centre adscrit a:
UNIVERSITAT DE BARCELONA

Animation Foundations

09. Introduction to Inverse Kinematics

Dr Joan Llobera – joanllobera@enti.cat

Rotate a vector in 3D with Quaternions

REMINDER

Rotating a vector **p** by a quaternion **q** is:

$$p' = qpq^*$$

However,in Unity, given

Vector3 p1;
Quaternion q;
Vector3 p2;

We can write this operation as:

p2= q*p1;

(vector3) = (Quaternion) * (Vector3)

This does the following:

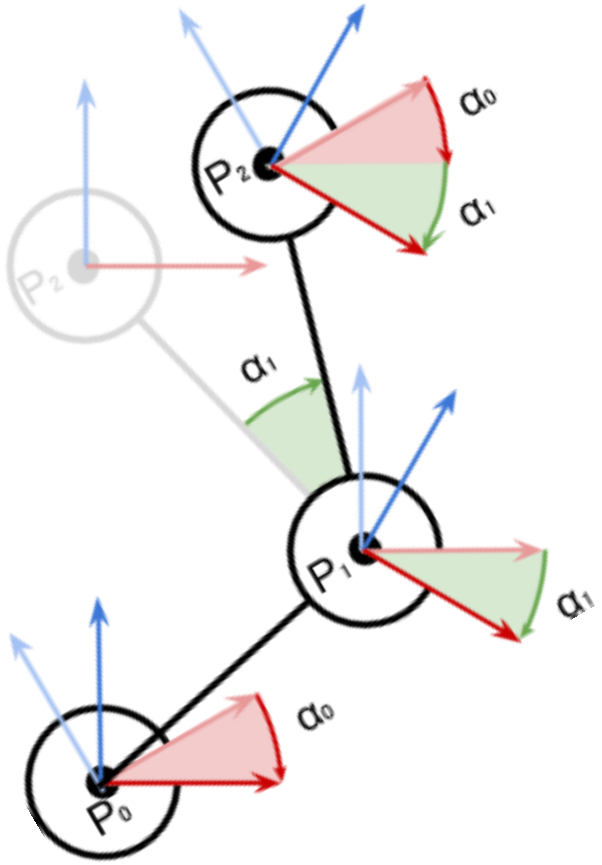
```
public static Vector3 operator *(Quaternion quat, Vector3 vec){
    float num = quat.x * 2f;      float num2 = quat.y * 2f;      float num3 = quat.z * 2f;
 float num4 = quat.x * num;      float num5 = quat.y * num2;      float num6 = quat.z * num3;
float num7 = quat.x * num2;      float num8 = quat.x * num3;      float num9 = quat.y * num3;
float num10 = quat.w * num;      float num11 = quat.w * num2;      float num12 = quat.w *num3;
    Vector3 result;
    result.x = (1f - (num5 + num6)) * vec.x + (num7 - num12) * vec.y + (num8 + num11) * vec.z;
    result.y = (num7 + num12) * vec.x + (1f - (num4 + num6)) * vec.y + (num9 - num10) * vec.z;
    result.z = (num8 - num11) * vec.x + (num9 + num10) * vec.y + (1f - (num4 + num5)) * vec.z;
    return result;
 }
```

https://answers.unity.com/questions/372371/multiply-quaternion-by-vector3-how-is-done.html

Inverse Kinematics. Introduction

IK:

How to find the joints that match a position?



Forward Kinematics. Reminder

To determine the position of hand:

$$r_i = \sum_{k=0}^{i-1} \alpha_k$$

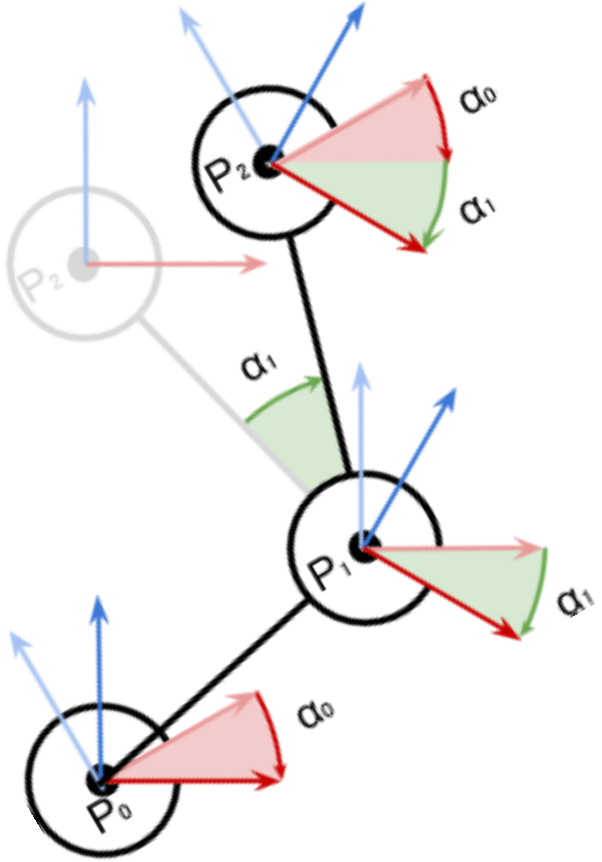
$$P_i = P_{i-1} + rotate\,(D_i, P_{i-1}, \sum_{k=0}^{i-1} \alpha_k)$$

P_i position of joint i

D_i distance of joint i

α_i local rotation of joint i

r_i global rotation of joint i

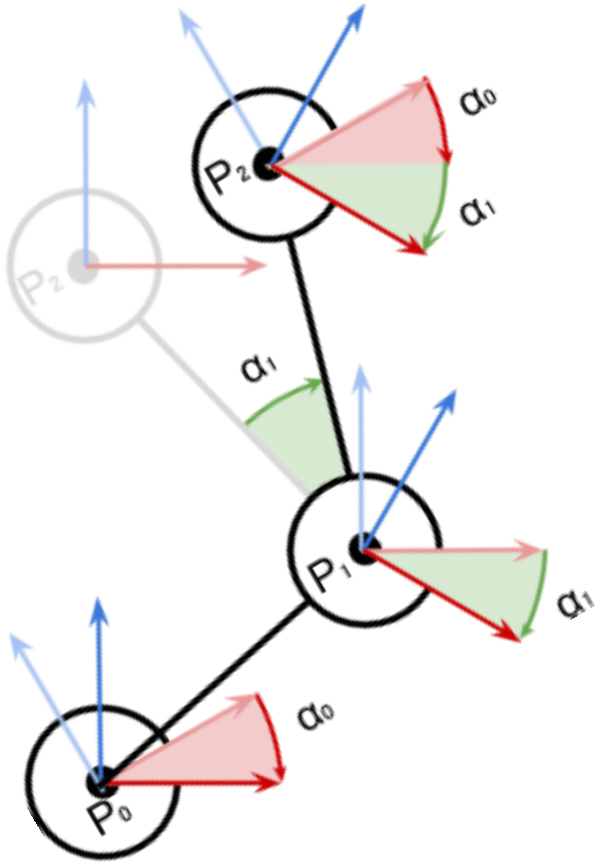


Direct kinematics and distances

Direct kinematics is a function that tells us the position of each limb from a set of angles.

With a direct kinematics function we can estimate the distance to a target.

Public Vector3 DistanceFromTarget(Vector3 target, float[] angles)



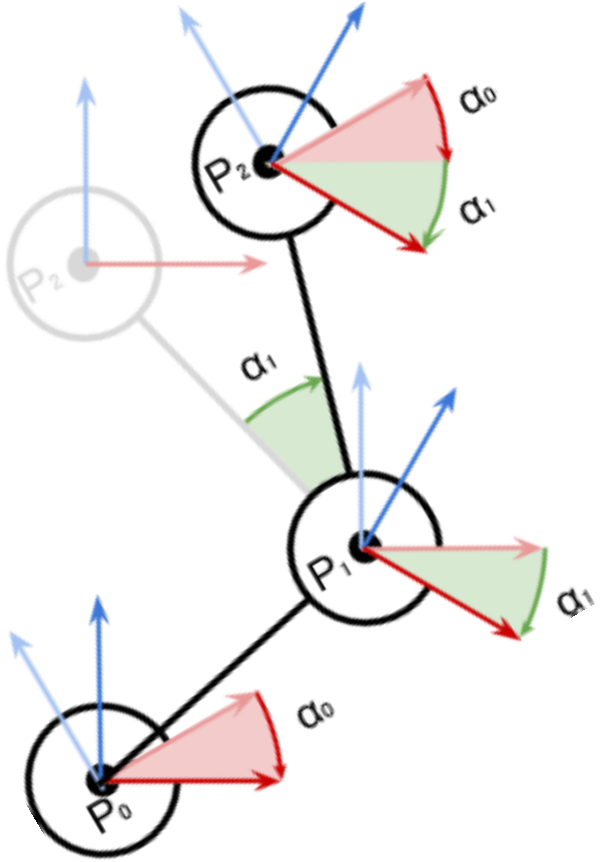
Direct Kinematics and distances. Implement

Public Vector3 DistanceFromTarget(Vector3 target, float[] angles)

$$P_i = P_{i-1} + rotate\ (D_i, P_{i-1}, \sum_{k=0}^{i-1} \alpha_k)$$

$$\text{DistTarg} = \text{T} - P_i$$

Exercise: download the .unitypackage and complete the function ForwardKinematics in the class InverseKinematics.cs



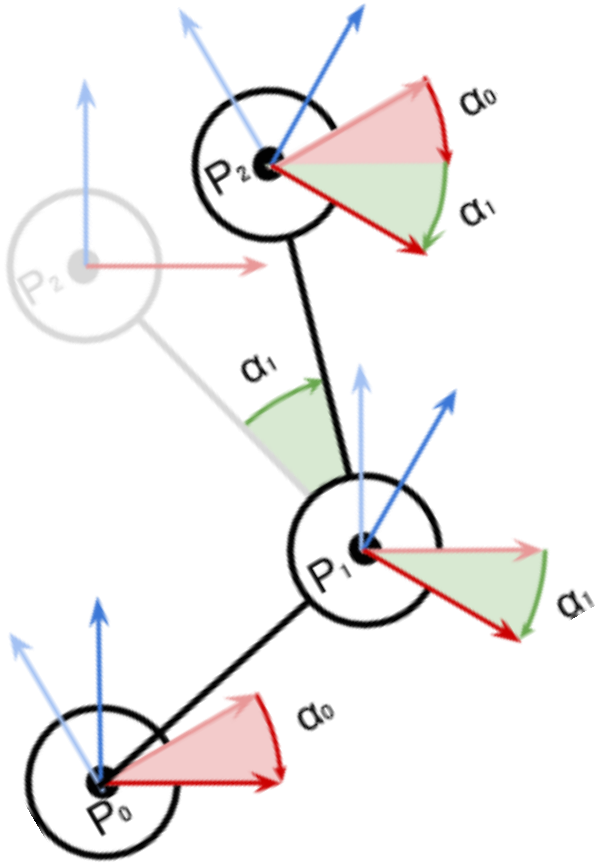
Inverse kinematics. The intro

Idea!

We can define an optimization function to minimize a distance depending on a certain number of angles

Min function(distance(angles) ,angles) = ?

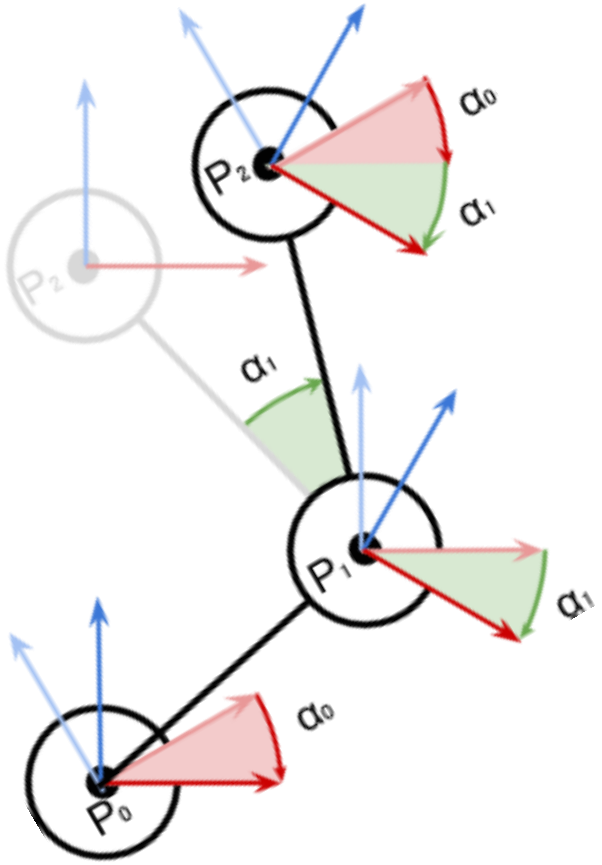
But how?



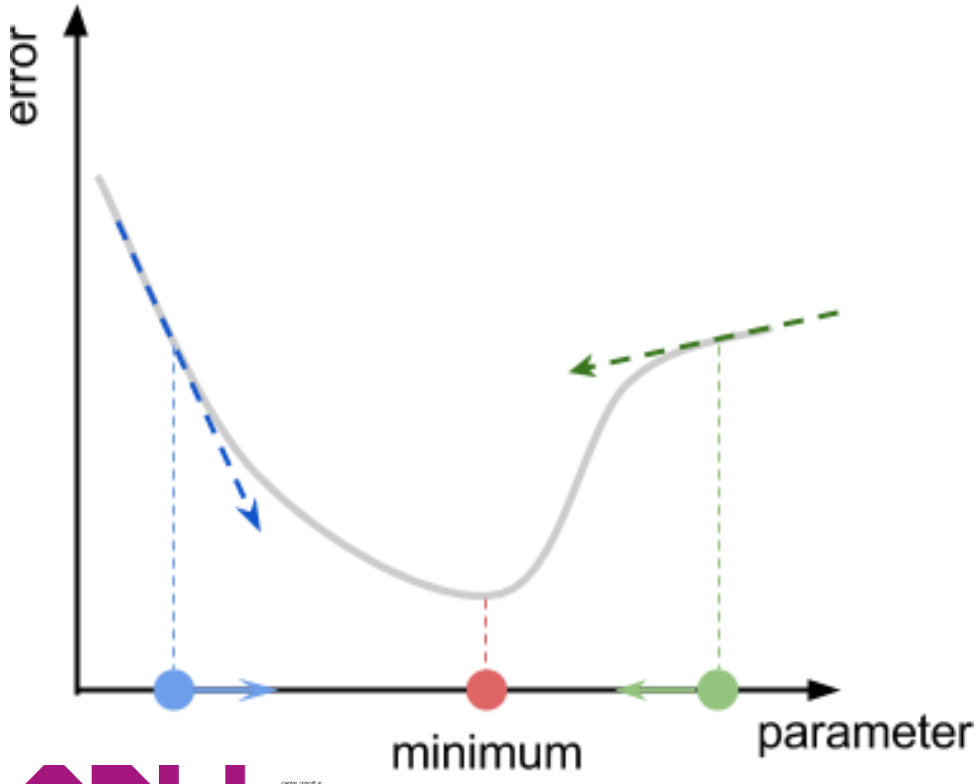
Inverse kinematics. The methods

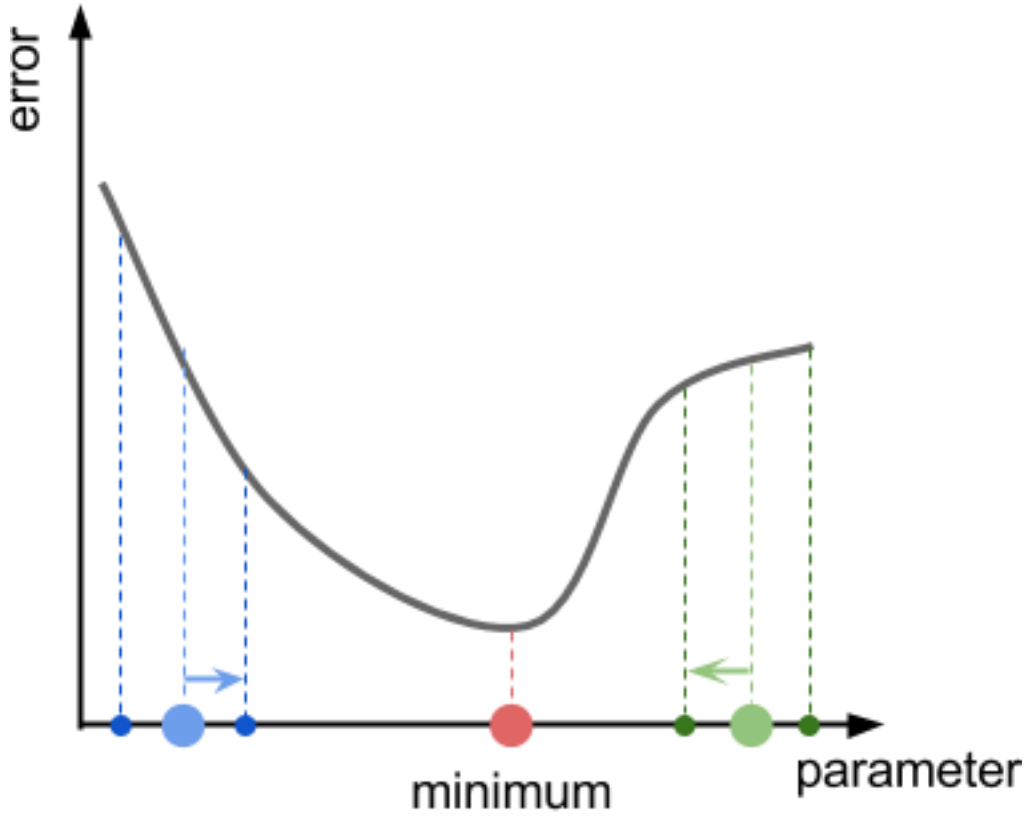
3 methods:

- Gradient Descent (GD)
- Cyclic Coordinate Descent (CCD)
- Forward and Backward Recursive Inverse Kinematic (FABRIK)

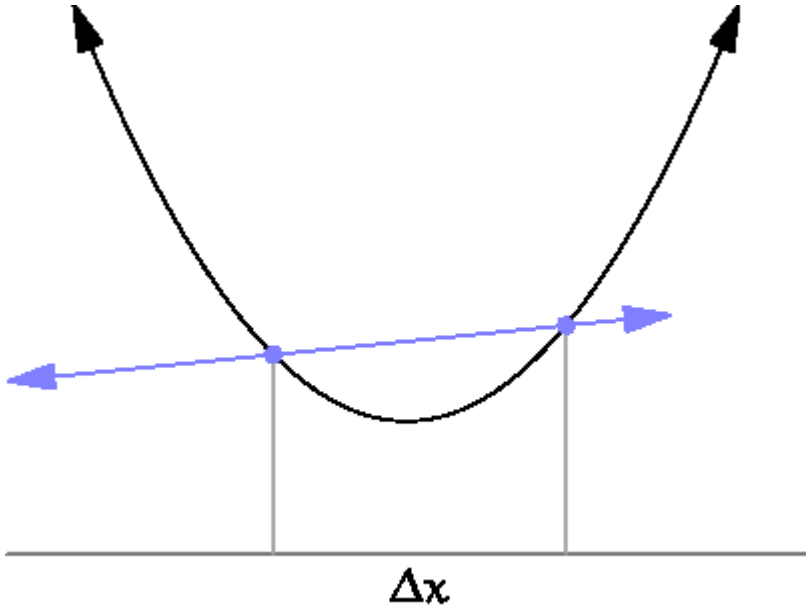


Gradient Descent. What is it?





Gradient Descent. Like a derivative?



Derivative

$$f'(p) = \lim_{\Delta x \to 0} \frac{f(p + \Delta x) - f(p)}{\Delta x}$$

Gradient

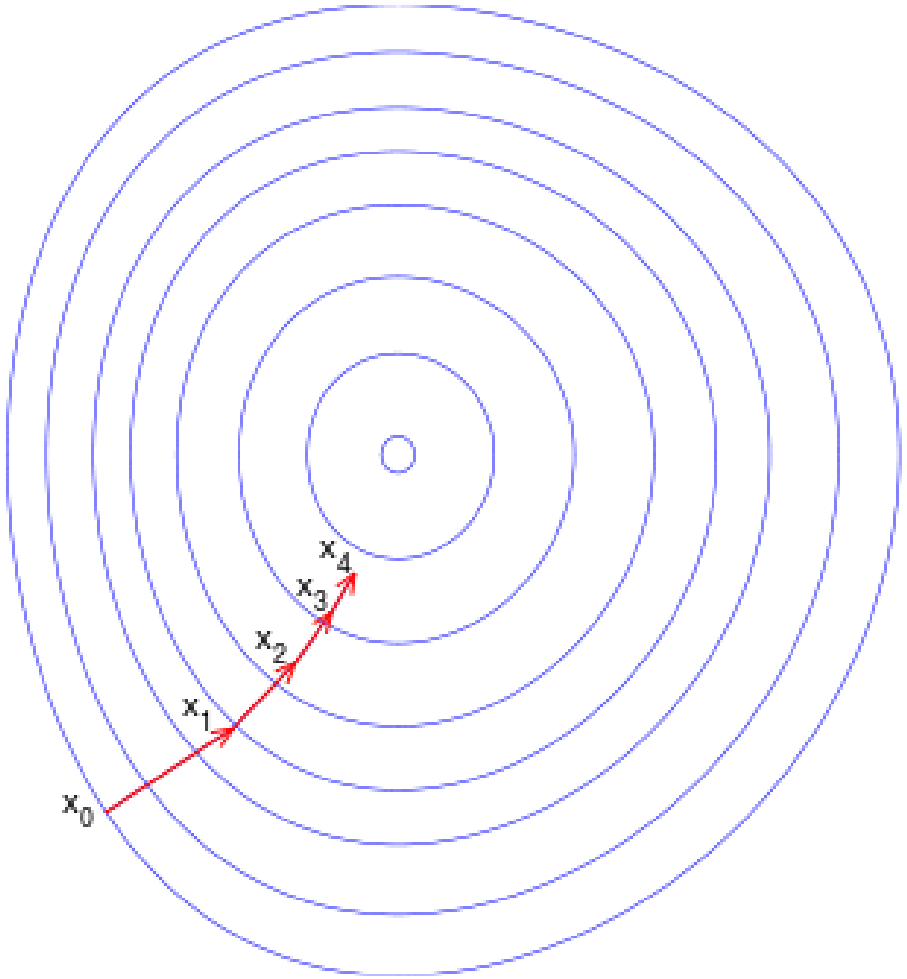
$$\nabla f(p) = \lim_{\Delta x \to 0} \frac{f(p + \Delta x) - f(p)}{\Delta x}$$

Iteration step:

$$p_{i+1} = p_i - L\nabla f(p_i)$$

Where L is the learning step

Gradient Descent. In 2D



Gradient Descent. The maths

Gradient 1D:

$$\nabla f(p) = \lim_{n \to 0} \frac{f(p + \Delta x) - f(p)}{\Delta x}$$

Gradient (for our robot):

$$\nabla f(\alpha_0, \alpha_1, \alpha_2) = \lim_{n \to 0} \frac{f(p + \Delta x) - f(p)}{\Delta x}$$

Which means:

$$\nabla f_{\alpha_0}(\alpha_0, \alpha_1, \alpha_2) = \lim_{n \to 0} \frac{f(\alpha_0 + \Delta\alpha_0, \alpha_1, \alpha_2) - f(\alpha_0, \alpha_1, \alpha_2)}{\Delta\alpha_0}$$

$$\nabla f_{\alpha_1}(\alpha_0, \alpha_1, \alpha_2) = \ \ldots$$

$$\nabla f_{\alpha_2}(\alpha_0, \alpha_1, \alpha_2) = \ \ldots$$

Iteration step:

$$(\alpha_0)_{i+1} = (\alpha_0)_i - L\nabla f_{\alpha_0}(\alpha_0, \alpha_1, \alpha_2)$$

$$(\alpha_1)_{i+1} = \cdots$$

$$(\alpha_2)_{i+1} = \cdots$$

Gradient Descent. Implement its calculation

```
Public float
PartialGradient(Vector3 target,
float[] angles, int i){



}
```

EXERCISE: implement function CalculateGradient in InverseKinematics.cs

IK with Gradient Descent. Implement it

Once we have the partialGradient, implementing the function to approach the target is actually straightforward

EXERCISE: implement function AproachTarget in InverseKinematics.cs, using CalculateGradient

EXERCISE: implement Update to check if we should aproach the target or not

Gradient Descent. Potential problems