in the figure (28) it does care about the vital area and changes it posture to make the trajectory correctly without touching the vital area. We can also conclude that in the figure (27) the robot doesn't change it's position much because it does not need to avoid anything, otherwise in the figure (28) the robot changes more it's position comparing to the initial position because it needs to avoid the vital area, so the angles of the joints, change more than without the null-space. These behaviour can be better viewed in the videos.

IV Free Research

In this part we had some freedom to put some creativity to work, for that we tried to search for some specific surgery to

implement. When we were doing some research, come to our minds that we could do an app to test different "surgeries". For that we used the "app designer" from matlab to create the app.

This app asks for some values relative to the robot's dimension, pose, the gains, the organ's place, dimension and the trajectory needed to be done by the robot. To test this part, it's only needed to start the main file, write the numbers for the test (the values for the pose needs to be in radians, like pi/2) and finally click the start button to get the surgery test.



Figure 29. App for the tests

The figure 29 is the app that we developed, where we need to write the values we want to use are (l1,l2,l3-dimensions of the robot; q1,q2,q3-pose of the robot; center-x,center-y,radius-values for the vital area; Kp,Kq-proportional gain and the force given to the vital area to keep the robot away; x0,y0,mx,my-values for the trajectory.

The simulink is the same as before, as in the exercise about trajectory design. Using this app we can change a lot of variants in these tests, we can change values about the robot, about the trajectory and even about the vital area that we want to use. This app was developed as a way to test different situations in surgeries, it can test if the robot behaves as it is supposed and also if the pose given to the robot is good enough so it can do well the trajectory.

We did some tests to see if the app is running good, but there is a lot of other tests that can be done.

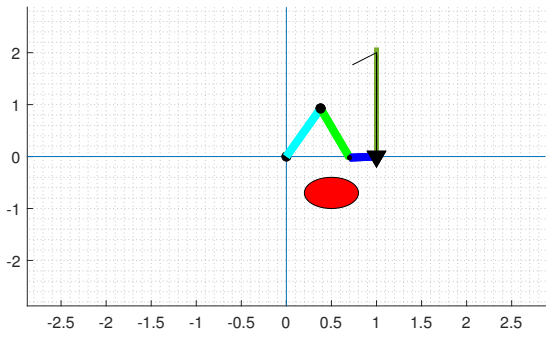

Figure 30. App for the tests

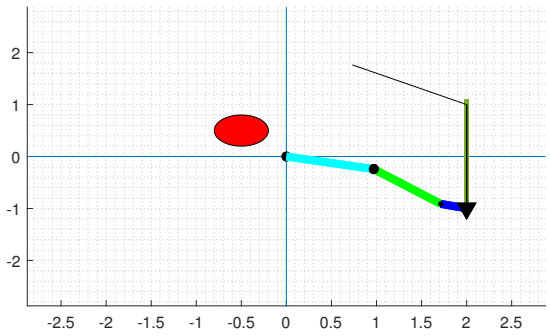

Figure 31. App for the tests

We can see in the examples, well done trajectories and in the Figure 30 the robot has to change it's pose to not touch the vital are. For the tests we used the next values:

Figure 30

- l1=1;l2=1;l3=0.3
- q1=pi/5;q2=pi/4;q3=pi/3
- center-x=0.5;center-y=-0.7;radius=0.3
- kp=100;kq=10;
- mx=0;my=-1.5;x0=1;y0=1

Figure 31

- l1=1;l2=1;l3=0.3
- q1=pi/5;q2=pi/4;q3=pi/3

- center-x=-0.5;center-y=0.5;radius=0.3
- kp=100;kq=10;
- mx=0;my=1;x0=2;y0=0

Concluding, we think that this app can help to predict if a surgery would succeed or not by testing here before, with the values from the robot, the vital area and the trajectory.

V Conclusion

In this work we worked with the main ideas from the discipline "Medical Robotics" in a Matlab ambient. We have done since tests with simple gravity and different values for viscous friction, gravity compensation, different cartesian forces, tests for position control, for compliance control and impedance control. Then we designed a trajectory simulating an operation where we have a vital organ that cannot be touched and finally a free research. All of these with different simulink schemes so that we could simulate every situation, and for that we created many different functions.

Finally, we think this project helped us to understand the theoretical part while implementing all of this project. And about the free research we think it is a good exercise, because it develops our creativity in these medical ambients after we understood the basic about medical robotics.