Burst IK Documentation

Thank you for your interest in our inverse kinematics solution for Unity using the Burst compiler. Before we get started, here are some limitations of the current version:

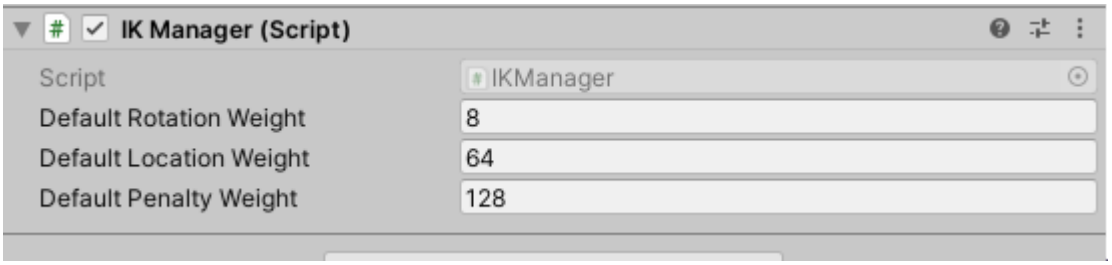
- Calculation weights can only be set globally
- No path planning → needs smooth movement of the ik target
- Single value joints → translation or rotation around one axis at a time
  Workaround: Just use multiple single joints in a hierarchy

Features:

- Inverse kinematics for an arbitrary series of hinge and translation joints
- parallel execution using the Burst compiler
- Support for a vast amount of robot instances (35 six axis robot take less than 0.4ms to compute)

Getting Started:

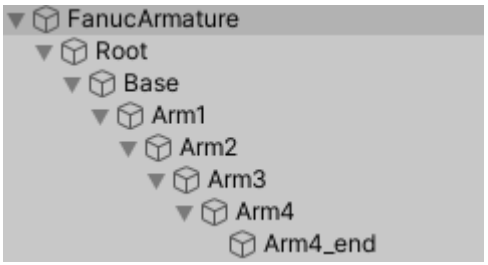
Every scene that needs ik calculations needs exactly one **IKManager** script.



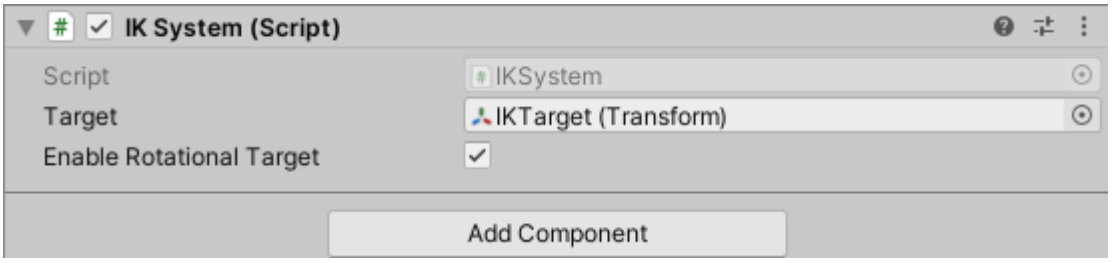
The script has three configurable values, which all correspond to how different properties are treated during calculations:

- **Default Rotation Weight:** How much the rotation of the ik target should be weighted
- **Default Location Weight:** How much the location of the ik target should be weighted
- **Default Penalty Weight:** How much the robot should be penalised for approaching invalid positions (like the limits of angles)

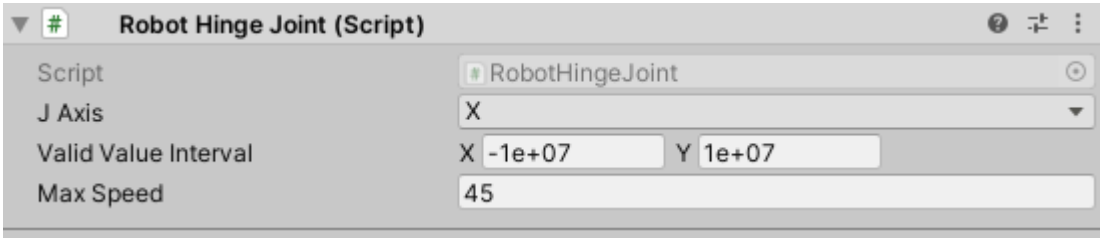
To add a robot ik chain, we need to have a hierarchy like this:



Here the root gets the **IKSystem** script to define a new joint chain. Note how every arm is just a child of the previous one. In the case of the example robot, the arms are implemented as bones in the 3d model.



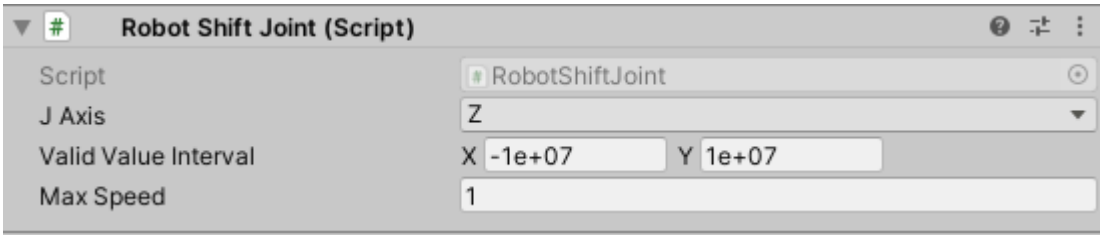
The **IKSystem** script contains information about the Target (some transform in the scene) and if the robot should take the rotation of the target into account. The rotation of the last joint, in our case **Arm4_end** is then aligned with the targets rotation.



Every arm then gets a **Robot Hinge Joint** script.

NOTE: We do not have a fixed joint yet, so the last joint will be used as the end effector. This leads to Hinge joints on **Base, Arm1, Arm2, Arm3, Arm4 and Arm4_end**.

The **Robot Hinge Joint** has
an **axis** value that defines the rotation axis,
a **valid value interval** that defines the limits of the joint and
a **max speed** that defines the max speed of the joint in **deg/s**.



In case you are modelling a machine with translation joints, like a 3D printer or cnc machine, the **Robot Shift Joint** is used for these joints.
The **Axis** value defines the translation axis, the **valid value interval** is analog to hinge joints and
max speed is the maximum translation speed in **m/s**.

Now you have successfully configured a robot. When starting the game view, you can move the corresponding **IKTarget** transform to see the inverse kinematics in action.