centre adscrit a:

Animation Foundations

13. Constraints for Inverse Kinematics

Lessons learnt from Direct Kinemat

Reminder

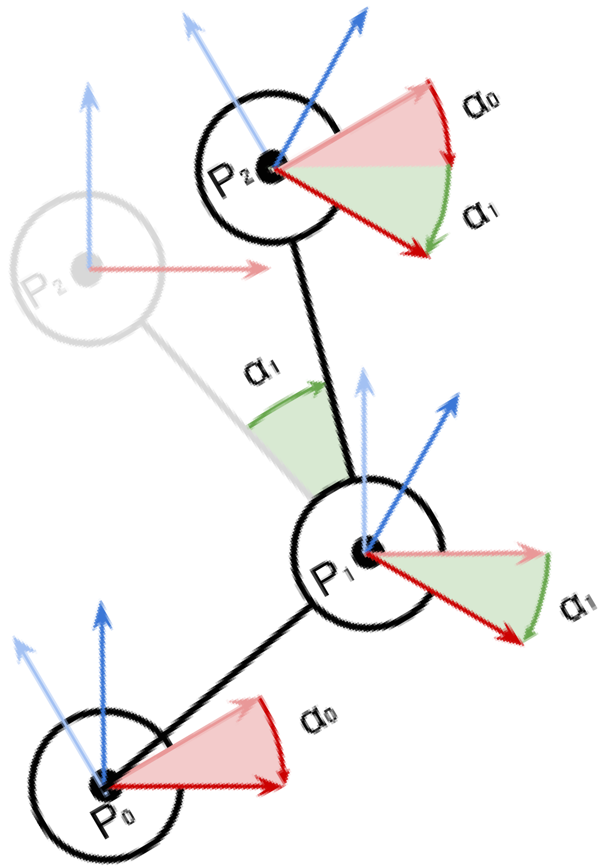
When introducing Direct_kinematics, we did the following:

1. Calculate direct kinematics
2. Start to add constraints (angles)

Check:

08.Direct_Kinematics.pdf

08.Direct_Kinematics_exercises-unfinished



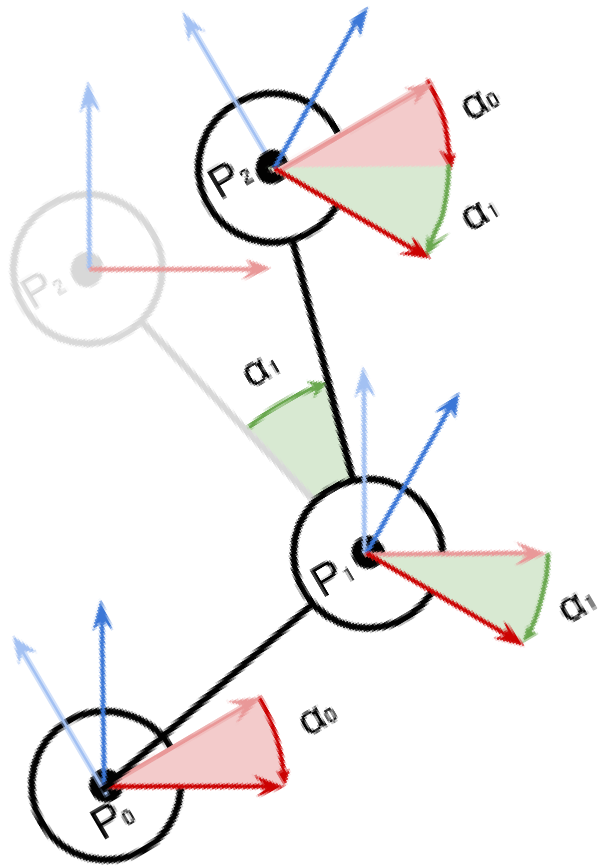
Lessons learnt from Inverse Kinematics

Now, we have:

1. An RL method that uses the function that calculates direct_kinematics: the goal of gradient descent is to minimize the direct_kinematics function
2. Many articulated entities have different kinds of constraints,

But:

- We have not explored enough how to introduce constraints
- We do not know how to combine constraints with



Constraints for Inverse Kinematics

Motivation: when using inverse kinematics methods we may need to:

- Constrain the angles of rotation of a joint (remember the robot joints)
- Constrain the plane of rotation of a joint (remember the robot joints)
- Introduce more general constraints (remember the example concerned with keeping the mass center within certain boundaries)

Constraints in IK. Exercise 1

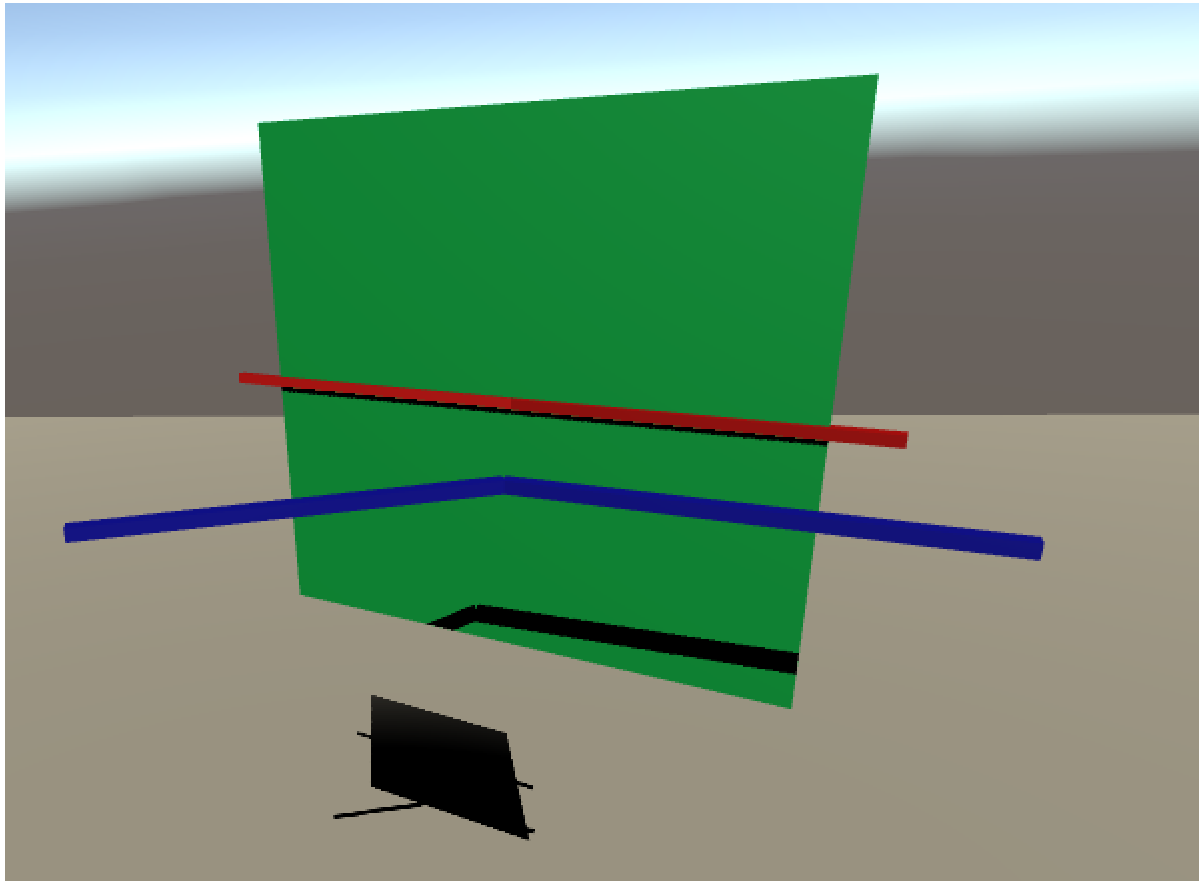
Import package Constraints4IK.unitypackage

Load scene mirrorMovement

a) Complete script MirrorMovement.cs in order the angle of rotation is between the minimum and maximum angle variables

b) Before the previous calculations, cancel the twist of the affected bone

c) Verify whether canceling the twist changes the behaviour, and why (it will depend on how you implemented a. and b.)



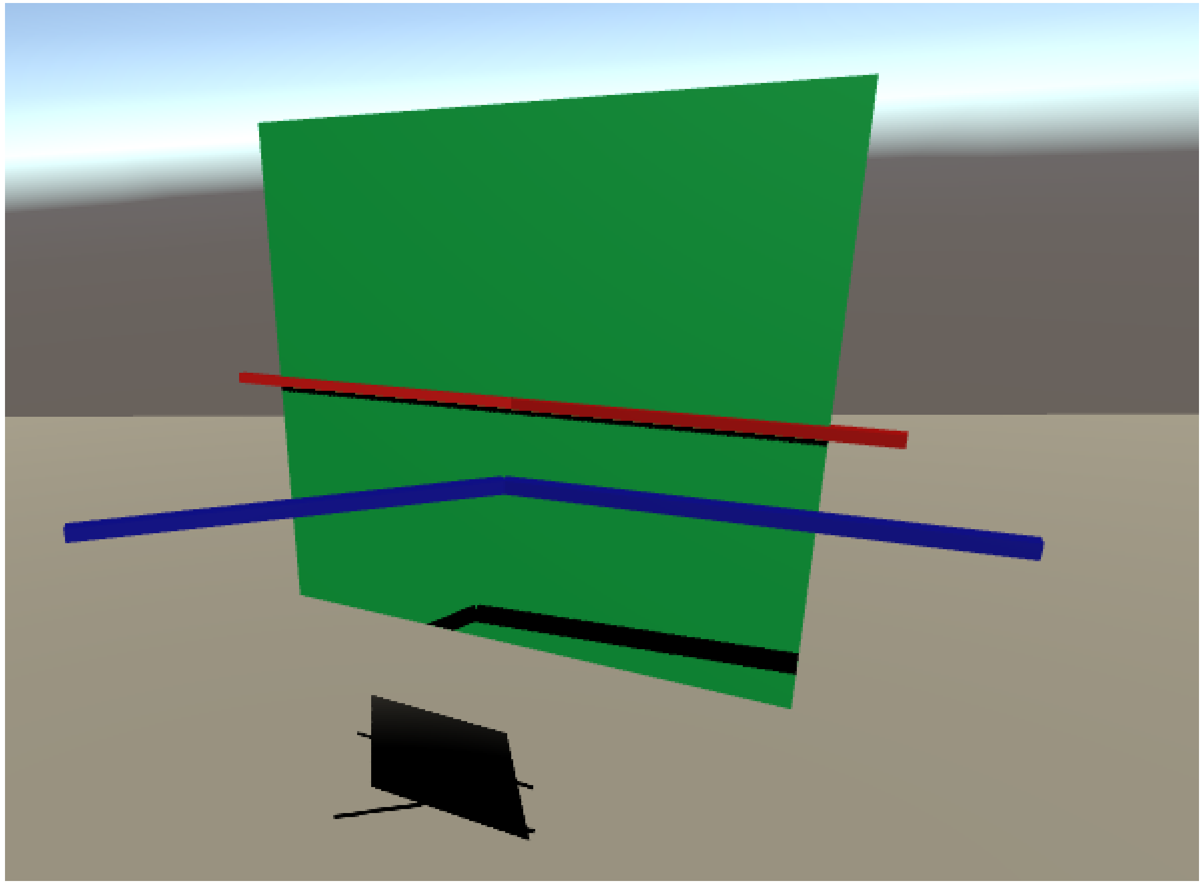
Note: We addressed a simpler version of this when introducing Direct Kinematics

Constraints in IK. Exercise 2

In scene mirrorMovement

a) Complete script MirrorMovement.cs in introducing a plane constraint, i.e., that the red joints are always on the surface of the plane.

b) Verify that you can combine this constraint with the angle constraint.

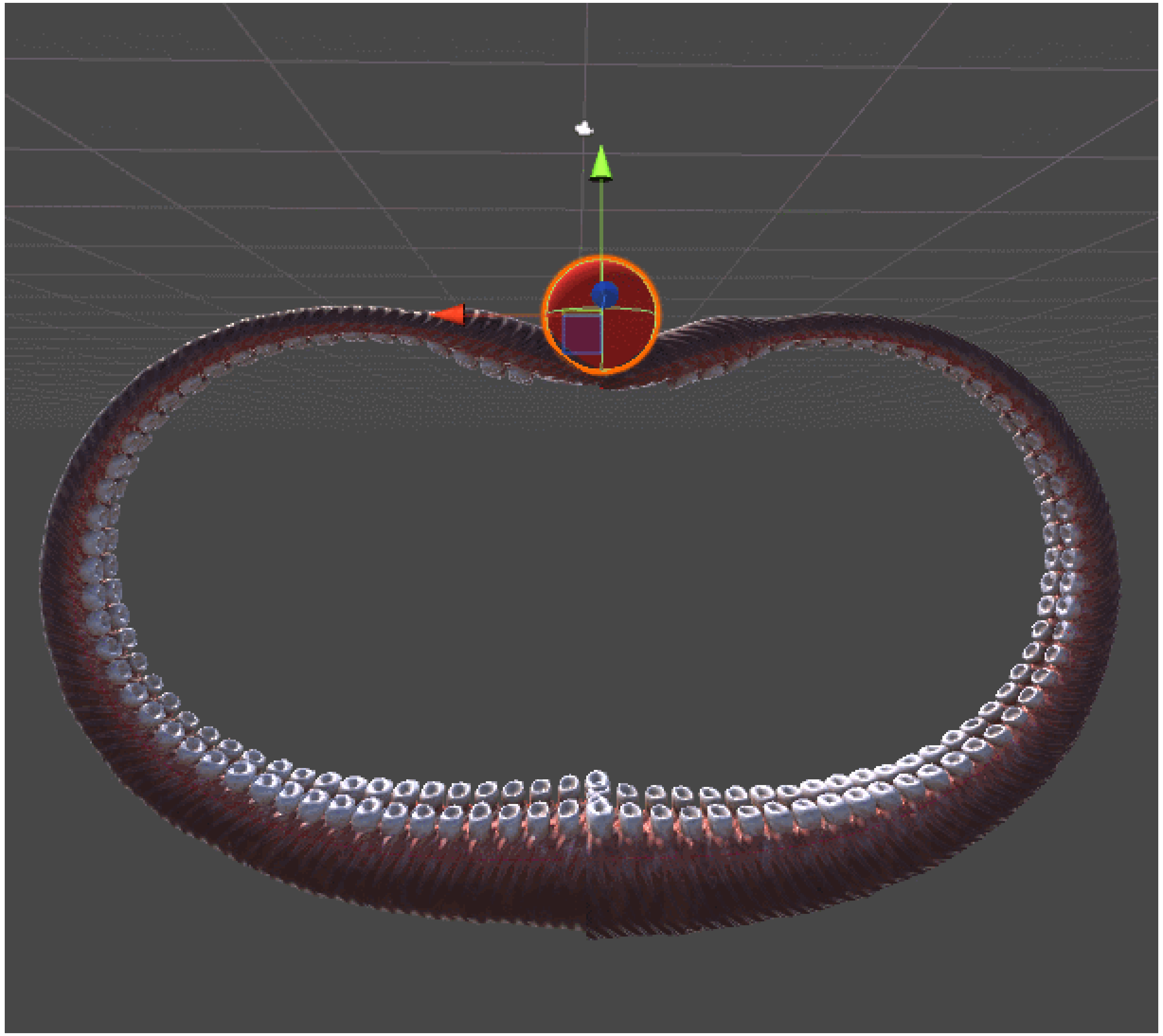


Constraints in IK. Towards global constraints

We can also explore more abstract constraints. For this, please notice:

1. When introducing the Jacobian, we discussed more global constraints could be introduced in the calculation of the Derivative.
2. When we programmed the Gradient descent method, we noticed that the distance was treated as an Error function (see Start() in InverseKinematics.cs)
3. In the right, a gif of a more abstract constraint. This is implemented with an error term focused on minimizing the angle between the object and the end-effector, as well as the torsion

Note: the gradient descent project was adapted from a tutorial by Alan Zucconi. See explanation on implementation here: https://www.alanzucconi.com/2017/04/12/tentacles

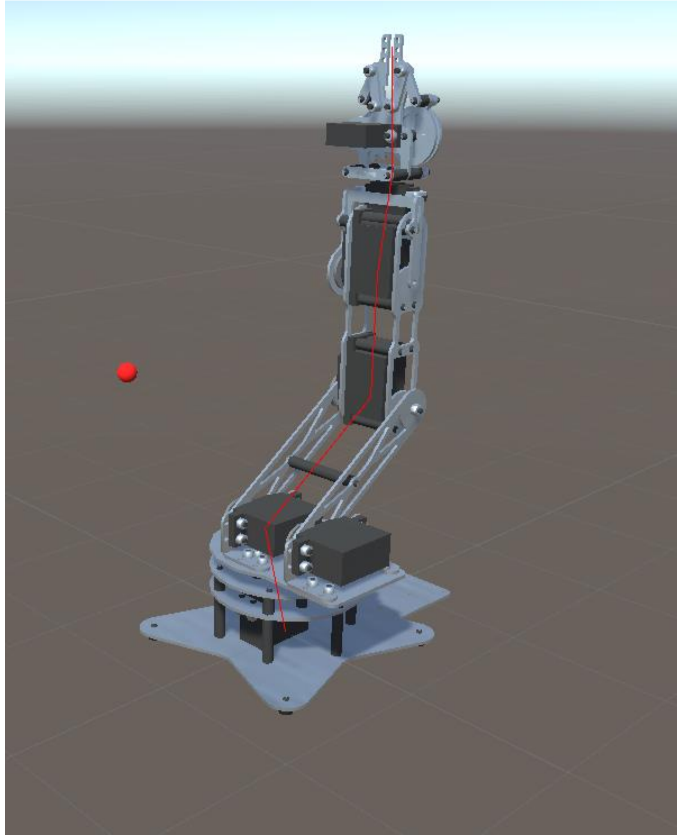


Constraints in IK. Exercise 3.

Open again the Gradient Descent package. In the InverseKinematics script, add a new function that you will use as an error function. Using it should:

a) Minimize distance to target, (just as function DistanceFromTarget)

b) Minimize the angle differences between joint1, joint2, and joint3 (torsion spread along the 3 joints)

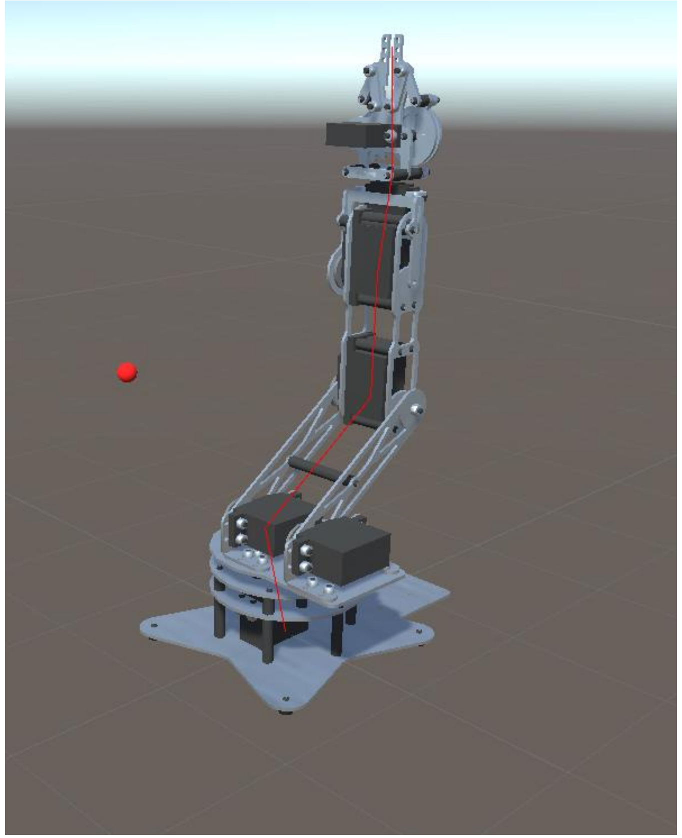


Constraints in IK. Exercise 4.

In exercise 3, add a slider that allows controling the importance of each of these factors. Then:

a) Write a new error function where: Each of the two previous factors a. and b. gives a measure that is between 0 and 1

b) Do a weighted sum of the factors, and check if the dynamics behaviour of the robot changes.



Constraints in IK. Exercise 5.

a. Calculate the center of mass (CoF) of the robot, assuming the different joints weigh the same, and that the weight of each joint is equally distributed

b. Write a new error function that minimizes the distance to target, while keeping the CoF as close as possible from Joint0