ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Ρομποτική ΙΙ: Ευφυή Ρομποτικά Συστήματα

Εξαμηνιαία Εργασία 1:

Κινηματικός έλεγχος ρομποτικού χειριστή με πλεονάζοντες βαθμούς ελευθερίας: Παρακολούθηση τροχιάς και αποφυγή εμποδίου

Αναφορά

Άγγελος Λουκάς 03119877
Νικόλαος Καραφύλλης 03119890
Ομάδα 50

Μάιος 2023

THEORETICAL ANALYSIS

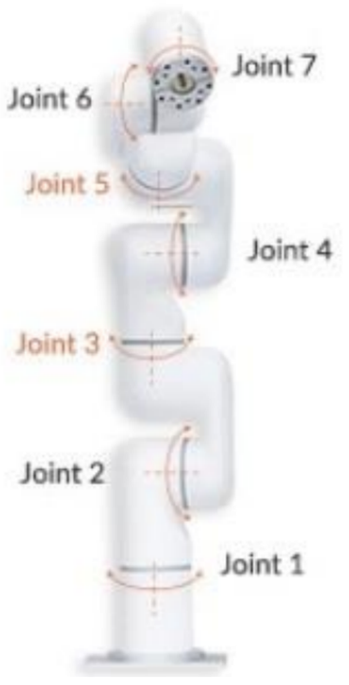
INTRODUCTION

In this study, we went over the redundant degrees of freedom in a robotic manipulator's kinematic control. More precisely, we were focused on monitoring a preset route for the robotic manipulator's end while avoiding certain space-based impediments.

DESCRIPTION OF THE ROBOTIC OPERATOR

The robot that we will use for the implementation of this project is the xArm 7 which belongs to the cobots category (collaborative robots).

KINEMATIC ANALYSIS



$$\boldsymbol{A_1^0(q_1)} = Rot(z, q_1) \cdot Tra(z, l_1), \quad \boldsymbol{l_1} = 26.7cm$$

$$\boldsymbol{A_2^1(q_2)} = Rot\left(x, -\frac{\pi}{2}\right) \cdot Rot(z, q_2)$$

$$\boldsymbol{A_3^2(q_3)} = Rot\left(x, +\frac{\pi}{2}\right) \cdot Rot(z, q_3) \cdot Tra(z, l_2), \quad \boldsymbol{l_2} = 29.3cm$$

$$\boldsymbol{A_4^3(q_4)} = Rot\left(x, +\frac{\pi}{2}\right) \cdot Tra(x, l_3) \cdot Rot(z, q_4), \quad \boldsymbol{l_3} = 5.25cm$$

$$\boldsymbol{A_5^4(q_5)} = Rot\left(x, +\frac{\pi}{2}\right) \cdot Tra(x, l_4 \sin\theta_1) \cdot Rot(z, q_5) \cdot Tra(z, l_4 \cos\theta_1), \quad \begin{matrix} \boldsymbol{l_4} = 35.12\ cm \\ \boldsymbol{\theta_1} = 0.2225\ rad \end{matrix}$$

$$\boldsymbol{A_6^5(q_5)} = Rot\left(x, +\frac{\pi}{2}\right) \cdot Rot(z, q_6)$$

$$\boldsymbol{A_7^6(q_6)} = Rot\left(x, -\frac{\pi}{2}\right) \cdot Tra(x, l_5 \sin\theta_2) \cdot Rot(z, q_7) \cdot Tra(z, l_5 \cos\theta_2), \quad \begin{matrix} \boldsymbol{l_5} = 12.32\ cm \\ \boldsymbol{\theta_2} = 0.6646\ rad \end{matrix}$$

Using the above given transition matrices we calculate matrices A_1^0,A_2^0,A_3^0,A_4^0,A_5^0, A_6^0, A_7^0 as well as the Jacobian matrix in the following jupyter notebook:

kinematic_analysis.ipynb

The resulting matrices are the following:

$$A_1^0(q1)=\begin{pmatrix} c1 & -s1 & 0 & 0 \\ s1 & c1 & 0 & 0 \\ 0 & 0 & 1 & l1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$A_2^1(q2)=\begin{pmatrix} c2 & -s2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -s2 & -c2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$A_3^2(q3)=\begin{pmatrix} c3 & -s3 & 0 & 0 \\ 0 & 0 & -1 & -l2 \\ s3 & c3 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$A_4^3(q4)=\begin{pmatrix} c4 & -s4 & 0 & l3 \\ 0 & 0 & -1 & 0 \\ s4 & c4 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$A_5^4(q5)=\begin{pmatrix} c5 & -s5 & 0 & l4\cdot\sin(\theta_1) \\ 0 & 0 & -1 & -l4\cdot\sin(\theta_1) \\ s5 & c5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

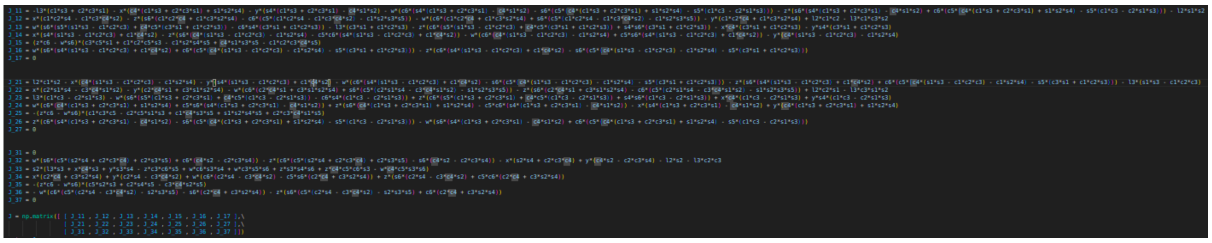
$$A_6^5(q6)=\begin{pmatrix} c6 & -s6 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ s6 & c6 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$A_7^6(q7)=\begin{pmatrix} c7 & -s7 & 0 & l5\cdot\sin(\theta_2) \\ 0 & 0 & 1 & l5\cdot\sin(\theta_2) \\ -s7 & -c7 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

For the Jacobian matrix we used the following on the position vector of A_7^0 :

$$\mathbf{J}=\begin{bmatrix}\dfrac{\partial \mathbf{f}}{\partial x_1} & \cdots & \dfrac{\partial \mathbf{f}}{\partial x_n}\end{bmatrix}=\begin{bmatrix}\nabla f_1 \\ \vdots \\ \nabla f_m\end{bmatrix}=\begin{bmatrix}\dfrac{\partial f_1}{\partial x_1} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_m}{\partial x_1} & \cdots & \dfrac{\partial f_m}{\partial x_n}\end{bmatrix}$$

and the result was(obviously we cannot display it fully):



```
J_11 = -l3*(c1*s3 + c2*c3*s1) - x*(c4*(c1*s3 + c2*c3*s1) + s1*s2*s4) - y*(s4*(c1*s3 + c2*c3*s1) - c4*s1*s2) - w*(c6*(s4*(c1*s3 + c2*c3*s1) - c4*s1*s2) - s6*(c5*(c4*(c1*s3 + c2*c3*s1) + s1*s2*s4) - s5*(c1*c3 - c2*s1*s3))) - z*(s6*(s4*(c1*s3 + c2*c3*s1) - c4*s1*s2) + c6*(c5*(c4*(c1*s3 + c2*c3*s1) + s1*s2*s4) - s5*(c1*c3 - c2*s1*s3))) - l2*s1*s2
J_12 = x*(c1*c2*s4 - c1*c3*c4*s2) - z*(s6*(c1*c2*c4 + c1*c3*s2*s4) - c6*(c5*(c1*c2*s4 - c1*c3*c4*s2) - c1*s2*s3*s5)) - w*(c6*(c1*c2*c4 + c1*c3*s2*s4) + s6*(c5*(c1*c2*s4 - c1*c3*c4*s2) - c1*s2*s3*s5)) - y*(c1*c2*c4 + c1*c3*s2*s4) + l2*c1*c2 - l3*c1*c3*s2
J_13 = w*(s6*(s5*(s1*s3 - c1*c2*c3) + c4*c5*(c3*s1 + c1*c2*s3)) - c6*s4*(c3*s1 + c1*c2*s3)) - l3*(c3*s1 + c1*c2*s3) - z*(c6*(s5*(s1*s3 - c1*c2*c3) + c4*c5*(c3*s1 + c1*c2*s3)) + s4*s6*(c3*s1 + c1*c2*s3)) - x*c4*(c3*s1 + c1*c2*s3) - y*s4*(c3*s1 + c1*c2*s3)
J_14 = x*(s4*(s1*s3 - c1*c2*c3) + c1*c4*s2) - z*(s6*(c4*(s1*s3 - c1*c2*c3) - c1*s2*s4) - c5*c6*(s4*(s1*s3 - c1*c2*c3) + c1*c4*s2)) - w*(c6*(c4*(s1*s3 - c1*c2*c3) - c1*s2*s4) + c5*s6*(s4*(s1*s3 - c1*c2*c3) + c1*c4*s2)) - y*(c4*(s1*s3 - c1*c2*c3) - c1*s2*s4)
J_15 = (z*c6 - w*s6)*(c3*c5*s1 + c1*c2*c5*s3 - c1*s2*s4*s5 + c4*s1*s3*s5 - c1*c2*c3*c4*s5)
J_16 = w*(s6*(s4*(s1*s3 - c1*c2*c3) + c1*c4*s2) + c6*(c5*(c4*(s1*s3 - c1*c2*c3) - c1*s2*s4) - s5*(c3*s1 + c1*c2*s3))) - z*(c6*(s4*(s1*s3 - c1*c2*c3) + c1*c4*s2) - s6*(c5*(c4*(s1*s3 - c1*c2*c3) - c1*s2*s4) - s5*(c3*s1 + c1*c2*s3)))
J_17 = 0


J_21 = l2*c1*s2 - x*(c4*(s1*s3 - c1*c2*c3) - c1*s2*s4) - y*(s4*(s1*s3 - c1*c2*c3) + c1*c4*s2) - w*(c6*(s4*(s1*s3 - c1*c2*c3) + c1*c4*s2) - s6*(c5*(c4*(s1*s3 - c1*c2*c3) - c1*s2*s4) - s5*(c3*s1 + c1*c2*s3))) - z*(s6*(s4*(s1*s3 - c1*c2*c3) + c1*c4*s2) + c6*(c5*(c4*(s1*s3 - c1*c2*c3) - c1*s2*s4) - s5*(c3*s1 + c1*c2*s3))) - l3*(s1*s3 - c1*c2*c3)
J_22 = x*(c2*s1*s4 - c3*c4*s1*s2) - y*(c2*c4*s1 + c3*s1*s2*s4) - w*(c6*(c2*c4*s1 + c3*s1*s2*s4) + s6*(c5*(c2*s1*s4 - c3*c4*s1*s2) - s1*s2*s3*s5)) - z*(s6*(c2*c4*s1 + c3*s1*s2*s4) - c6*(c5*(c2*s1*s4 - c3*c4*s1*s2) - s1*s2*s3*s5)) + l2*c2*s1 - l3*c3*s1*s2
J_23 = l3*(c1*c3 - c2*s1*s3) - w*(s6*(s5*(c1*s3 + c2*c3*s1) + c4*c5*(c1*c3 - c2*s1*s3)) - c6*s4*(c1*c3 - c2*s1*s3)) + z*(c6*(s5*(c1*s3 + c2*c3*s1) + c4*c5*(c1*c3 - c2*s1*s3)) + s4*s6*(c1*c3 - c2*s1*s3)) + x*c4*(c1*c3 - c2*s1*s3) + y*s4*(c1*c3 - c2*s1*s3)
J_24 = w*(c6*(c4*(c1*s3 + c2*c3*s1) + s1*s2*s4) + c5*s6*(s4*(c1*s3 + c2*c3*s1) - c4*s1*s2)) + z*(s6*(c4*(c1*s3 + c2*c3*s1) + s1*s2*s4) - c5*c6*(s4*(c1*s3 + c2*c3*s1) - c4*s1*s2)) - x*(s4*(c1*s3 + c2*c3*s1) - c4*s1*s2) + y*(c4*(c1*s3 + c2*c3*s1) + s1*s2*s4)
J_25 = -(z*c6 - w*s6)*(c1*c3*c5 - c2*c5*s1*s3 + c1*c4*s3*s5 + s1*s2*s4*s5 + c2*c3*c4*s1*s5)
J_26 = z*(c6*(s4*(c1*s3 + c2*c3*s1) - c4*s1*s2) - s6*(c5*(c4*(c1*s3 + c2*c3*s1) + s1*s2*s4) - s5*(c1*c3 - c2*s1*s3))) - w*(s6*(s4*(c1*s3 + c2*c3*s1) - c4*s1*s2) + c6*(c5*(c4*(c1*s3 + c2*c3*s1) + s1*s2*s4) - s5*(c1*c3 - c2*s1*s3)))
J_27 = 0


J_31 = 0
J_32 = w*(s6*(c5*(s2*s4 + c2*c3*c4) + c2*s3*s5) + c6*(c4*s2 - c2*c3*s4)) - z*(c6*(c5*(s2*s4 + c2*c3*c4) + c2*s3*s5) - s6*(c4*s2 - c2*c3*s4)) - x*(s2*s4 + c2*c3*c4) + y*(c4*s2 - c2*c3*s4) - l2*s2 - l3*c2*c3
J_33 = s2*(l3*s3 + x*c4*s3 + y*s3*s4 - z*c3*c6*s5 + w*c6*s3*s4 + w*c3*s5*s6 + z*s3*s4*s6 + z*c4*c5*c6*s3 - w*c4*c5*s3*s6)
J_34 = x*(c2*c4 + c3*s2*s4) + y*(c2*s4 - c3*c4*s2) + w*(c6*(c2*s4 - c3*c4*s2) - c5*s6*(c2*c4 + c3*s2*s4)) + z*(s6*(c2*s4 - c3*c4*s2) + c5*c6*(c2*c4 + c3*s2*s4))
J_35 = -(z*c6 - w*s6)*(c5*s2*s3 + c2*s4*s5 - c3*c4*s2*s5)
J_36 = - w*(c6*(c5*(c2*s4 - c3*c4*s2) - s2*s3*s5) - s6*(c2*c4 + c3*s2*s4)) - z*(s6*(c5*(c2*s4 - c3*c4*s2) - s2*s3*s5) + c6*(c2*c4 + c3*s2*s4))
J_37 = 0

J = np.matrix([ [ J_11 , J_12 , J_13 , J_14 , J_15 , J_16 , J_17 ],\
                [ J_21 , J_22 , J_23 , J_24 , J_25 , J_26 , J_27 ],\
                [ J_31 , J_32 , J_33 , J_34 , J_35 , J_36 , J_37 ]])
```

The aim of this task is the implementation of an algorithm for tracking a trajectory from the final action element of the robotic device (xArm 7) with simultaneous obstacle avoidance. More specifically, the arm's end action element is asked to perform rectilinear periodic motion between positions PA and PB. So, to determine the appropriate angular velocity of the arm joints at each time instant, we can break the robotic process into two sub-tasks:
- sub-task A: designing the trajectory and controlling the position of the end-of-arm tool to perform the desired motion
- subtask B: controlling the distance from obstacles to avoid collision.

Sub-task A - Trajectory planning

Initially we need to design the trajectory which we chose to do with a polynomial interpolation. Since after the initialization process the robotic arm is at the position Pinit = (0.617 , 0 , 0.199) , we design the periodic movement:
Pinit ⇒ Pb ⇒Pa ⇒ Pb ⇒Pa⇒ …
We have drawn the required location of the final action element by interpolation polynomials, particularly by dividing it into three sub-phases, to guarantee smooth motion in terms of velocity and acceleration. The intermediate phase, when we want the end-effector to move smoothly and with a steady speed, is what makes up the motion. A polynomial of first degree with regard to time was selected as a result. The remaining two stages guarantee that the tool moves smoothly as it approaches the two end locations, preventing sudden changes in speed from stressing the joints. Fifth-degree polynomials were chosen because we want the final action element's acceleration and velocity to be continuous. As a result, we can calculate the time function of the required end-effector location using the following formula(only for y axis):

Phase 1: $p_{1d}(t) = a_0 + a_1 \cdot t + a_2 \cdot t^2 + a_3 \cdot t^3 + a_4 \cdot t^4 + a_5 \cdot t^5$, for $t < T_1$

Phase 2: $p_{2d}(t) = b_0 + b_1 \cdot t$, for $T_1 < t < T_2$

Phase 3: $p_{3d}(t) = c_0 + c_1 \cdot t + c_2 \cdot t^2 + c_3 \cdot t^3 + c_4 \cdot t^4 + c_5 \cdot t^5$, for $T_2 < t < T$

,where T is the time it takes the robotic arm to move from the one end point to the other, T_1 is the end time of the acceleration and T_2 is the start time of the deceleration.

From the derivatives of the above equations we can also find the speeds and the accelerations for each phase:

speeds:
Phase 1: $v_{1d}(t) = a_1 + 2 \cdot a_2 \cdot t + 3 \cdot a_3 \cdot t^2 + 4 \cdot a_4 \cdot t^3 + 5 \cdot a_5 \cdot t^4$, for $t < T_1$

Phase 2: $v_{2d}(t) = b_1$, for $T_1 < t < T_2$

Phase 3: $v_{3d}(t) = c_1 + 2 \cdot c_2 \cdot t + 3 \cdot c_3 \cdot t^2 + 4 \cdot c_4 \cdot t^3 + 5 \cdot c_5 \cdot t^4$, for $T_2 < t < T$

accelerations:

Phase 1: $a_{1d}(t) = 2 \cdot a_2 + 6 \cdot a_3 \cdot t + 7 \cdot a_4 \cdot t^2 + 20 \cdot a_5 \cdot t^3$, for $t < T_1$

Phase 2: $a_{2d}(t) = 0$, for $T_1 < t < T_2$

Phase 3: $a_{3d}(t) = 2 \cdot c_2 + 6 \cdot c_3 \cdot t + 12 \cdot c_4 \cdot t^2 + 20 \cdot c_5 \cdot t^3$, for $T_2 < t < T$

We have these initial conditions:

for positions:
p1d(t0) = yA (initial position)
p1d(T1) = p2d(T1) (continuity of positions)
p2d(T2) = p3d(T2)
p3d(T) = yA (final position)

similarly, for speed and accelaration:
u1d(t0) = uA
u1d(T1) = u2d(T1)
u2d(T2) = u3d(T2)
u3d(T) = uB

a1d(t0) = aA
a1d(T1) = a2d(T1)
a2d(T2) = a3d(T2)
a3d(T) = aB

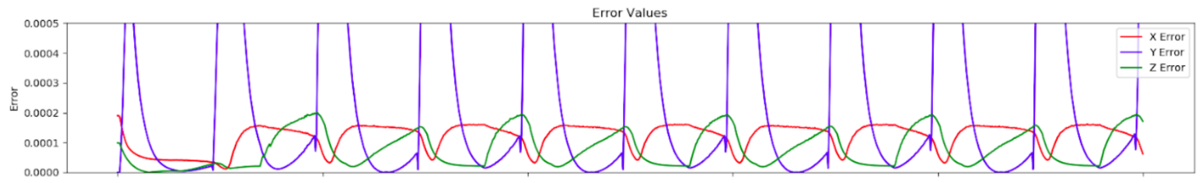
To solve the above equation system and to find the coefficients for each polynomial we use the function coeff(), that takes as arguments start and end positions, start and end time. It outputs 14 coefficients, 6 for each polynomial(accelerating, decelerating) and 2 for the constant motion between.

We need a position controller(the system is non-linear) to adjust the positional error:

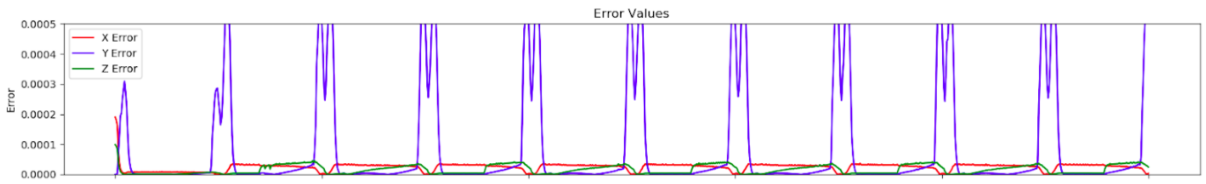
q’ = J^+*(u1d + K1*error), where error = p1d - p1real

We find the best K1 parameter value by testing:

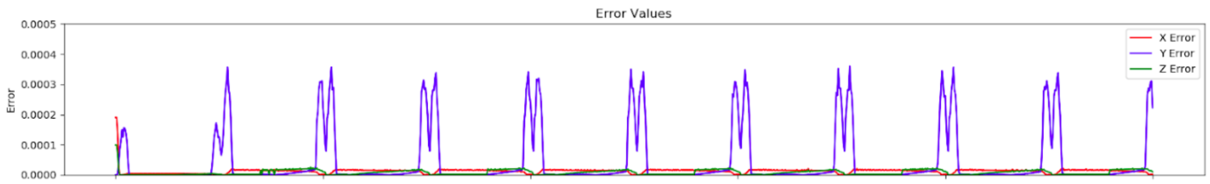
for K1 = 10:



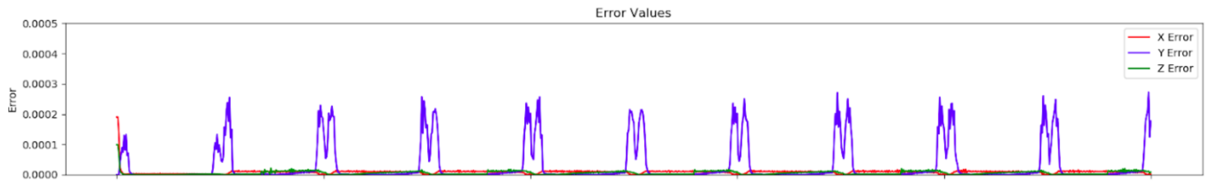
for K1 = 50:



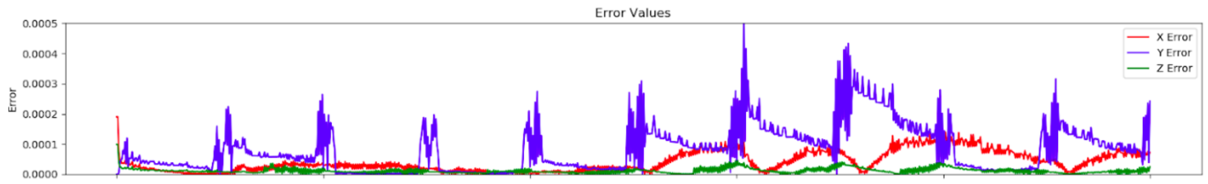
for K1 = 100:



for K1 = 150:



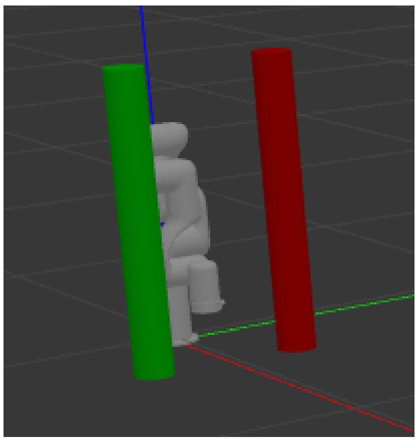
for K1 = 200:



So, obviously we select K1 = 150 (it has the lowest y error)

Sub-task B - Obstacle Avoidance

As second subtask, we have to prevent collision between the robotic manipulator and the two cylindrical static obstacles, that are shown as red and green in the simulation below:



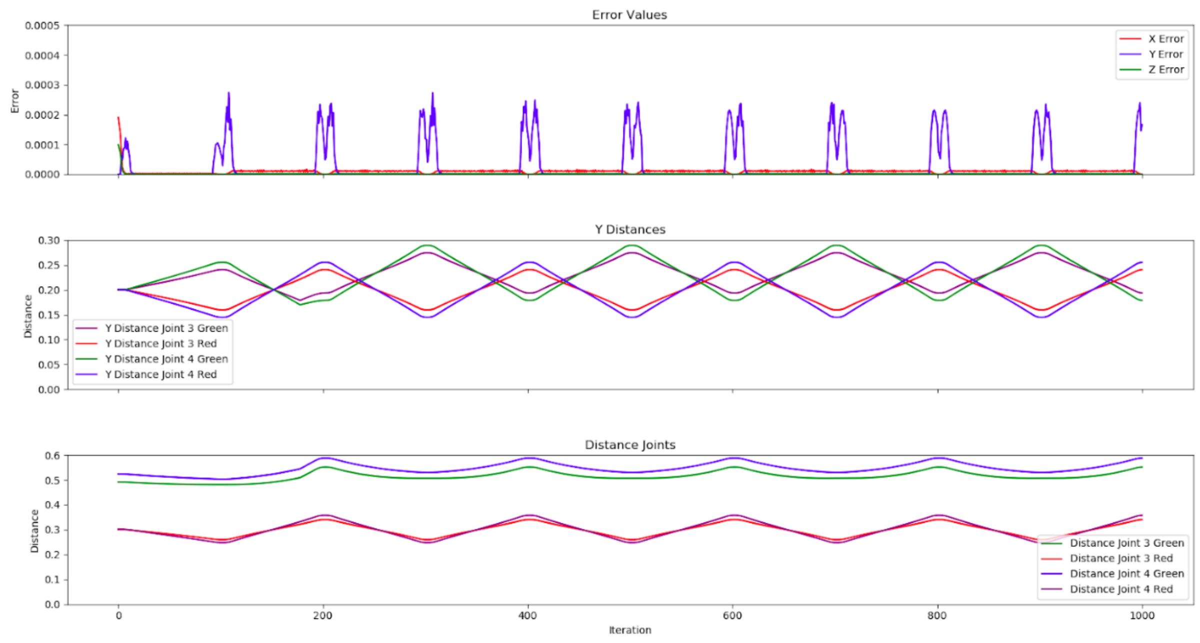
By observation in the gazebo simulation environment, q3 and q4 are the most likely joints to collide with the aforementioned obstacles.
For this reason, we decide to create 2 criterion equations for the y-axis distances of these components:

D1= ½ * Kc * (y_{q4} - $y_{obstacle\_red}$)^2
D2 = ½ * Kc * (y_{q3} - $y_{obstacle\_green}$)^2

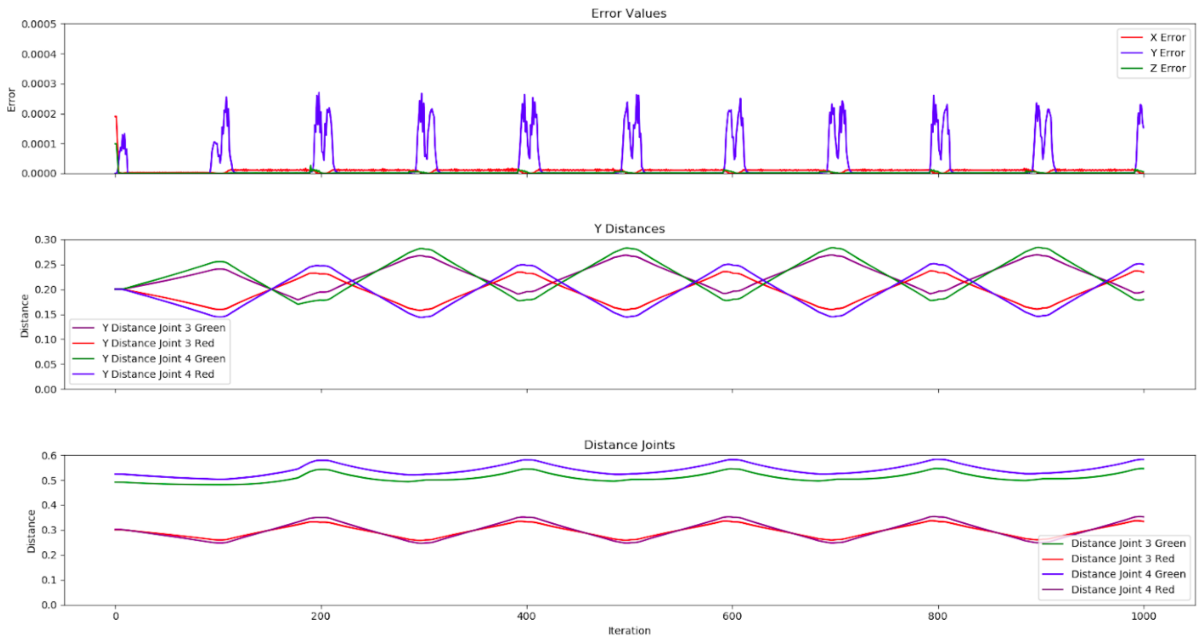
that calculate the parabolic distances between joints q3 q4 (the middle of them) and the corresponding nearest obstacle.

<u>We find the best Kc parameter value by testing(observe the middle diagram):</u>

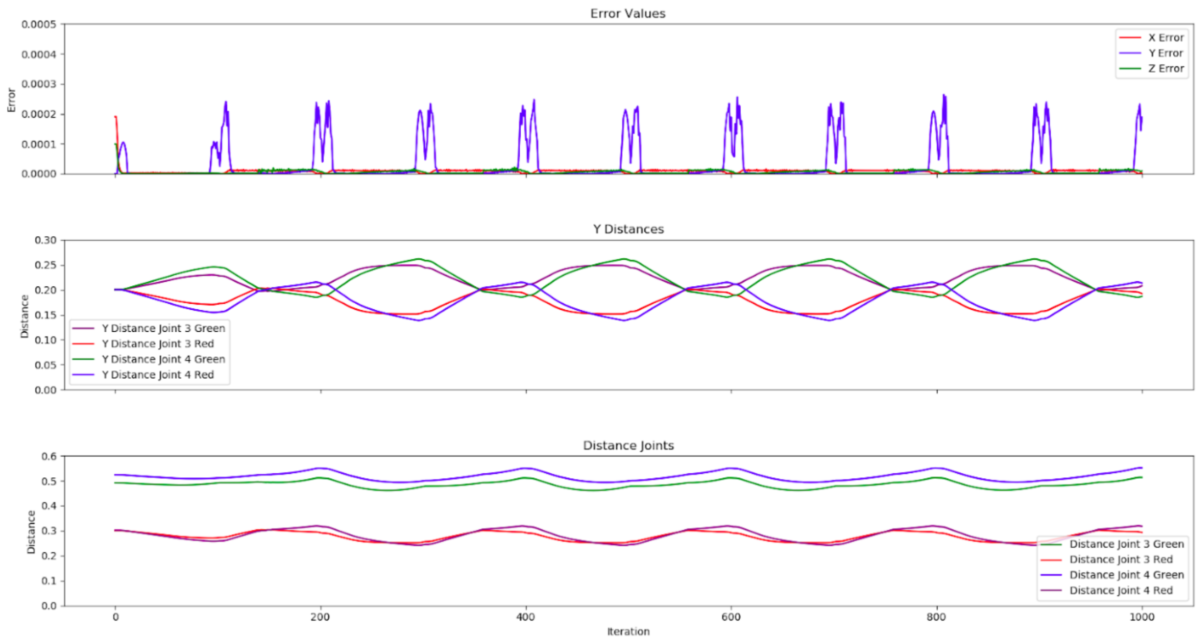
for Kc = 2:



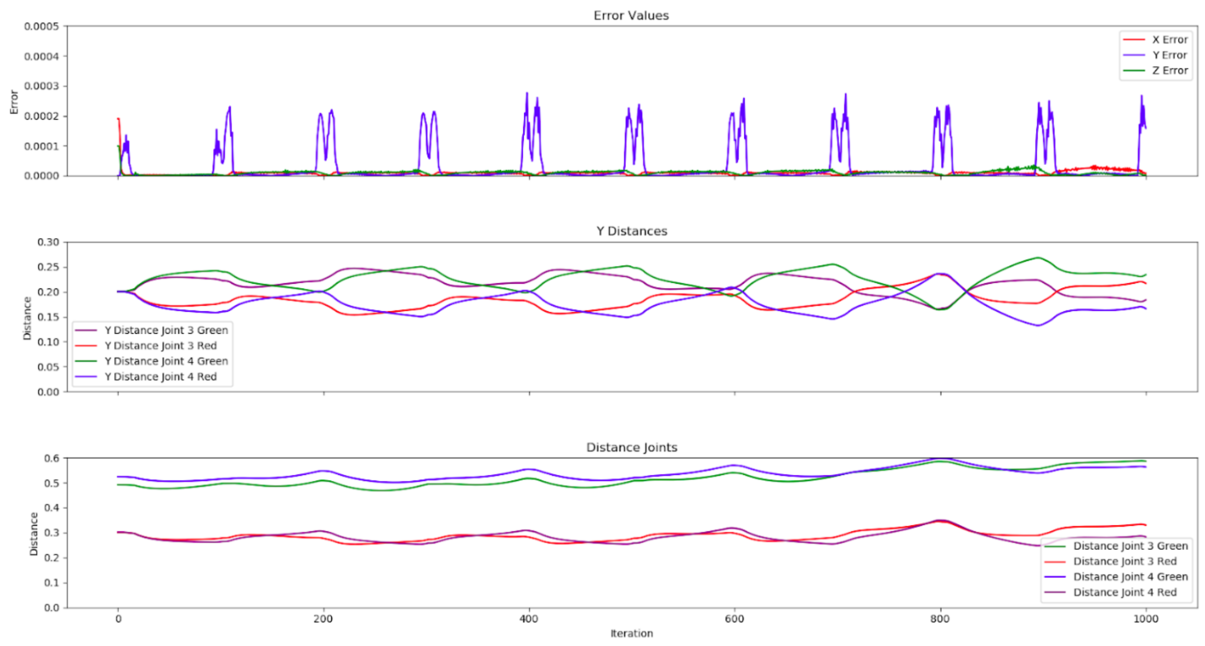
for Kc = 5:



for Kc = 10:



for Kc = 20:



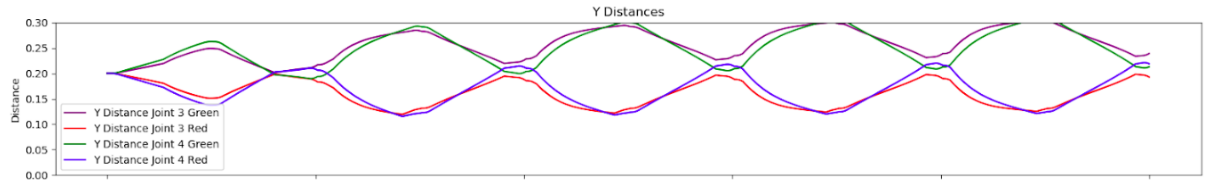
So, we observe that <u>Kc = 10</u> is the best one (actually Kc = 20 has better y distances than Kc = 10, however it causes a more abrupt movement, so we select 10 that seems to cause a more uniform oscillating movement). For Kc = 5 or 10 , the distance to the obstacles is very small and it cannot guarantee that we will not have a collision, even in another joint.

In joint space, the equation (for the joint velocities) will be:

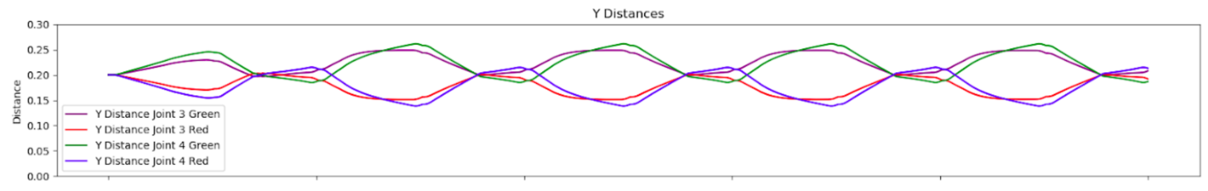
$q' = K2^* (I - J^+J)^*\xi 1 + K3^* (I - J^+J)^*\xi 2$, where $\xi 1$ and $\xi 2$ are the partial derivative vectors of D1 and D2 to the velocities qi of each joint.

<u>We find the best K2,K3 parameter combination value by testing:</u>

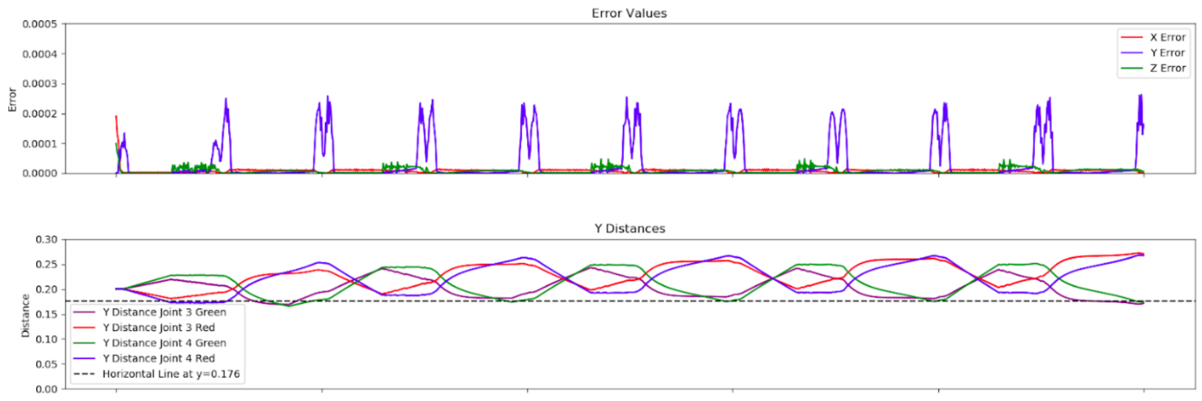
for K2 = 5, K3 = 10:



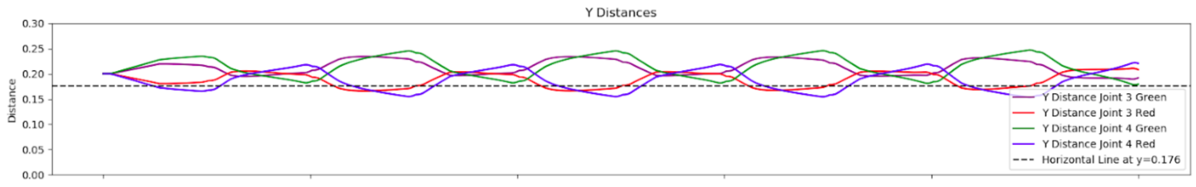
for K2 = 10, K3 = 15:



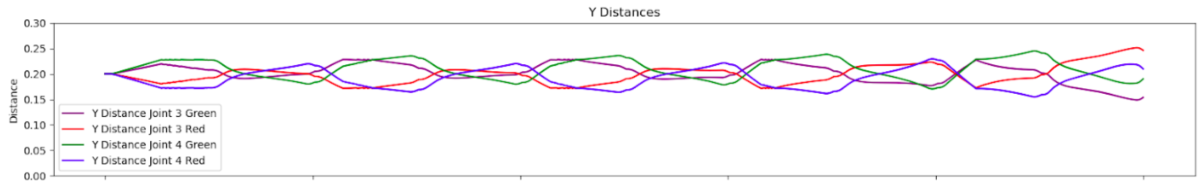
for K2 = 15, K3 = 15:(we also display the error because it is the only one that has z-axis error):



for K2 = 15, K3 = 20:



for K2 = 20, K3 = 25:



We observe that the best combination is K2 = 10, K3 = 15 and next is K2 = 20, K3 = 25. Our selection criterions were error stability, obstacle distance, movement smoothness.

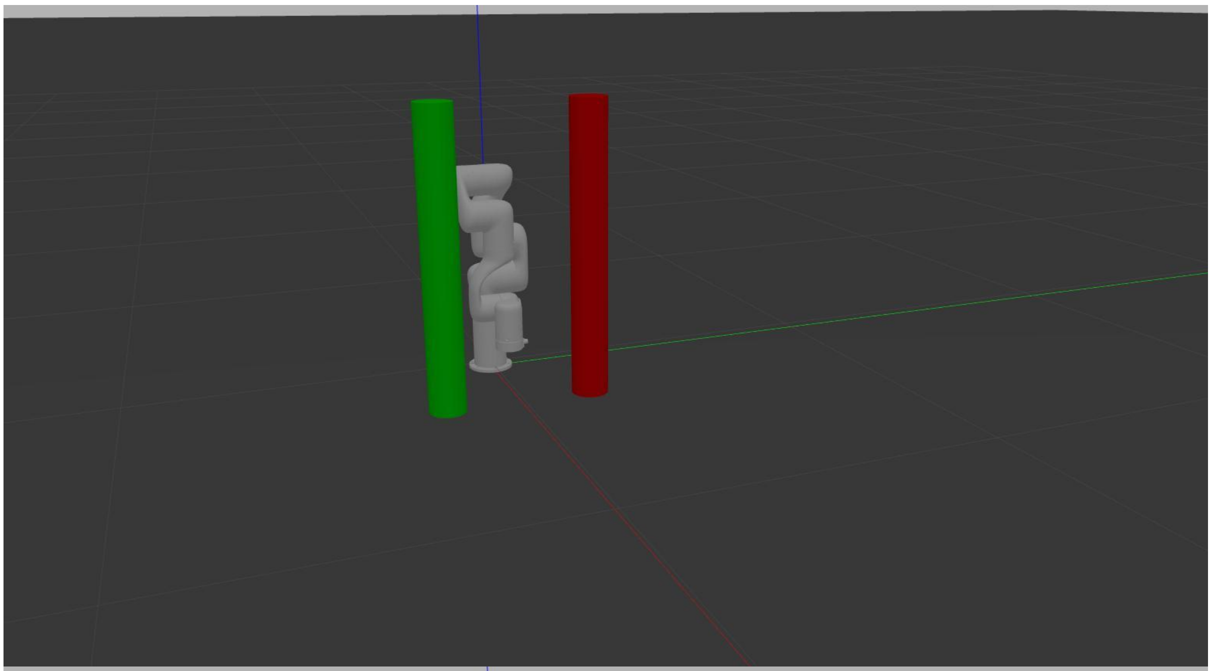
The final equation of our model (for both subtasks):

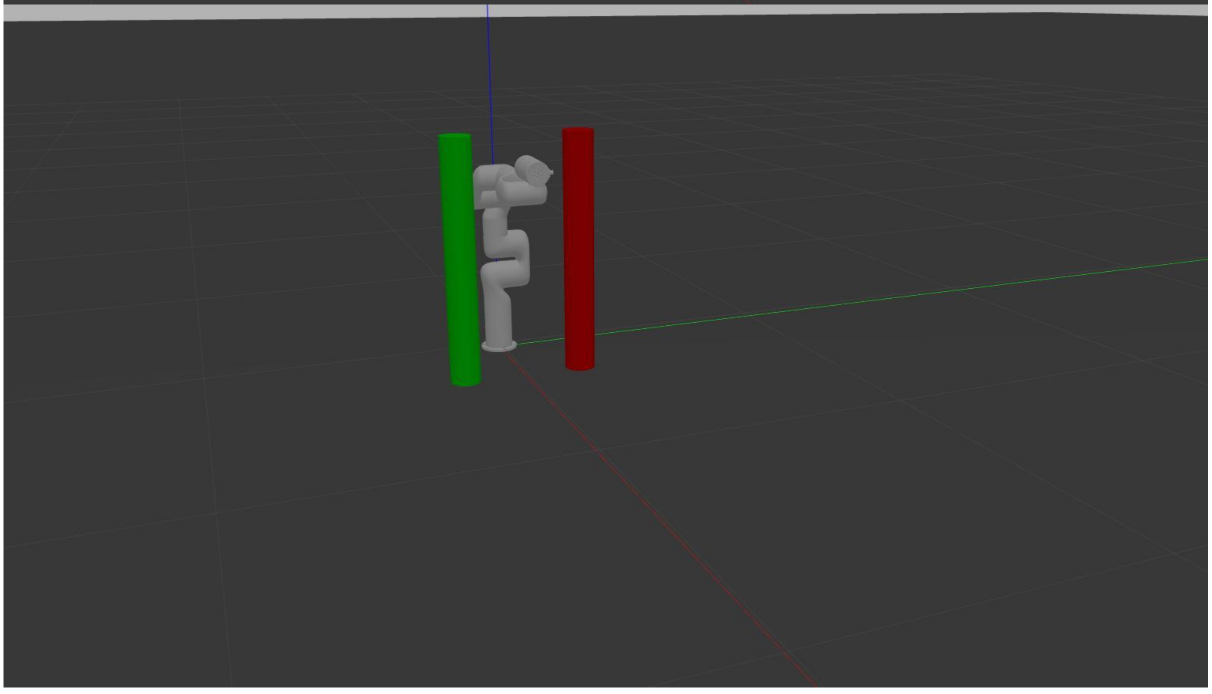
$q' = J^{+} *(u1d + K1*error) + K2* (I - J^{+}J)*\xi 1 + K3* (I - J^{+}J)*\xi 2$

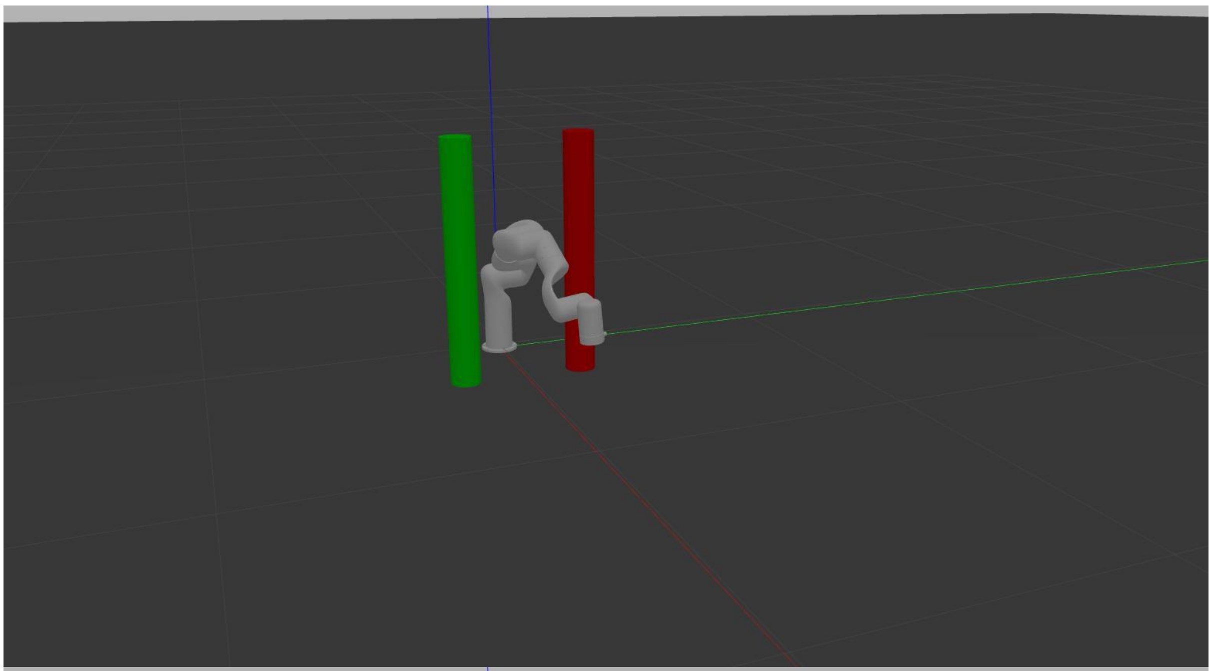
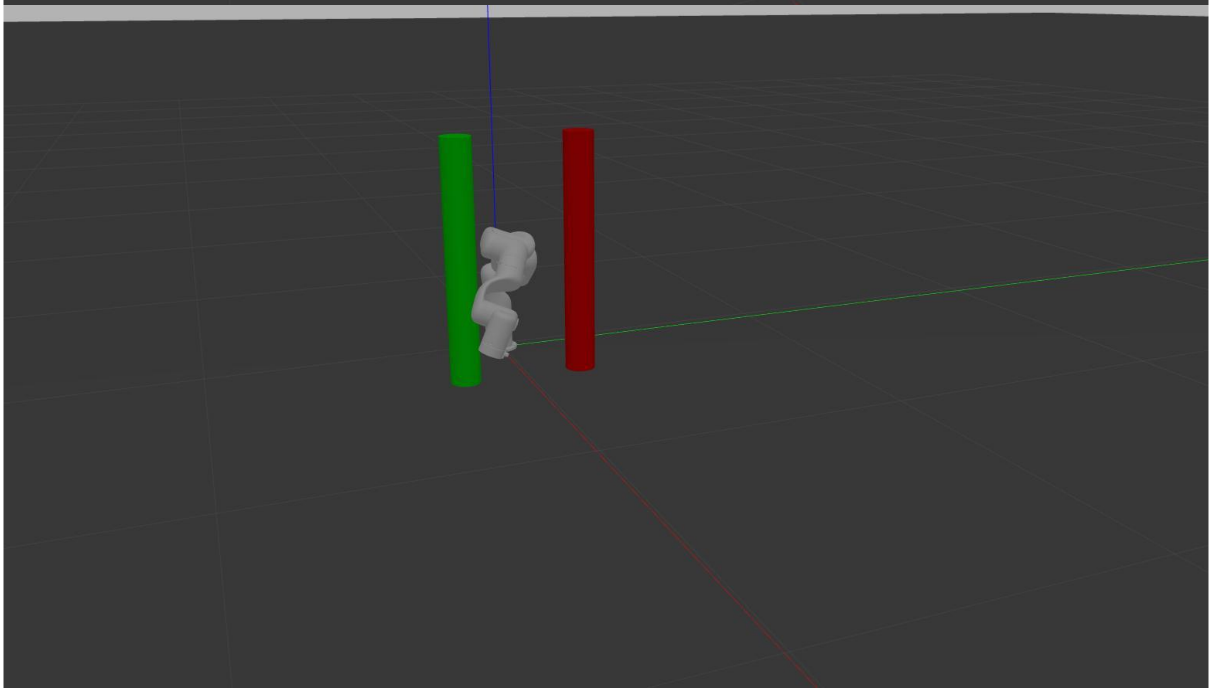
From the above equation we can compute the qi with integration

SIMULATION

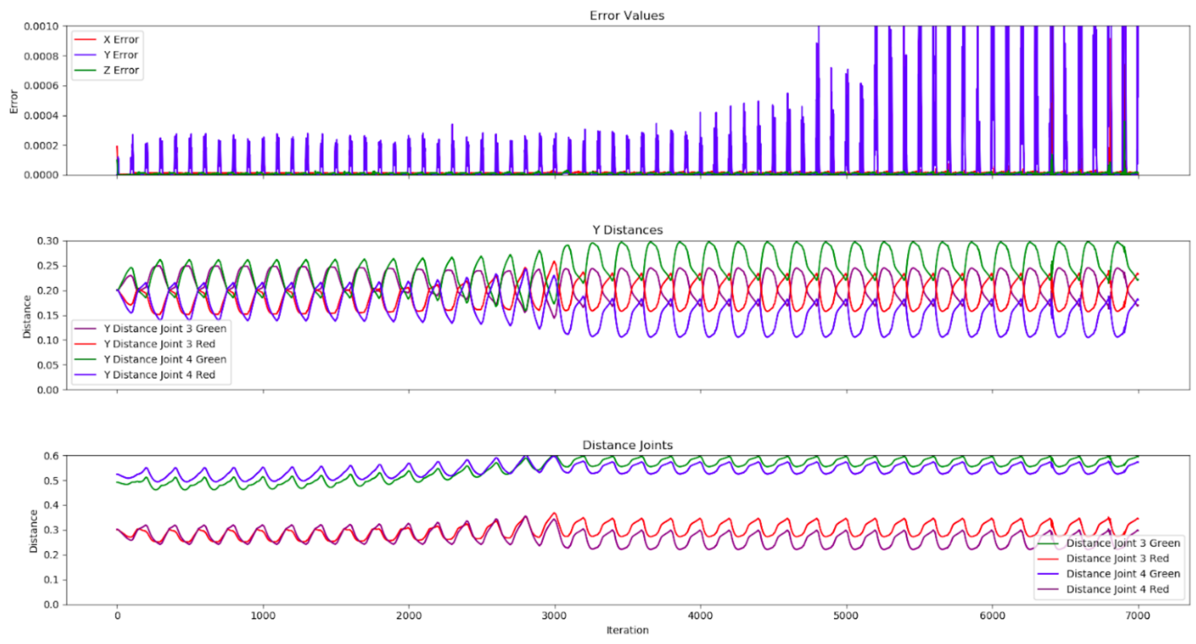
Below we can observe some instances of the simulation in the gazebo environment.



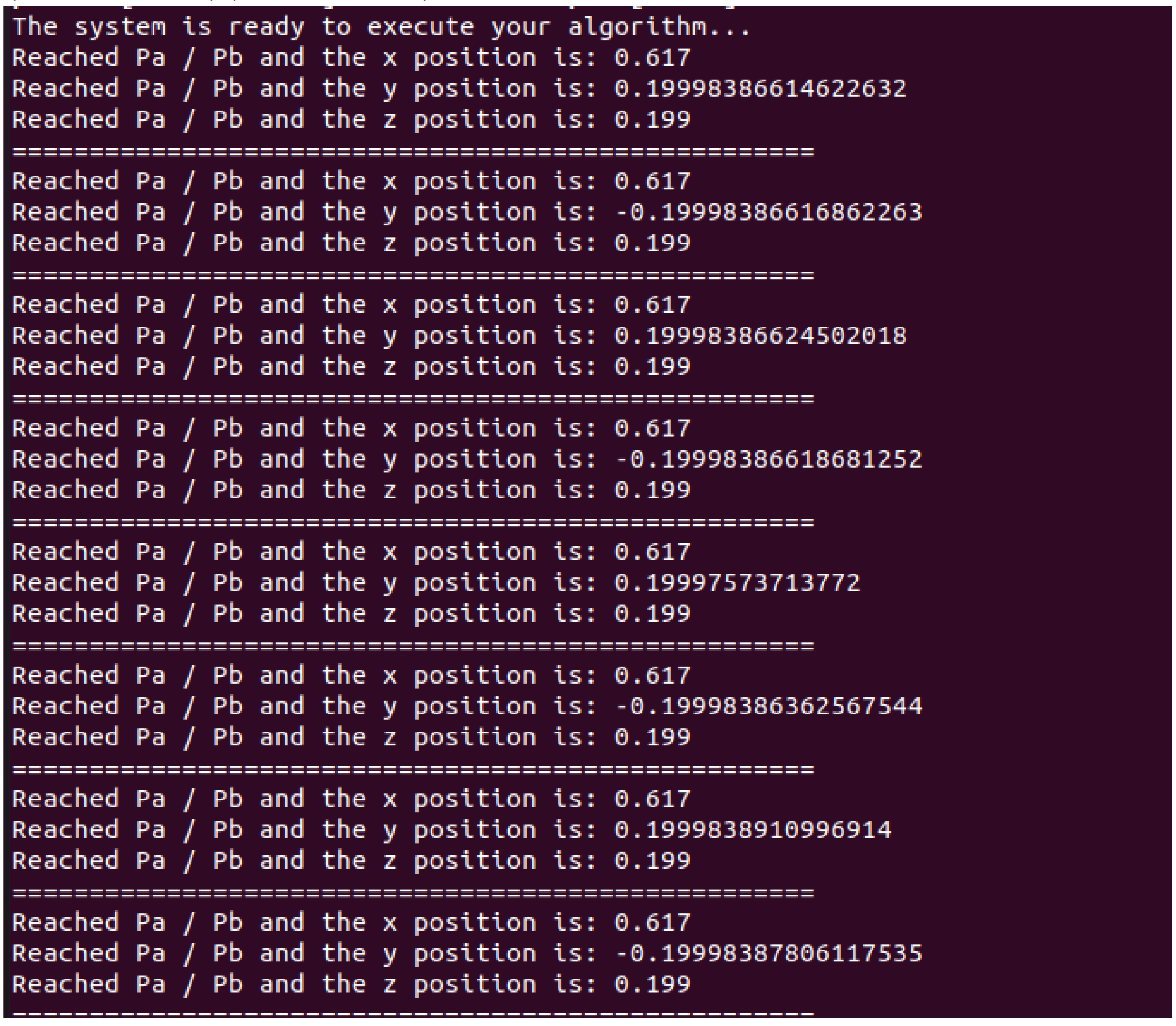


In the diagrams below we observe that our model can achieve at least 10 periodic movements before
it starts getting out of hand (increasing positional errors and decreasing obstacle distances).
One way to make it more stable would be to reset to its initial conditions after a number of movements.
Another one could be to add more control constraints that would increase the computational complexity of our model.



Finally we also print the (x,y,z) positions when the robotic arm reaches our desired positions where we can see that we have an insignificant error for the y axis and actually reach the positions (0.617,+0.2,0.199),(0.617,-0.2,0.199).



```
The system is ready to execute your algorithm...
Reached Pa / Pb and the x position is: 0.617
Reached Pa / Pb and the y position is: 0.19998386614622632
Reached Pa / Pb and the z position is: 0.199
=====================================================
Reached Pa / Pb and the x position is: 0.617
Reached Pa / Pb and the y position is: -0.19998386616862263
Reached Pa / Pb and the z position is: 0.199
=====================================================
Reached Pa / Pb and the x position is: 0.617
Reached Pa / Pb and the y position is: 0.19998386624502018
Reached Pa / Pb and the z position is: 0.199
=====================================================
Reached Pa / Pb and the x position is: 0.617
Reached Pa / Pb and the y position is: -0.19998386618681252
Reached Pa / Pb and the z position is: 0.199
=====================================================
Reached Pa / Pb and the x position is: 0.617
Reached Pa / Pb and the y position is: 0.19997573713772
Reached Pa / Pb and the z position is: 0.199
=====================================================
Reached Pa / Pb and the x position is: 0.617
Reached Pa / Pb and the y position is: -0.19998386362567544
Reached Pa / Pb and the z position is: 0.199
=====================================================
Reached Pa / Pb and the x position is: 0.617
Reached Pa / Pb and the y position is: 0.1999838910996914
Reached Pa / Pb and the z position is: 0.199
=====================================================
Reached Pa / Pb and the x position is: 0.617
Reached Pa / Pb and the y position is: -0.19998387806117535
Reached Pa / Pb and the z position is: 0.199
```