Dominick Mulder dam3888
Nicolas Brissonneau nb24488

Programming Assignment

Introduction

In this report, we aimed to develop optimization algorithms to control the Panda robot while taking constraints into account. We will show and compare fairly the performances of different approaches in similar contexts. The functions we have been using include those we wrote in our previous assignments, but also Matlab built-in functions such as lsqlin and fmincon.

We discussed potential approaches to the problems in the assignment and agreed on a plan for solving these problems. Regarding the written Homework Assignments, Dominick primarily focused on HA2 and Nicolas on HA1. Regarding the Programming Assignments, both Nicolas and Dominick have walked through the steps of deriving algorithms to solve for the questions. Dominick has investigated the use of non-linear optimization through fmincon and Nicolas has investigated the use of least-square linear optimization through lsqlin.

To summarize, we distributed the workload as follows:
- Programming assignment: **50% Dominick** and **50% Nicolas.**
- Homework assignment: **50% Nicolas** and **50% Dominick.**

PA

In this assignment, we once again operate the robot Panda but this time adding a 100mm long cylindrical tool attached to Panda's tool frame. The endpoint of this cylinder will be called the tip for the remaining of this report.

Constraint in tip position and joint limits

First, we want to be able to drag the tip accurately to a desired position defined in space frame, we will call this position p_{goal}. In order to do so we need to express the distance between the tip and p_{goal}, as a function of the transformation which will guide us from the tip p_{tip} to the end target p_{goal}. We have thus defined $D(\Delta\vec{x})$:

$$D(\Delta\vec{x}) = \Delta F(\vec{q}, \Delta\vec{q}) \cdot F \cdot p_{tip} - p_{goal}$$

By introducing $\vec{t} = F \cdot p_{tip}$ we can express $D(\Delta\vec{x})$ as:

$$D(\Delta\vec{x}) = \vec{\alpha} \times \vec{t} + \vec{\varepsilon} + \vec{t} - \vec{p}_{goal}, \text{ where } \vec{\alpha} = J_{\vec{\alpha}}(\vec{q})\,\Delta\vec{q};\ \vec{\varepsilon} = J_{\vec{\varepsilon}}(\vec{q})\,\Delta\vec{q}$$

With $J_{\vec{\alpha}}(\vec{q})$ and $J_{\vec{\varepsilon}}(\vec{q})$ respectively the rotational and translational components of the robot's Jacobian matrix. Adding to this the knowledge of the joint limits, we have all the elements to express our problem as a least-square linear optimization:

$$\Delta\vec{q} = argmin_{\Delta\vec{q}} \ \| \Delta\vec{q} \|^2$$

Subject to:

$$\| D(\Delta\vec{x}) \| \leq 3$$

$$\vec{q}_L - \vec{q} \leq \Delta\vec{q} \leq \vec{q}_U - \vec{q}$$

We have chosen to express the optimization with the $A\Delta\vec{q} = b$ format by noticing that:

$$D(\Delta\vec{x}) = \vec{\alpha} \times \vec{t} + \vec{\varepsilon} + \vec{t} - \vec{p}_{goal}$$

$$= -\vec{t} \times J_{\vec{\alpha}}(\vec{q})\,\Delta\vec{q} + J_{\vec{\varepsilon}}(\vec{q})\,\Delta\vec{q} + \vec{t} - \vec{p}_{goal}$$

$$= \left(-\left[\vec{t}\right]_X J_{\vec{\alpha}}(\vec{q}) + J_{\vec{\varepsilon}}(\vec{q}) \right) \Delta\vec{q} + \vec{t} - \vec{p}_{goal}$$

With $\left[\vec{t}\right]_X$ the cross-product matrix form of $\vec{t}$. As we want $\| D(\Delta\vec{x}) \| \leq 3$, we can express:

$$A = J_{\vec{\varepsilon}}(\vec{q}) - \left[\vec{t}\right]_X J_{\vec{\alpha}}(\vec{q})$$

$$b = \vec{p}_{goal} - \vec{t}$$

We now have a solution for x which minimizes the distance, respects the joints limits and can keep iterating until a solution is found with a value inferior to 3mm.

Constraint in tip position, joints limits and orientation

Now, we want to maintain the previous objective function minimizing for the angular displacements, while keeping the previously defined constraints and adding a second objective function aiming at minimizing the rotational component of the transformation. The second objective function is thus $\| \alpha \times R \cdot \vec{z} \|^2$, one can notice that $\alpha \times R \cdot \vec{z} = -R \times \alpha \vec{z} = -R \times J_{\vec{\alpha}}(\vec{q}) \Delta\vec{q} = C_R \Delta\vec{q}$. By defining $C = [\zeta I_3; \eta C_R]$ with $\zeta = w_\zeta / (w_\zeta + w_\eta)$ and $\eta = w_\eta / (w_\zeta + w_\eta)$ we have the following optimization problem:

$$\Delta\vec{q} = argmin_{\Delta\vec{q}} \, \zeta \| \Delta\vec{q} \|^2 + \eta \| \alpha \times R \cdot \vec{z} \|^2 = argmin_{\Delta\vec{q}} \| C\Delta\vec{q} \|^2$$

Subject to:

$$\vec{x} = F \cdot \vec{p}_{tip}$$

$$D(\Delta\vec{x}) = \vec{\alpha} \times \vec{t} + \vec{\varepsilon} + \vec{x} - \vec{p}_{goal}$$

$$\vec{\alpha} = J_{\vec{\alpha}}(\vec{q}) \Delta\vec{q}; \; \vec{\varepsilon} = J_{\vec{\varepsilon}}(\vec{q}) \Delta\vec{q};$$

$$\| D(\Delta\vec{x}) \| \leq 3$$

$$\vec{q}_L - \vec{q} \leq \Delta\vec{q} \leq \vec{q}_U - \vec{q}$$

Constraint in tip position, joints limits and environment limitations

The next objective is to repeat the previous optimization problems, but with the added constraint of a virtual wall. The wall is defined by a point $\vec{w}_0$ and a normal vector $\vec{n}$ which are both defined relative to the space frame. The linearly approximated tip position $\vec{t}'$ after the transformation is defined as:

$$\vec{t}' = \vec{\alpha} \times \vec{t} + \vec{\varepsilon} + \vec{x}$$

Therefore, the distance vector between the origin of the wall and the transformed tool tip is:

$$P'_{tip} = \vec{t}' - \vec{w}_0$$

The distance from the tool tip to the closest point on the wall can be found with the projection of P'_{tip} on the normal vector $\vec{n}$. To avoid letting the tool tip cross the wall, this distance must be a positive value. This leads to the additional inequality constraint:

$$\vec{n} \, . \, P'_{tip} \geq 0$$

Results and performance comparisons

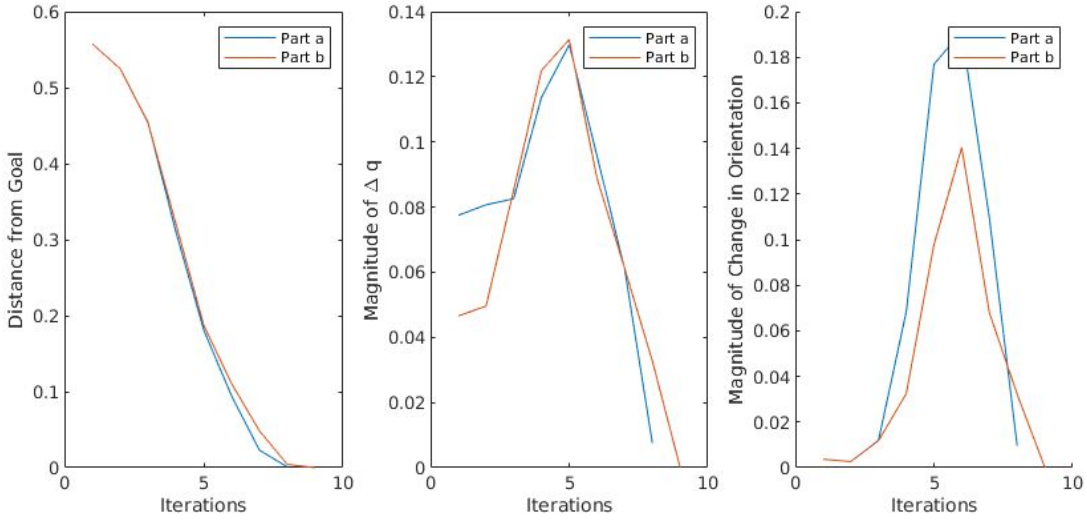
Scenario of a virtual wall constraint with a reachable target:

To solve our constrained optimization problems, we attempted approaches with both lsqlin() and fmincon(), which are built-in Matlab functions. When using the lsqlin() approach, the norm of the distance vector $\| D(\Delta\vec{x}) \|$ could not be written as a linear function. Since the syntax for lsqlin() requires constraints to be written as linear function, the $\| D(\Delta\vec{x}) \| \leq 3$ constraint could not be directly implemented. Instead, the distance vector $D(\Delta\vec{x})$ is driven towards 0, while an outer loop iterates the lsqlin() solver until $\| D(\Delta\vec{x}) \| \leq 3$ is satisfied.

Another method utilized the fmincon() function, which can accept nonlinear constraints. This allowed the $\| D(\Delta\vec{x}) \| \leq 3$ constraint to be directly implemented. However, the vector $D(\Delta\vec{x})$ is calculated using a linearly approximation and therefore is not completely accurate, especially if the tool tip is far away from the goal position. For this reason, it is still necessary to iterate fmincon() multiple times before $\| D(\Delta\vec{x}) \| \leq 3$ is actually satisfied. For this reason, the performance of the lsqlin() and fmincon() methods are essentially the same.

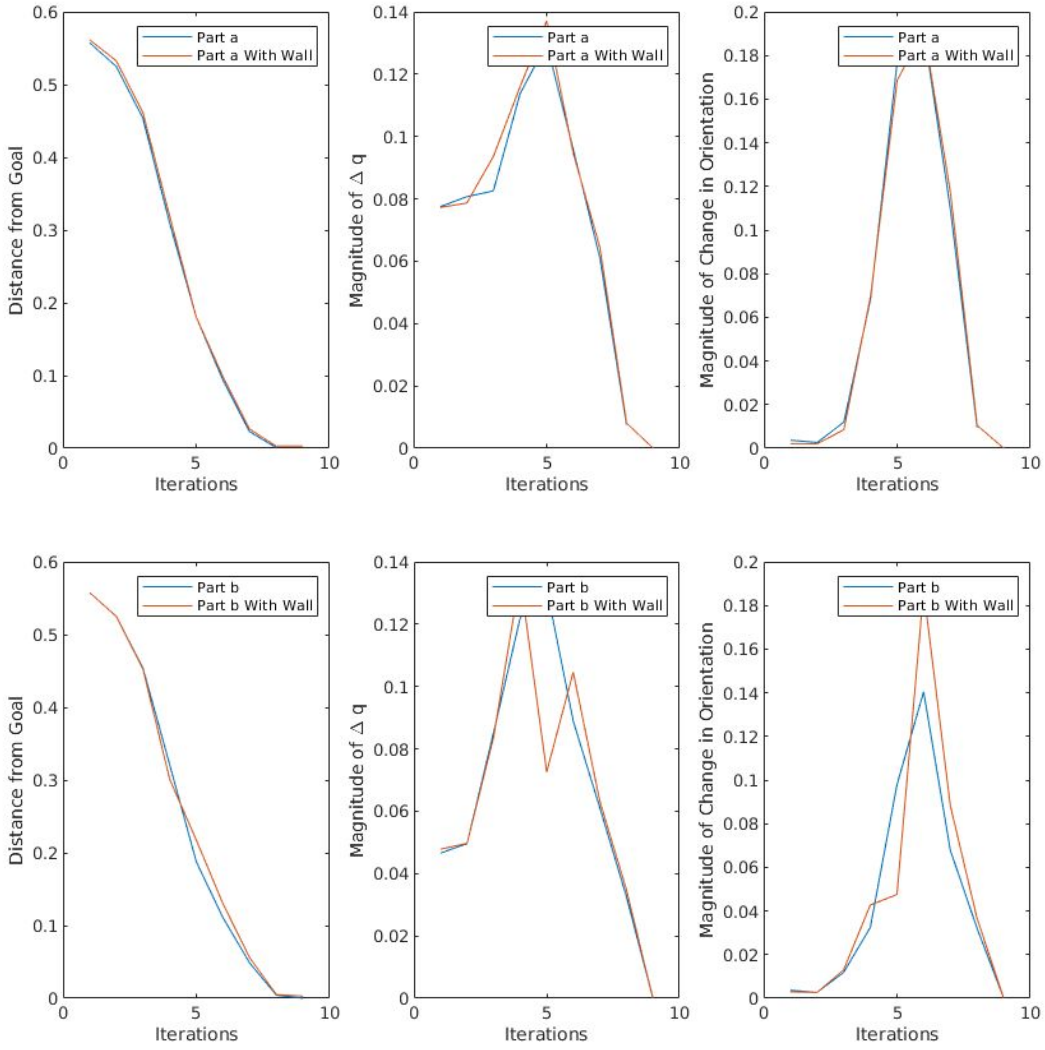
The output of both solvers is a set of changes in joint positions $\Delta\vec{q}$. To generate smoother trajectories, the maximum change in joint position is set as 0.2. In the event that a solver returns a $\Delta\vec{q}$ in which an element has a magnitude greater than 0.2, this $\Delta\vec{q}$ is scaled down such that the greatest magnitude of an element is 0.2.

The first comparison is between cases (a) and (b), where case (a) attempts to minimize the norm of $\Delta\vec{q}$ and case (b) attempts to minimize both the norm of $\Delta\vec{q}$ and the change in tool tip orientation. For case (b), the two objective functions each have a weight of 0.5. It can be seen from the left-most plot below that the goal is reached at approximately the same rate in both cases. From the center plot, it is seen that the magnitude of $\Delta\vec{q}$ is also very similar in both cases. The most noticeable difference is in the right-most plot, where case (b) maintains a smaller change in tool tip orientation, which is exactly the intention of the modified objective function.

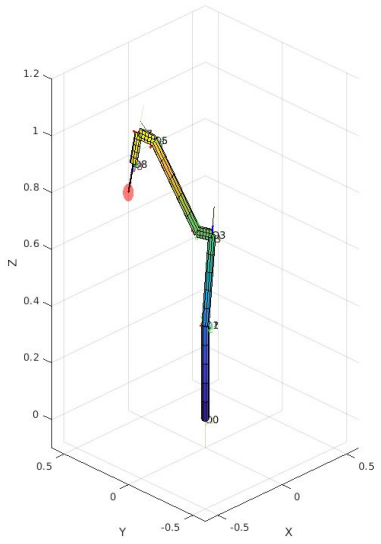


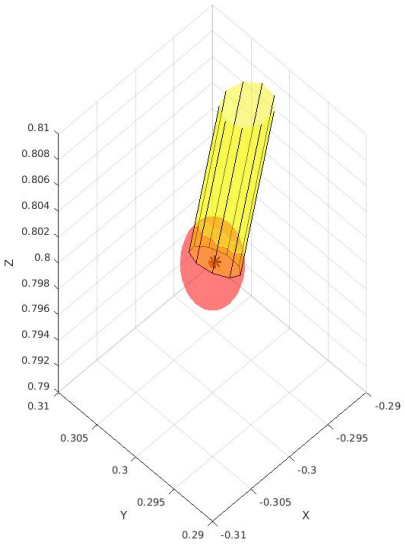
The plots below show the effect of adding a virtual wall constraint for both cases (a) and (b). To ensure that the wall has an effect, it was defined such that it passes through the 3mm sphere surrounding the

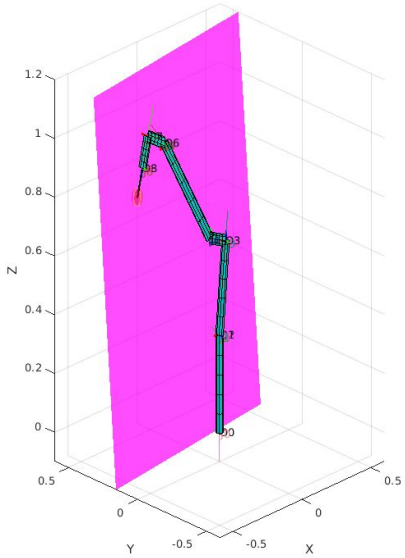
goal point. As the tool tip approaches the goal point, we notice very similar performance in all metrics. While difficult to see in the plots, the most noticeable difference is that the tool tip tends to finish comfortably within 3mm of the goal point when there is no virtual wall, but when the wall is active the tool tip often finishes along the edge of the 3mm sphere. This is likely due to the solver attempting to avoid the wall.

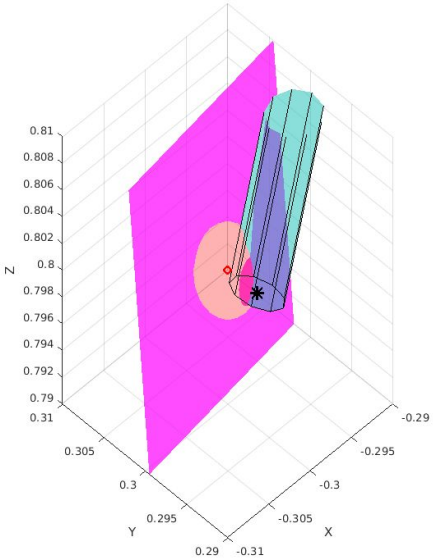


The images below show the ending position after solving the constrained optimization problem with and without the virtual wall fixture. The plots on the left show the full robot with an enlarged sphere representing the goal point. The plots on the right show a zoomed in view of the tool tip, with a 1-to-1 scale sphere representing the 3mm distance constraint. It is seen that in the case with a wall, the tool tip maintains a distance from the wall while also meeting the 3mm distance constraint.

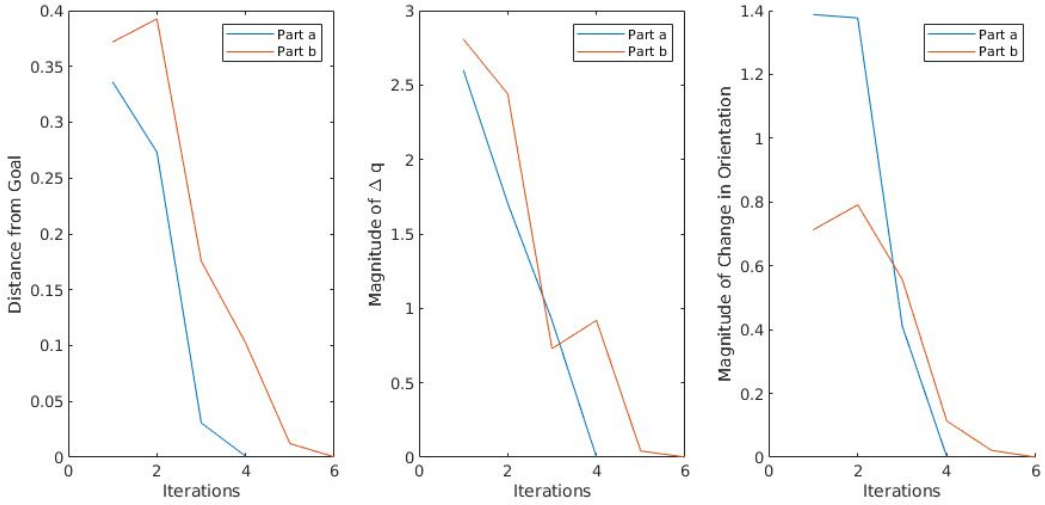






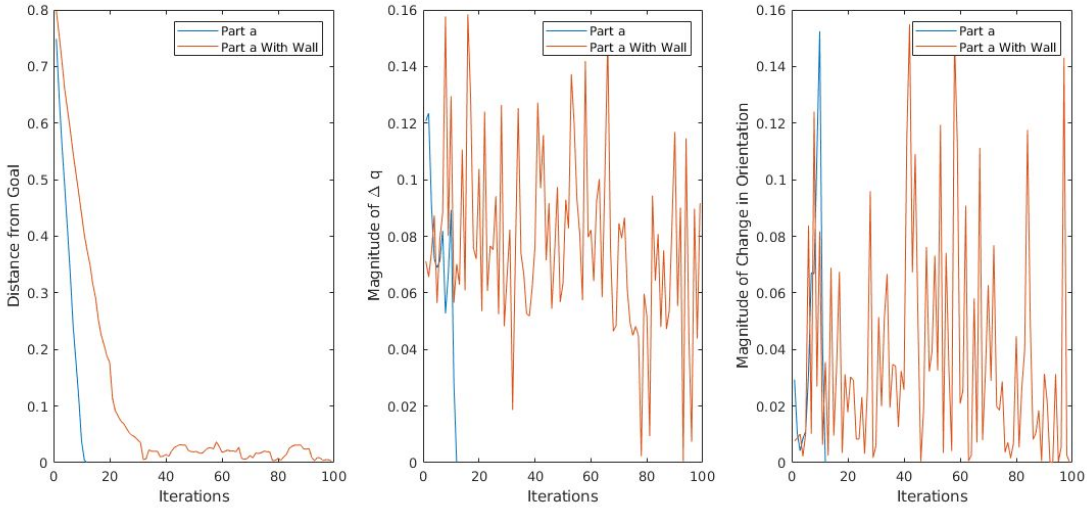


The plots below show the results of removing the limitation on the change in joint angles. This results in unrealistically high changes in joint angles per step, but it also shows interesting results from the algorithms. For example, when not constrained by joint angles, the algorithm for case (a) converges faster, as it does not need to account for minimizing the change in orientation.

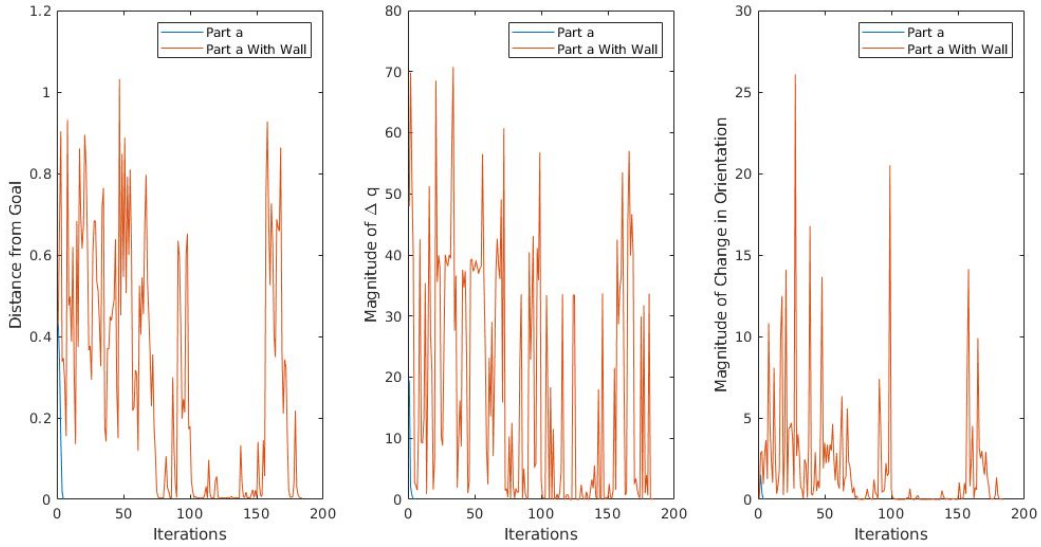


Scenario of a virtual wall constraint with an unreachable target:

Another interesting problem is defined such that the goal point is defined more than 3mm behind the wall. In other words, it is impossible to satisfy both constraints simultaneously. The results of this case are shown below, where the change in joint angles is limited to 0.2. It can be seen by the left-most plot that the algorithm approaches a solution, but struggles to actually reach within 3mm of the goal point.



The plots below show the results of the above scenario after removing the limits on joint motion. It can be seen that the robot does not stay near the goal point, and that there are large spikes in joint motion. This erratic behavior shows another benefit of setting a limit on joint motion.



Scenario of a moving target:

The final problem that we looked at is that of a non-stationary goal point. In this problem the position $\vec{p}_{goal}$ changes after each iteration of the solver. The solver then aims to move the tool tip to this new position. Once the initial gap between the tool tip and the goal point is closed, the tool tip is then able to continuously track the moving goal. This behavior is shown in the **Supplemental Videos** section, for the constraints associated with cases (a) and (b).

Additional Notes

An important note is that for each test, the initial joint positions are randomly generated within the Panda Robot's joint limits. On rare occasions, these initial joint positions can be set very close to the joint limits, which causes the solver to struggle to reach the goal point. However, the solver functions successfully in the majority of cases.

Another interesting behavior is that the robot links sometimes pass through the virtual wall, even if the tool tip does not. This could potentially be resolved by adding additional constraints for each joint of the robot to ensure they do not cross the wall.

Supplemental Videos

https://www.youtube.com/watch?v=urBAatKId_8: Reaching within 3mm of a goal point while minimizing the change in joint angles.

https://www.youtube.com/watch?v=GvQwMp2ea_E: Reaching within 3mm of a goal point while minimizing the change in joint angles and minimizing change in tool tip orientation.

https://www.youtube.com/watch?v=e1lyOZJXFwc: Reaching within 3mm of a goal point while minimizing the change in joint angles and avoiding a virtual wall. Note that even though parts of the robot pass through the wall, the tool tip stays in the allowed region.

https://www.youtube.com/watch?v=MtXCP4i3-Hg: Reaching within 3mm of a goal point while minimizing the change in joint angles, minimizing change in tool tip orientation, and avoiding a virtual wall. Note that even though parts of the robot pass through the wall, the tool tip stays in the allowed region.

https://www.youtube.com/watch?v=07oS8PO1QCc: Attempting to reach a point behind a virtual wall while the maximum change in joint angles is constrained to 0.2.

https://www.youtube.com/watch?v=SkBTA0sAvSM: Attempting to reach a point behind a virtual wall while the maximum change in joint angles is unconstrained. This causes erratic behavior.

https://www.youtube.com/watch?v=_UwOj1TKxx8: Tracking a moving goal point while minimizing the change in joint angles.

https://www.youtube.com/watch?v=SntApwe-TQI: Tracking a moving goal point while minimizing the change in joint angles and minimizing change in tool tip orientation.

Program listing

Scripts

- **Homework_4.m:** main file running code solving the homework's problems. Note that there is a pause between each animation, so any key must be pressed to resume the script.
- **Moving_Goal_Point.m:** script for running solvers with a moving goal point. Note that there is a pause between each animation, so any key must be pressed to resume the script.

Functions

- **LinearSolve.m:** Linear least-square solver for tip-goal distance and joints limits constraints.
- **LinearSolveOri.m:** Linear least-square solver for tip-goal distance, tip orientation, and joints limits constraints.
- **MinJointMotion.m:** fmincon() solver for minimizing joint motion.
- **MinJointMotionRotation.m:** fmincon() solver for minimizing joint motion and minimizing change in tool tip orientation.
- **MinJointMotionWall.m:** fmincon() solver for minimizing joint motion while avoiding a virtual wall.
- **MinJointMotionRotation.m:** fmincon() solver for minimizing joint motion and minimizing change in tool tip orientation while avoiding a virtual wall.
- **constraints_Distance.m:** defines the 3mm distance inequality constraint for use in the fmincon() solvers.
- **constraints_Distance_Wall.m:** defines the 3mm distance and virtual wall inequality constraints for use in the fmincon() solvers.
- **Generate_SS.m:** takes the rotation axis ω_s, joints frame origin q_S and vectors indicating positive translation t_s as inputs to generate a matrix Ss containing the screw axis S_i.
- **Plot_initial_conf.m:** takes q_s, Ss, as well as the set of parameters $params$ and Ts the concatenated zero-configuration space transformation matrix of each joint, as inputs to generate a plot showing the zero-configuration frames and screw axis.
- **Show_frame.m:** takes a specific transformation matrix T as an input and plots its associated frame
- **update_joint_frames.m:** plots the frame associated with each joint, taking the angles θ, screw axis Ss, T_s and $params$ as inputs, also calls for the cylinders plotting function.
- **update_joint_frames_tool.m:** plots the frame associated with each joint, taking the angles θ, screw axis Ss, T_s and $params$ as inputs, also calls for the cylinders plotting function. Includes visualization of tool tip.
- **update_joint_frames_tool_wall.m:** plots the frame associated with each joint, taking the angles θ, screw axis Ss, T_s and $params$ as inputs, also calls for the cylinders plotting function. Includes visualization of tool tip and virtual wall.
- **FK_space.m:** takes Ss, θ, M and $params$ as inputs to derive the spatial forward kinematics FK_s by using the Product of Exponentials.
- **FK_body.m:** takes Ss, θ, M and $params$ as inputs to derive the body forward kinematics
- **J_space.m:** takes Ss and θ as inputs to derive the space Jacobian J_s.

- **J_body.m:** takes Ss, M and θ as inputs to derive the body Jacobian J_b.
- **Cylinder2P.m:** takes the radius R, beginning and end points $r1$ and $r2$ of the desired cylinder, as well as N the number of points as inputs to plot a cylinder.
- **DH.m:** takes all denavit-hartenberg parameters as input, as well as a Boolean indicating convention the user wants to select, and outputs the resulting transformation matrix.
- **getAd.m:** takes the transformation matrix T as an input and returns the corresponding adjoint matrix.
- **getCrossform.m:** takes a vector v as an input and returns the matrix form of the cross product.
- **getTinv.m: :** takes the matrix T as input and returns its inverse.
- **getSfromSM.m:** takes the matrix form of S as input to return its vector form.
- **getE_St.m:** takes the screw axis S as well as the angle θ as inputs to derive ω and uses $\omega \; and \; \theta$ to find the rotation matrix form $e^{S\theta}$.
- **getJpsinv.m:** takes the Jacobian J as an input and returns it pseudo-inverse according to its size.