IKPy

An Inverse Kinematics toolbox

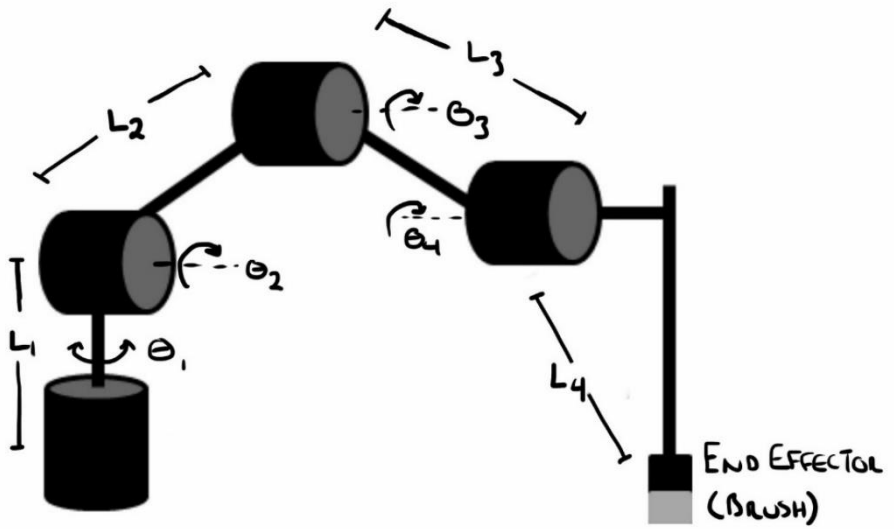
Outline

1. An inverse kinematics toolbox
2. IKPy functionnalities
3. Using IKPy
4. Live Demo

Inverse Kinematics

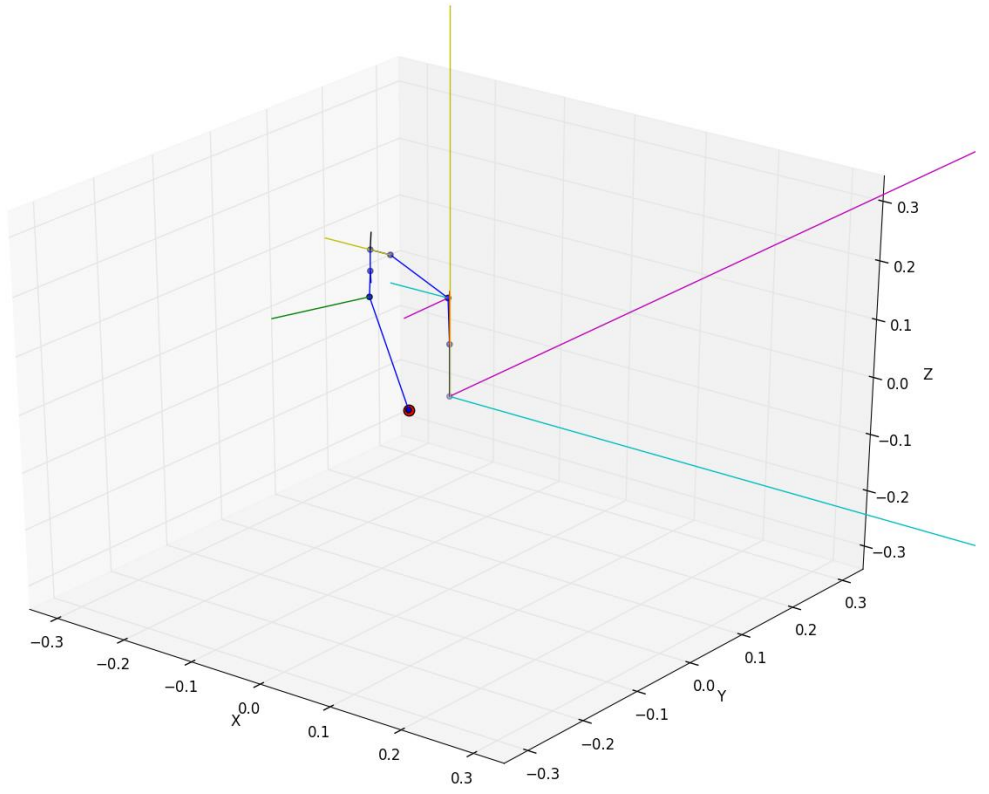
Inverse Kinematics (IK)

- What you have : A robot where you can choose each motor’s angle
- What you want to do : move the end effector to this position
- How to do it : Find a combination of motor angles



Desired functionnalities

- Compatibility :
  - Poppy Humanoid, Torso,
  - Baxter
  - Human
  - Everything else…
- Modularity
- Plotting tools
- Easy to use and setup
- Fast enough



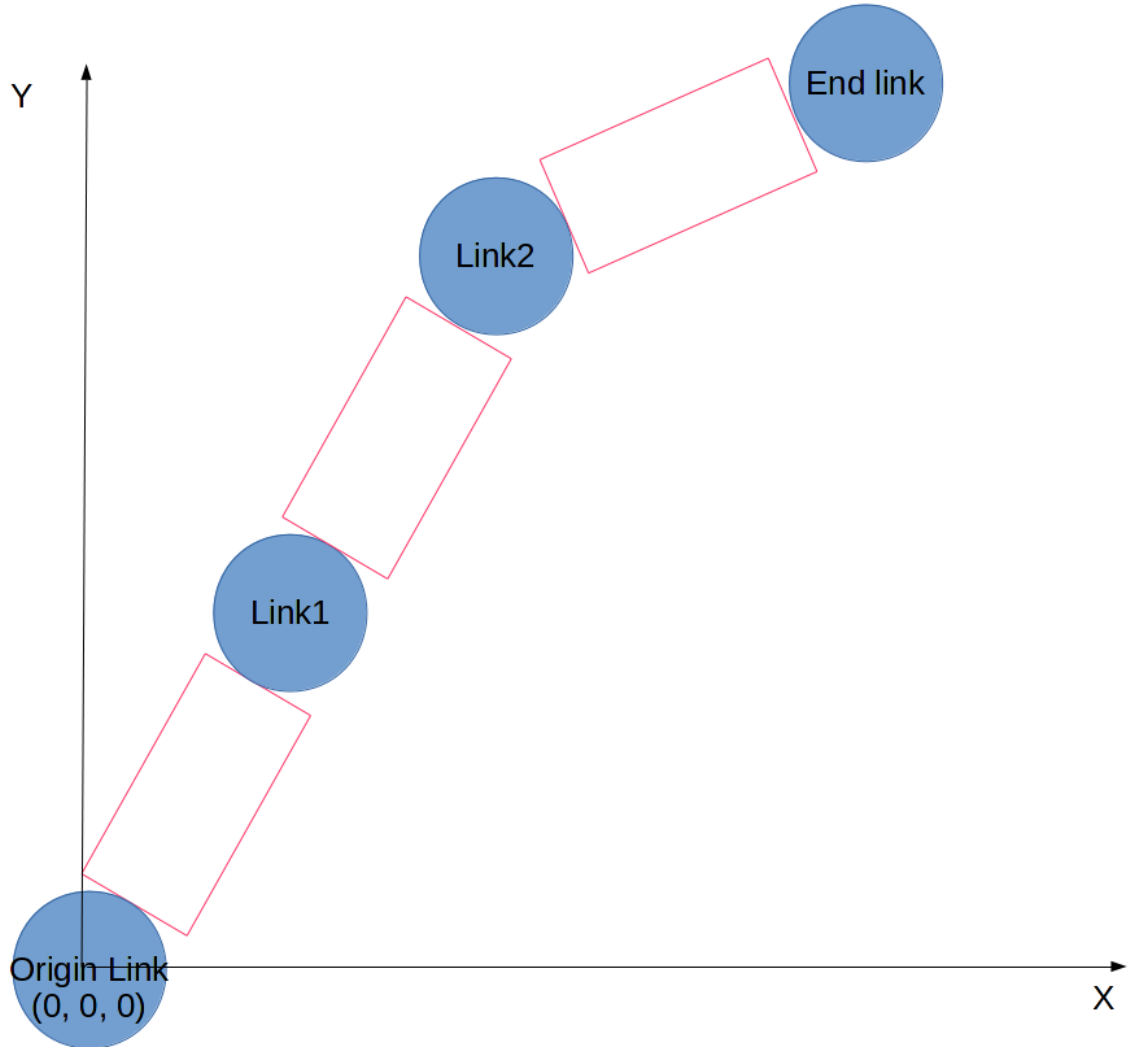
IKPy at a glance

Optimized computations

- Almost based on C structures
- Fast Forward-Kinematics
  - Symbolic geometrical computations
- Possibility to control the computation precision
  - Tradeoff between precision and speed
- Average computation time : 7 ms – 50 ms
- Special optimisations :
  - Multicore? (Work in progress)
  - Dynamic IK

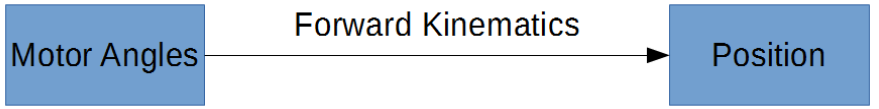
Simple API

- 2 objects :
  - Link ~= A motor
  - Chain = Ordered list of Links

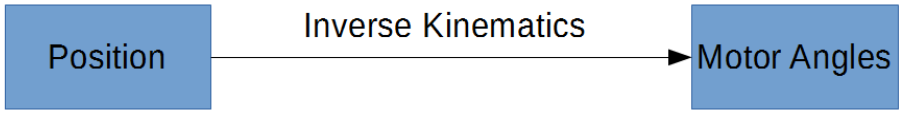


Simple API (2)

- 2 functions :
  - Chain.forward_kinematics



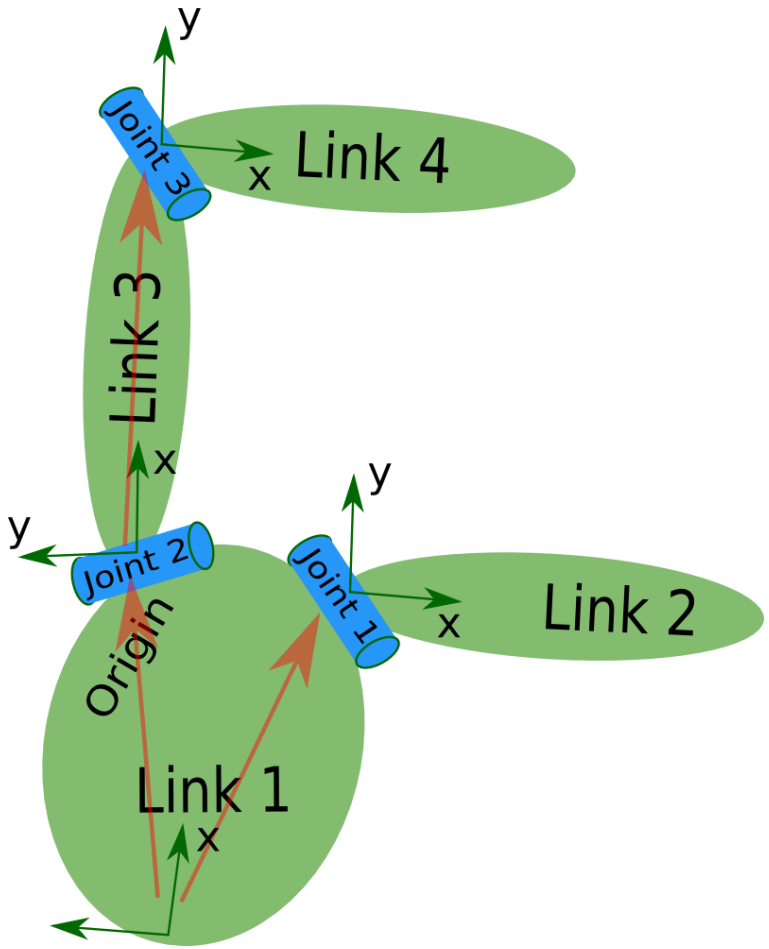
  - Chain.inverse_kinematics



- Additional features :
  - Active links
  - Smoothness control (very related to precision control)

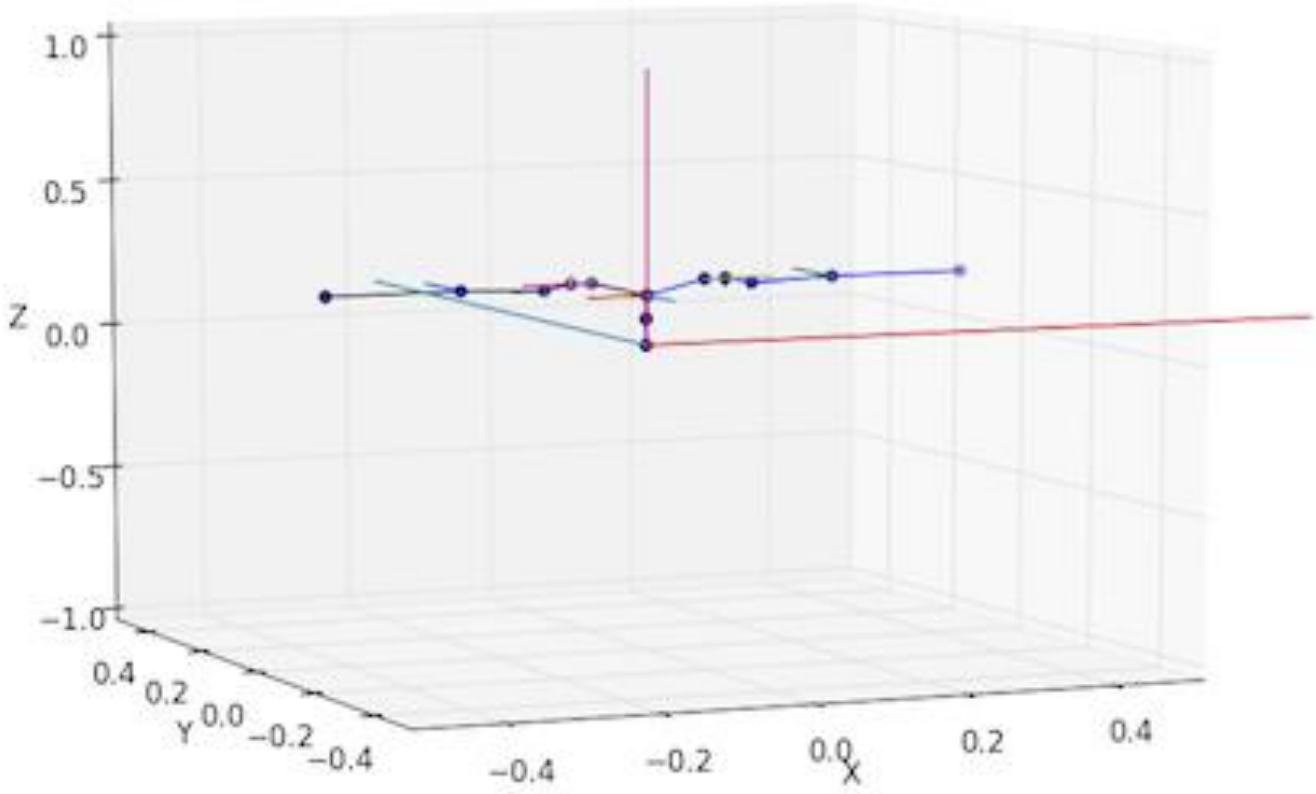
URDF Import

- URDF : XML describing a Robot (as graph)
  - Used by ROS, Poppy
  - Compatible with 3D editors
- Extract a chain from every URDF topology
- Extreme precision : No limitation at the Software level



3D Plotting of the robot

- Method plot() : Matplotlib 3D view
- Standalone use



Using IKPy

A lightweight library

- Easy to install : Pip or Conda
- No compilation required
- Compatibility with most systems :
  - Python 2/3
- Dependencies : Scipy / Numpy, Sympy
- Fast enough for Embedded systems

```
pip install ikpy
```

Developer API

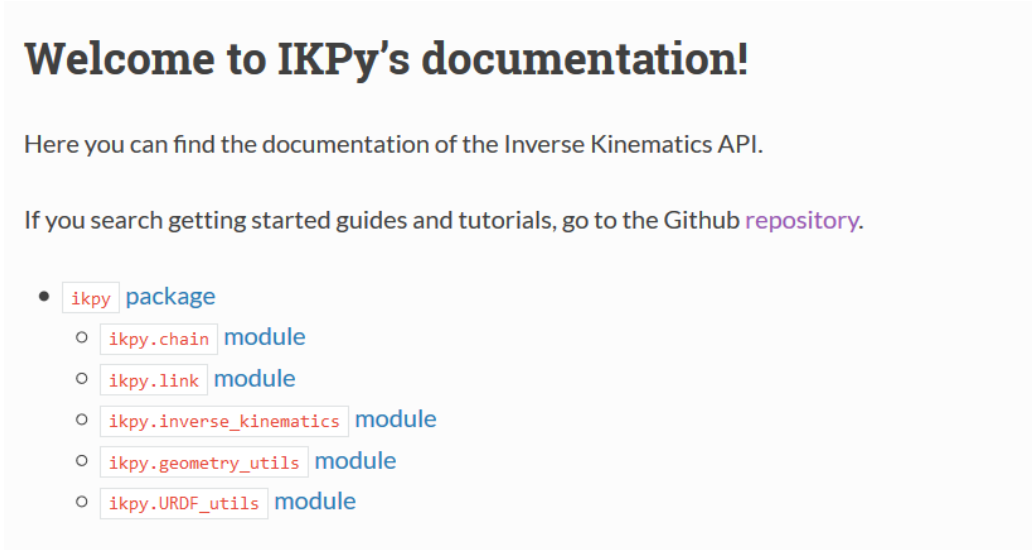
- API to customize IKPy behavior :
- New representations
  - URDF
  - DH parameters
- New Inverse Kinematics methods
  - New constraints : effort, speed
  - New solvers

```python
class OriginLink(Link):
    """The link at the origin of the robot"""
    def __init__(self):
        Link.__init__(self, name="Base link")
        self._length = 1

    def _get_rotation_axis(self):
        return [0, 0, 0, 1]

    def get_transformation_matrix(self, theta):
        return np.eye(4)
```

Last features (1)

- Open source
- Online documentation

GitHub



Welcome to IKPy's documentation!

Here you can find the documentation of the Inverse Kinematics API.

If you search getting started guides and tutorials, go to the Github repository.

- `ikpy` package
  - `ikpy.chain` module
  - `ikpy.link` module
  - `ikpy.inverse_kinematics` module
  - `ikpy.geometry_utils` module
  - `ikpy.URDF_utils` module

Last features (2)

- Tutorials and notebooks

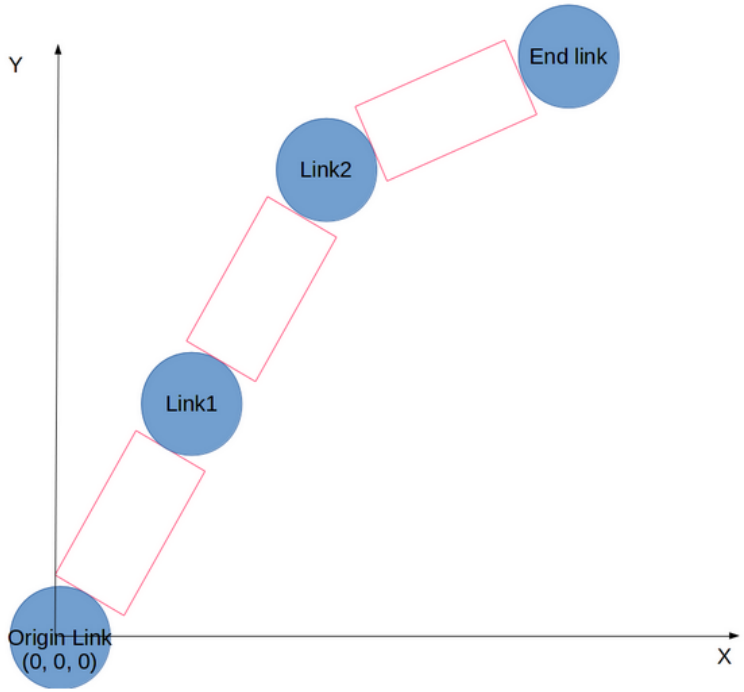
Chain

The `Chain` is the main object of IKpy. It provides the Inverse Kinematics and Forward Kinematics methods, as long as other helper methds (URDF import, plotting...)

Chain structure

A chain is basically a list of `Link` objects. Each link has a position and an orientation, that are stored in its transformation matrix (for more details, go to the links section of the doc). Moreover, each link represents a motor (currently revolute with one degree of freedom), that moves the next links.

Let's consider the 2D robot below :



The first link (called the "Origin" link) is the origin of the frame, at position (0, 0, 0). The last link is the end effector of the chain. Being the last link, its degree of freedom is purely virtual and has no interest : this link can't move anything. It's just here to have a position.

Example : Pypot Integration

- Fully integrated with Pypot
- Chains = attributes of creatures
  - Ex of Torso : torso.right_arm / torso.left_arm
- IK = goto() method

Demo

Live Demo

- Poppy torso : right hand must follow left hand.
- IKPy features : Demo on ErgoJr.