Computer Animation

Lab 6 - A2 - IK & Obstacle Avoidance

COMP 477

UNIVERSITÉ Concordia UNIVERSITY

Preliminaries and Readings

Slides based on [Buss et al. 2004] + [Maciejewski et al. 1985]

Assignment 2

Implement 2D Inverse Kinematics with Obstacle Avoidance

- Forward Kinematics:
    - Going from local positions and rotations to global positions and rotations
- Inverse Kinematics:
    - Going from global positions to local positions and rotations

References
[Buss et al. 2004] Samuel R Buss. Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. IEEE Journal of Robotics and Automation, 17(1-19):16, 2004.
[Maciejewski et al. 1985] Maciejewski, Anthony A., and Charles A. Klein. "Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments." The international journal of robotics research 4.3 (1985): 109-117

IK - Additional Info

As mentioned in [Buss et al. 2004] , damped least squares algorithm reduces oscillations caused by being close to singularities, but might not sufficient. One relatively easy to implement fix for this is to clamp the magnitude of our error vector (the vector from the end-effector to the mouse pointer).

$$e = \text{ClampMag}(\text{target\_position} - \text{end\_effector\_position}, D_{max})$$

$$\text{where ClamMag}(w, d) = \begin{cases} w & \text{if } ||w|| < d \\ d * w \,/\, ||w|| & \text{otherwise} \end{cases}$$

References

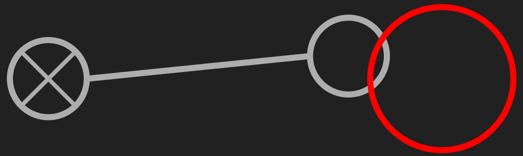
[Buss et al. 2004] Samuel R Buss. Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. IEEE Journal of Robotics and Automation, 17(1-19):16, 2004.

Obstacles

Can be separated into two parts:

- Obstacle Collision
- Obstacle Avoidance

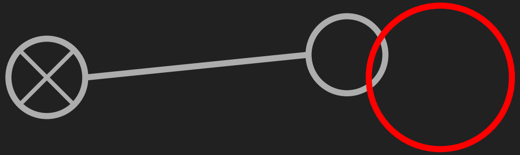
Let’s start with the simpler one: Obstacle Collision



Obstacle Collision

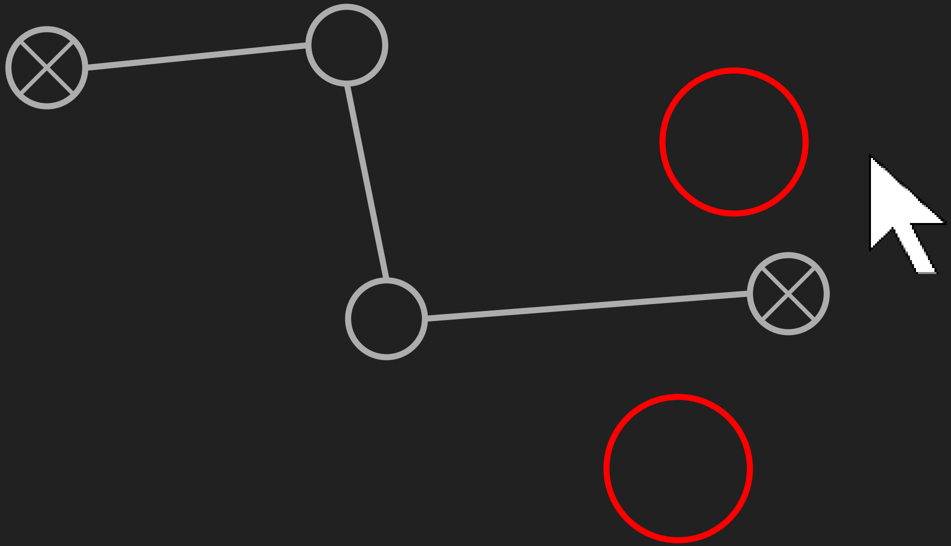
Obstacle collision is making sure that our arm does not clip through an obstacle. It can be viewed as a “last resort”. If all else has failed, make sure that our arm does not end up inside an obstacle.

After our IK, before updating the visual positions of the joints, check each joint and link in our hierarchy. If anything is clipping with an obstacle, cancel the update.



Obstacle Avoidance

Obstacle avoidance is making an arm avoids an obstacle by going around it. We will accomplish this using a similar Jacobian framework as we did for IK.



Computing Distances

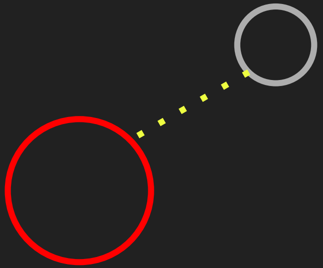
Note: we need to do this for each obstacle! So if we have 3 obstacles, we will have 3 closest points and 3 vectors!

First we need to know some values:

- Shortest vector from joint to obstacle
- Shortest vector from link to obstacle
- Closest point to obstacle

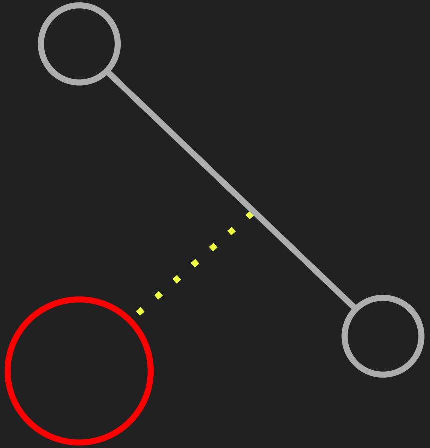
Joint to Obstacle

Treat them as points, then subtract radius obstacle , radius joint



Joint to Link

Treat obstacle as point, compute shortest vector from point to link, subtract radius of obstacle



Obstacle Jacobian

We want to build an obstacle jacobian J_O which will describe how a small change in a joint angle will change the positions of our closest points to our obstacles.

There are two cases for our Jacobian:

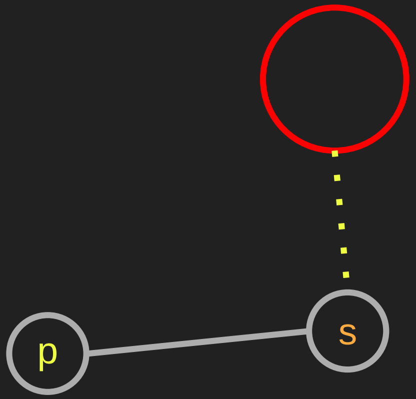
- Closest point to obstacle is on a joint
- Closest point to obstacle is on a link

Obstacle Jacobian

Closest point on joint

Simplest case, as it behaves exactly like if it was a simple end effector back in standard IK!

$$\frac{\partial \mathbf{s}_i}{\partial \theta_j} = \mathbf{v}_j \times (\mathbf{s}_i - \mathbf{p}_j)$$

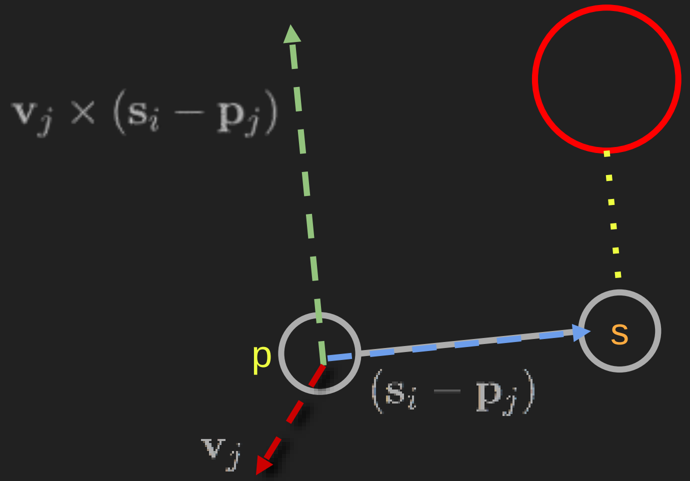


Obstacle Jacobian

Closest point on joint

Simplest case, as it behaves exactly like if it was a simple end effector back in standard IK!

$$\frac{\partial \mathbf{s}_i}{\partial \theta_j} = \mathbf{v}_j \times (\mathbf{s}_i - \mathbf{p}_j)$$

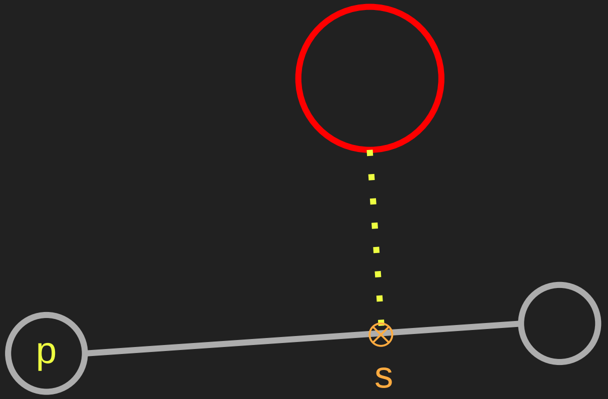

$\mathbf{v}_j \times (\mathbf{s}_i - \mathbf{p}_j)$
p
s
$(\mathbf{s}_i - \mathbf{p}_j)$
$\mathbf{v}_j$

Obstacle Jacobian

Closest point on link

Not too different. Treat wherever the closest point is on a link as an end effector, apply the same formula!

$$\frac{\partial \mathbf{s}_i}{\partial \theta_j} = \mathbf{v}_j \times (\mathbf{s}_i - \mathbf{p}_j)$$

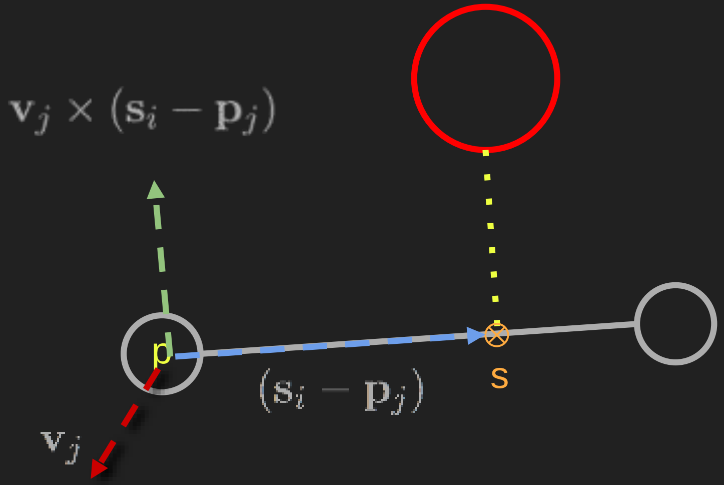


Obstacle Jacobian

Closest point on link

Not too different. Treat wherever the closest point is on a link as an end effector, apply the same formula!

$$\frac{\partial \mathbf{s}_i}{\partial \theta_j} = \mathbf{v}_j \times (\mathbf{s}_i - \mathbf{p}_j)$$



$\mathbf{v}_j \times (\mathbf{s}_i - \mathbf{p}_j)$
p
$(\mathbf{s}_i - \mathbf{p}_j)$
s
$\mathbf{v}_j$

Obstacle Jacobian

Sidenote:

The actual “closest point” doesn’t quite behave like this. At every iteration, the closest point will likely change location and can even move to another link/joint. However, since our approach is iterative, this won’t be an issue as at each given timestep we will recompute the new closest point and a new jacobian.

Obstacle Jacobian

If a joint does not affect a closest point, its corresponding entry in the Jacobian will be 0!

The size of our jacobian will be:

num_rows = 2*num_obstacles

num_columns = num_rotational_joints

Obstacle Avoidance

Now that we have our obstacle Jacobian J_O , how do we integrate it into our inverse kinematics?

Our main formula is equation 15 from [Maciejewski et al. 1985]

$$\begin{aligned}\dot{\theta} = {} & J_e^{+}\dot{\mathbf{x}}_e \\ & + \alpha_\eta[J_0(I - J_e^{+}J_e)]^{+}(\alpha_0\dot{\mathbf{x}}_0 - J_0J_e^{+}\dot{\mathbf{x}}_e),\end{aligned}$$

We will make a small change to the formula: instead of using the pseudoinverse (marked $^{+}$), we will use the the **DLS** (damped least squares) method

References
[Maciejewski et al. 1985] Maciejewski, Anthony A., and Charles A. Klein. "Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments." The international journal of robotics research 4.3 (1985): 109-117

Obstacle Avoidance

Rewriting the equation to match our notation and apply the minor change mentioned, we get:

$$\Delta\theta = \underbrace{\mathrm{DLS}(\mathbf{J}_e)\mathbf{e}_e}_{\substack{\text{End-effector}\\ \text{tracking IK}\\ \text{implemented}\\ \text{in lab 5}}} + \underbrace{\alpha_\eta \mathrm{DLS}(\mathbf{J}_o(I - \mathrm{DLS}(\mathbf{J}_e)\mathbf{J}_e))}_{\substack{\text{Homogeneous term} \rightarrow \text{controls the}\\ \text{blending between obstacle avoidance}\\ \text{and end-effector tracking}}}\underbrace{(\alpha_o \mathbf{e}_o - \mathbf{J}_o \mathrm{DLS}(\mathbf{J}_e)\mathbf{e}_e)}_{\substack{\text{Obstacle term} \rightarrow \text{controls}\\ \text{tracking of the end effector}}}$$

Obstacle Avoidance

Rewriting the equation to match our notation and apply the minor change mentioned, we get:

$$\Delta\theta = \mathrm{DLS}(\mathrm{J}_e)\mathrm{e}_e + \alpha_\eta \mathrm{DLS}(\mathrm{J}_o(I - \mathrm{DLS}(\mathrm{J}_e)\mathrm{J}_e))(\alpha_o \mathrm{e}_o - \mathrm{J}_o \mathrm{DLS}(\mathrm{J}_e)\mathrm{e}_e)$$

DLS $\rightarrow$ Damped least squares inverse

$\mathbf{J}_e \rightarrow$ end-effector jacobian

$\mathbf{e}_e \rightarrow$ end-effector error

$\boldsymbol{\alpha}_\eta \rightarrow$ homogeneous term gain

$\mathbf{J}_o \rightarrow$ obstacle jacobian

$\mathbf{e}_o \rightarrow$ obstacle error

$\boldsymbol{\alpha}_o \rightarrow$ obstacle avoidance gain

$I \rightarrow$ identity matrix

Obstacle Avoidance

$$\Delta\theta = \mathrm{DLS}(\mathrm{J}_e)\mathrm{e}_e + \alpha_\eta \mathrm{DLS}(\mathrm{J}_o(I - \mathrm{DLS}(\mathrm{J}_e)\mathrm{J}_e))(\alpha_o \mathrm{e}_o - \mathrm{J}_o \mathrm{DLS}(\mathrm{J}_e)\mathrm{e}_e)$$

$$\mathrm{DLS}(\mathrm{J}) = \mathrm{J}^\top(\mathrm{J}\mathrm{J}^\top + \lambda^2 I)^{-1}$$

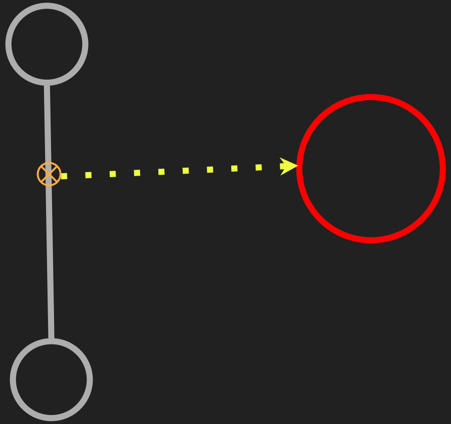
Note that each call to to **DLS** necessitates its own value for the damping factor λ .

You will need to experiment with the values to find a setup that works. Values will typically range between 5-100.

Obstacle Error

$$\Delta\theta = \mathrm{DLS}(\mathbf{J}_e)\mathbf{e}_e + \alpha_\eta \mathrm{DLS}(\mathbf{J}_o(I - \mathrm{DLS}(\mathbf{J}_e)\mathbf{J}_e))(\alpha_o \mathbf{e}_o - \mathbf{J}_o \mathrm{DLS}(\mathbf{J}_e)\mathbf{e}_e)$$

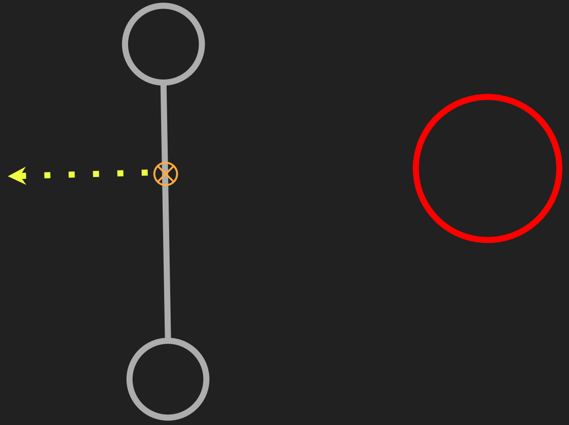
$\mathbf{e}_o$ will contain our obstacle error. Using the vector we computed prior, we will construct the error vector to be a normalized vector starting from the closest point, pointing away from the obstacle.



Obstacle Error

$$\Delta\theta = \mathrm{DLS}(\mathrm{J}_e)\mathrm{e}_e + \alpha_\eta \mathrm{DLS}(\mathrm{J}_o(I - \mathrm{DLS}(\mathrm{J}_e)\mathrm{J}_e))(\alpha_o \mathrm{e}_o - \mathrm{J}_o \mathrm{DLS}(\mathrm{J}_e)\mathrm{e}_e)$$

e$_o$ will contain our obstacle error. Using the vector we computed prior, we will construct the error vector to be a normalized vector starting from the closest point, pointing away from the obstacle.



Obstacle Avoidance

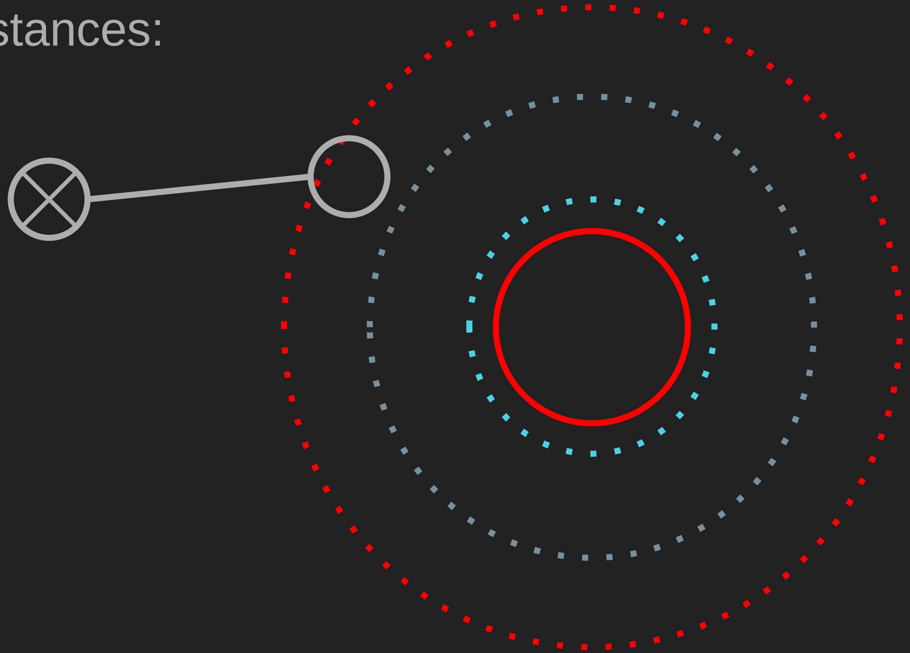
α_η and α_0 are scalar variables that control the importance of obstacle avoidance.

These scalars will depend on three distances:

d_{ta} $\rightarrow$ **Task abort distance**

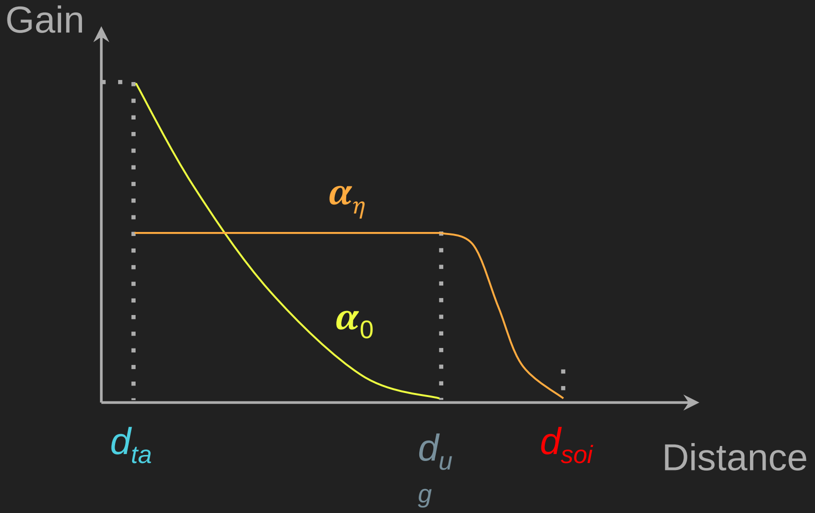
d_{soi} $\rightarrow$ **Sphere of influence distance**

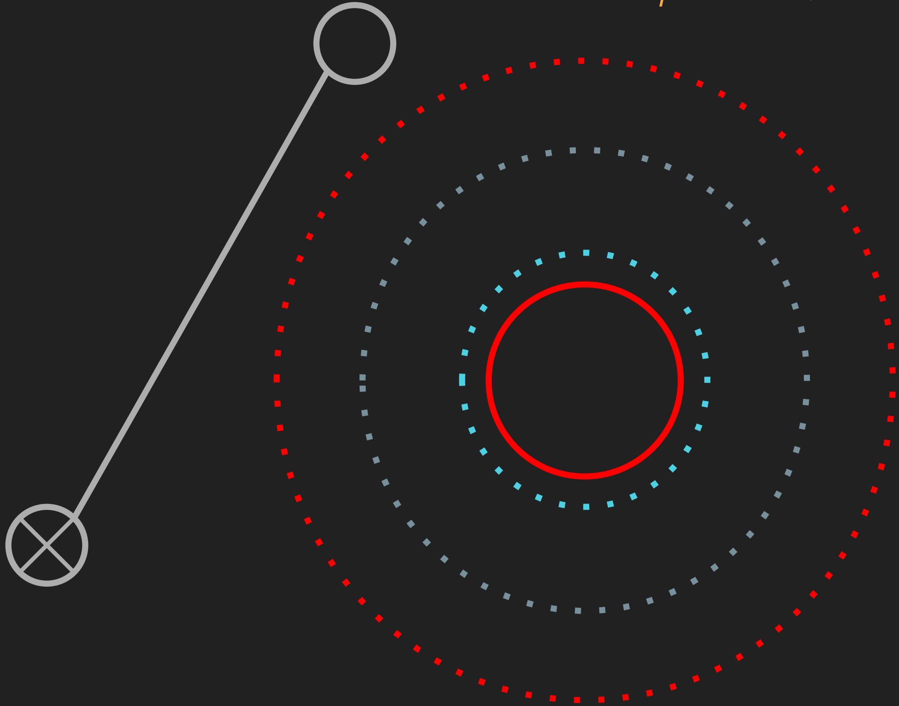
d_{ug} $\rightarrow$ **Unity gain distance**



Obstacle Avoidance

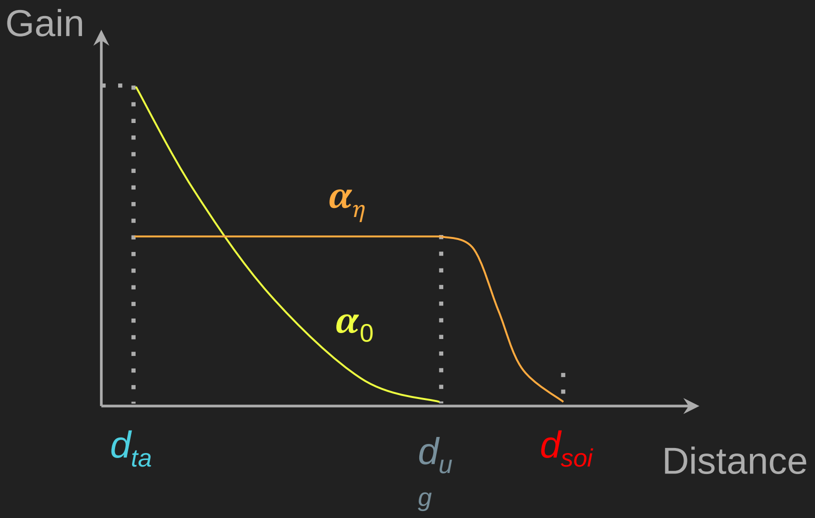
Passed d_{soi} the obstacle should have no effect on our manipulator. α_η and α_0 are both 0.

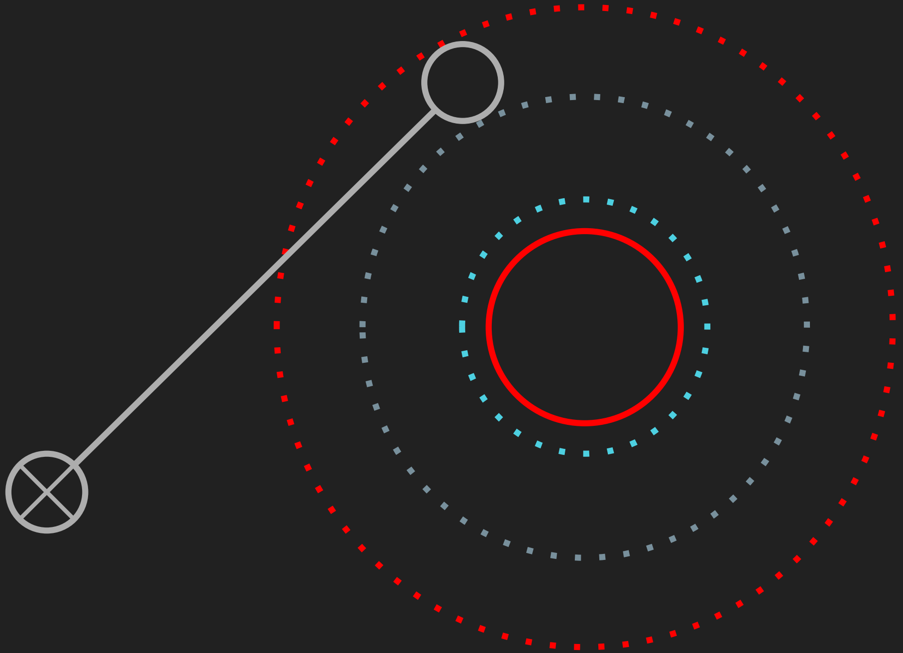




Obstacle Avoidance

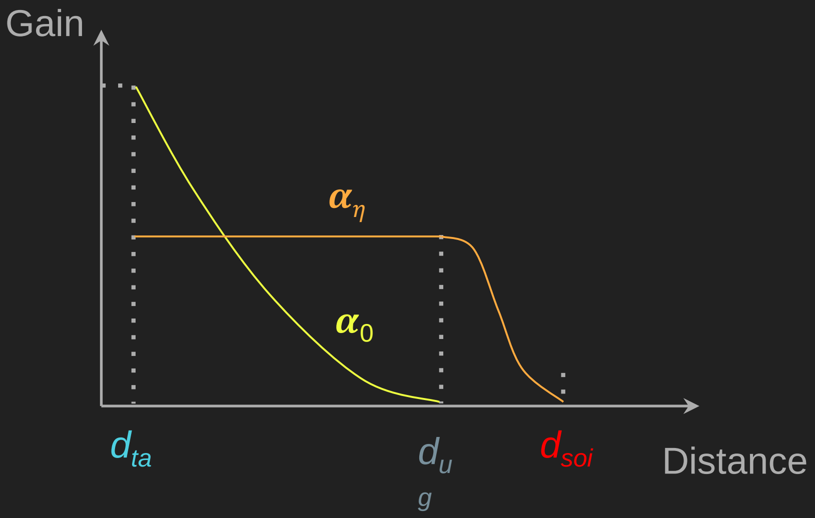
Between d_{soi} and d_{ug} , we transition into including the obstacle's influence in our solution. α_η should smoothly transition from 0 to 1.
α_0 remains 0.

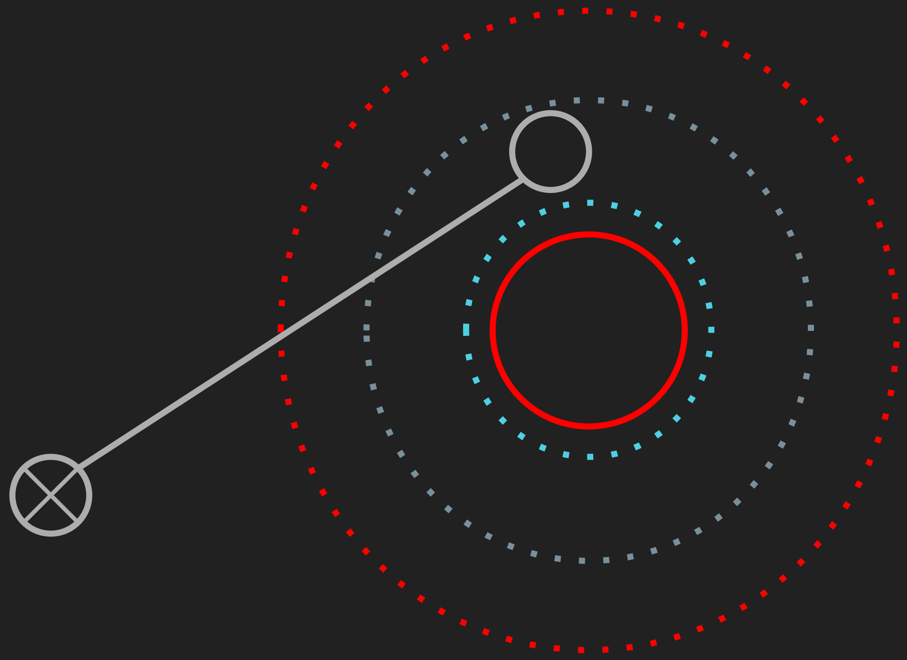




Obstacle Avoidance

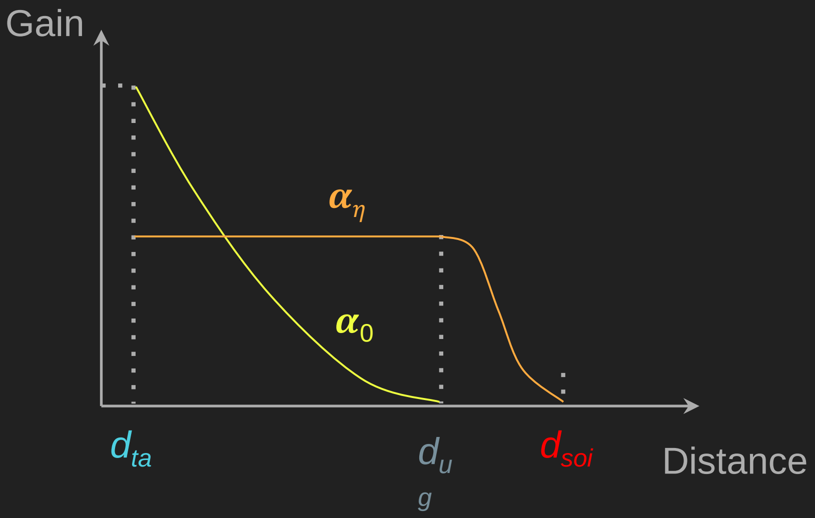
Between d_{ug} and d_{ta} , the influence of the obstacle (α_0) scales inversely proportional to the distance. α_η remains 1.

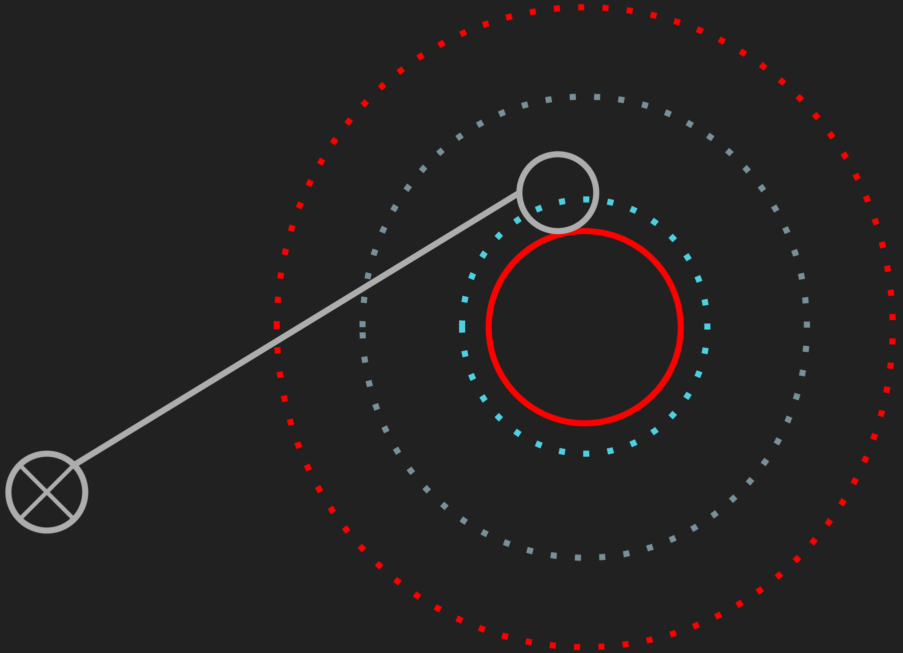




Obstacle Avoidance

Passed d_{ta} collision is considered imminent. Terminate the update. Should be a small amount larger than the radius of the obstacle.





Obstacle Avoidance

If we have multiple obstacles (and thus multiple closest points), take the shortest distance to compute our α's